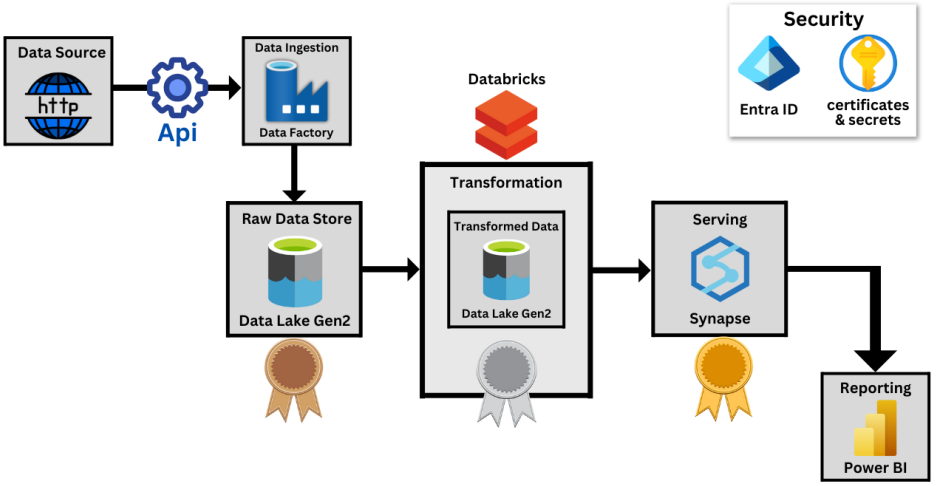
Azure End-to-End Data Pipeline Project - by Md Tarif



Architecture

1. **Data Source:** HTTP (GitHub)
2. **Data Ingestion:** Azure Data Factory
3. **Raw Data Store:** Azure Data Lake Gen 2 (Bronze Layer)
4. **Transformed Data Store:** Azure Data Lake Gen 2 (Databricks) (Silver Layer)
5. **Serving Layer:** Azure Synapse Analytics (Gold Layer)
6. **Reporting:** Power BI
7. **Security:** Azure Entra ID, Azure Certificates, and Secrets
8. **File Format:** CSV, Parquet

Dataset

It contains **10 tables**:

- **AdventureWorks_Calendar.csv**
- **AdventureWorks_Customers.csv**
- **AdventureWorks_Product_Categories.csv**
- **AdventureWorks_Product_Subcategories.csv**
- **AdventureWorks_Products.csv**
- **AdventureWorks_Returns.csv**
- **AdventureWorks_Sales_2015.csv**
- **AdventureWorks_Sales_2016.csv**
- **AdventureWorks_Sales_2017.csv**
- **AdventureWorks_Territories.csv**

Steps

Step 1:

Create an Azure Free Account (provides **USD 200 credit** for 1 month).

Step 2:

Create an **Azure Resource Group**.

Step 3:

Create the first resource under the Resource Group: a **Storage Account (Data Lake Gen 2)**.

Key Points for Configuration:

1. **Data Redundancy:**

   In this project, we use **LRS (Locally Redundant Storage)**.

   - **LRS (Locally Redundant Storage):** Replicates data within a single data center in one region.
   - **ZRS (Zone-Redundant Storage):** Replicates data across multiple availability zones in the same region.
   - **GRS (Geo-Redundant Storage):** Replicates data to a secondary region for disaster recovery.

2. **Enable Hierarchical Namespace:**

   To create a **Data Lake Gen 2**, check the **"Enable hierarchical namespace"** option. If unchecked, it will create a **Blob Storage** instead.

3. **Access Tier:**

   Choose **Hot Tier** for this project.

   - **Hot Tier:** For frequently accessed data.
   - **Cool Tier:** For infrequently accessed data (e.g., monthly).
   - **Archive Tier:** For rarely accessed data (e.g., yearly or longer).

4. **Network Access:**

   Select **"Enable public access from all networks"**.

After completing these steps, your **Data Lake Gen 2** will be created.

Step 4:

Create layers in the **Azure Data Lake Gen 2 Containers**:

- **Bronze Layer** (Raw Data)
- **Silver Layer** (Transformed Data)
- **Gold Layer** (Ready-to-Serve Data)

This structure is known as the **Medallion Architecture**.

Step 5:

Create the second resource: **Azure Data Factory**.

Step 6:

Creating Static Pipeline:

Extract a single file from GitHub (HTTP) and load it into the **Bronze Container** using **Linked Service** and **Copy Activity** in Azure Data Factory.

Step 7:

Creating Dynamic Pipeline:

In real-time scenarios, handling a large number of files (e.g., 10 files in this project) requires a **Dynamic Pipeline** instead of a Static Pipeline.

Key Steps for Dynamic Pipeline:

1. **Dynamic Parameters:**

   Define these parameters to make the pipeline dynamic:

   - **Relative URL** (Source)
   - **Folder** (Sink)
   - **File** (Sink)

2. **JSON File for Dynamic Input:**

   Upload a JSON file to another container in **ADLS Gen 2**, which lists all the file details. This will serve as a loop input.

   Example JSON structure:

```
[
    {
        "p_rel_url": "mdntarif/Azure-End-To-End-Project_1/blob/main/Dataset/AdventureWorks
        "p_sink_folder": "AdventureWorks_Product_Categories",
        "p_sink_file": "AdventureWorks_Product_Categories.csv"
    },
    {
        "p_rel_url": "mdntarif/Azure-End-To-End-Project_1/blob/main/Dataset/AdventureWorks
        "p_sink_folder": "AdventureWorks_Calendar",
        "p_sink_file": "AdventureWorks_Calendar.csv"
    }
    ...
]
```

   git.json

3. **Parameterized Copy Activity:**

   Create a **Copy Activity** with dynamic parameters for the source and sink, and include it in a **ForEach Activity**.

4. **Lookup Activity:**

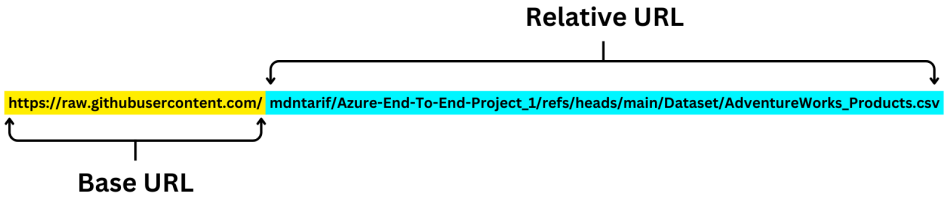
   Use a **Lookup Activity** to retrieve data from the JSON file. The pipeline structure becomes:

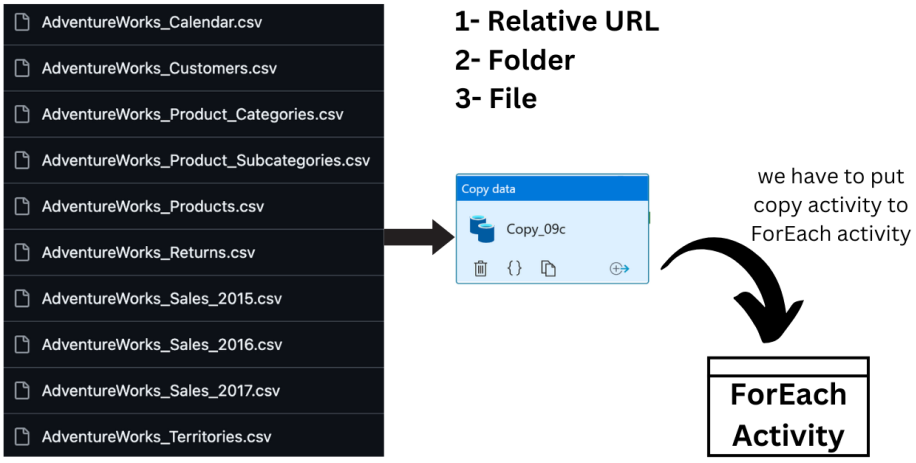
   Lookup → ForEach (containing Copy Activity).

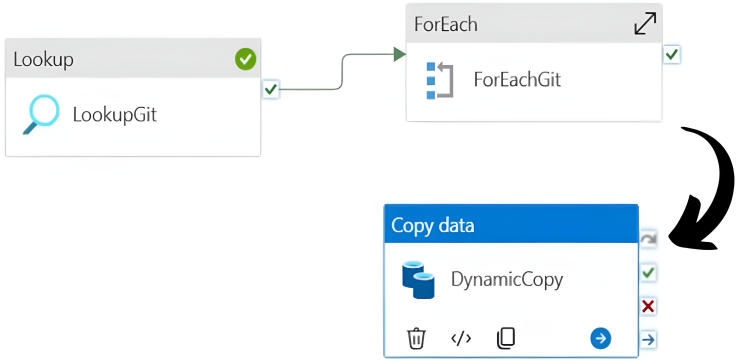
Result:

After debugging the pipeline, all files will be stored in the **Bronze Container** in **ADLS Gen 2**.

This **Dynamic Pipeline** is a key feature of this project, saving **over 70% of time** in real-world scenarios. It simplifies the migration of on-premise data to the cloud with minimal time and effort.







Step 8:

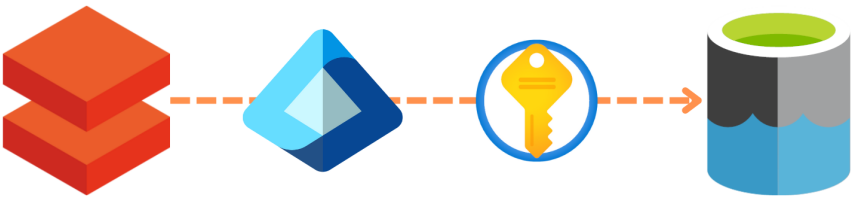
Create the third resource: **Azure Databricks**.

Step 9:

Create a **single-node cluster** in Databricks.

Step 10:

Before working with Databricks, set up permissions to access the data from the Data Lake.

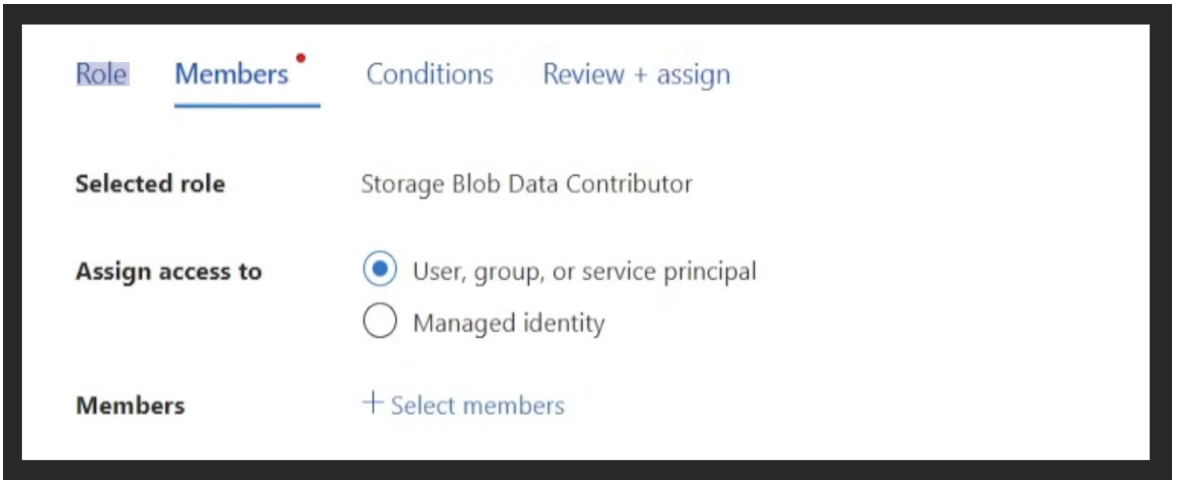


Step 10.1: Application Registration

1. Go to **Microsoft Entra ID** → **App Registrations** → Register an application.
2. Note the **Application (Client) ID** and **Directory (Tenant) ID**.
3. Go to **Certificates and Secrets** → Add a **new client secret** and note the **Value**.

Step 10.2: Role Assignment in Storage Account

1. Navigate to the **Storage Account** → **Access Control (IAM)** → Add **Role Assignment**:
   - Role: **Storage Blob Data Contributor**
   - Assign access to: **User, group, or service principal** (not **Managed Identity**).
2. Select the previously created application from Step 10.1.



Go to **Databricks** and create a notebook. Follow this documentation and use the following code to connect to the Data Lake:

```
spark.conf.set("fs.azure.account.auth.type.<storage-account>.dfs.core.windows.net", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type.<storage-account>.dfs.core.windows.net",
spark.conf.set("fs.azure.account.oauth2.client.id.<storage-account>.dfs.core.windows.net", "<
spark.conf.set("fs.azure.account.oauth2.client.secret.<storage-account>.dfs.core.windows.net'
spark.conf.set("fs.azure.account.oauth2.client.endpoint.<storage-account>.dfs.core.windows.ne
```

Replace the placeholders:

- `<storage-account>`: **tarifstoragedatalake**
- `<application-id>`: **Application (Client) ID**
- `<directory-id>`: **Directory (Tenant) ID**
- `service_credential`: **Secret Value** (in double quotes).

Step 11:

Data Loading:

Load each file in the specified format for each dataset. For example, to load the `AdventureWorks_Calendar` table:

```
df_cal = spark.read.format('csv')\
            .option("header", True)\
            .option("inferSchema", True)\
            .load('abfss://bronze@tarifstoragedatalake.dfs.core.windows.net/AdventureWorks_Ca
```

Step 12:

Data Transformation:

Let's start with transforming the **Calendar** table. Initially, the table looks like this:

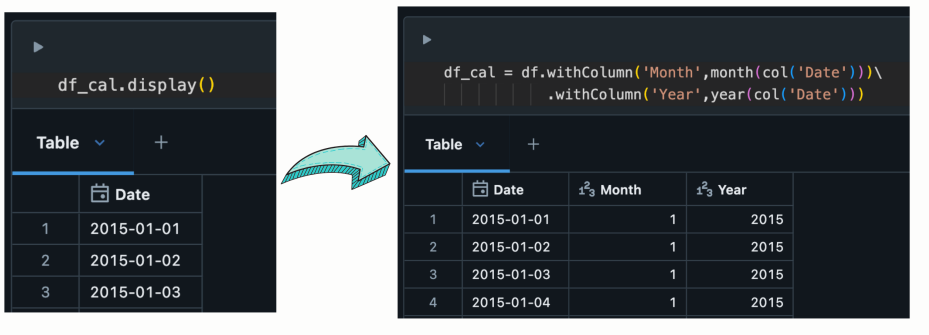
Date
2015-01-14
2015-01-15
2015-01-16

We will add the **Month** and **Year** columns using the following code:

```
df_cal = df.withColumn('Month',month(col('Date')))\
           .withColumn('Year',year(col('Date')))
```

After transformation, the data will look like this:

Date	Month	Year
2015-01-01	1	2015
2015-01-02	1	2015
2015-01-03	1	2015
2015-01-04	1	2015



Step 13:

Loading Transformed Data to Silver Layer:

Now, we will push the transformed **Calendar** data into the **Silver Layer** in **Parquet** format:

```
df_cal.write.format('parquet')\
            .mode('append')\
            .option("path","abfss://silver@tarifstoragedatalake.dfs.core.windows.net/Adventur
            .save()
```

Here, we also become familiar with the **Parquet** file format.

Step 14:

Transformation in Customers Table:

Display the **Customers** table using `df_cus.display()`. The pattern of the table looks like this:

CustomerKey	Prefix	FirstName	LastName	BirthDate	MaritalStatus	Gender	EmailAdd
11000	MR.	JON	YANG	1966-04-08	M	M	jon24@a works.cc
11001	MR.	EUGENE	HUANG	1965-05-14	S	M	eugene1 works.cc
11002	MR.	RUBEN	TORRES	1965-08-12	M	M	ruben35 works.cc

We will create a new column, **fullName**, by combining **Prefix**, **FirstName**, and **LastName**:

```
df_cus = df_cus.withColumn('fullName',concat_ws(' ',col('Prefix'),col('FirstName'),col('lastN
```

After this transformation, the **fullName** column will be added.

Now, let's push this transformed data into the **Silver Layer** in **Parquet** format:

```
df_cus.write.format('parquet')\
            .mode('append')\
            .option("path","abfss://silver@tarifstoragedatalake.dfs.core.windows.net/Adventur
            .save()
```

Step 15:

Transformation in Subcategories Table:

The **Subcategories** table looks like this:

ProductSubcategoryKey	SubcategoryName	ProductCategoryKey
1	Mountain Bikes	1
2	Road Bikes	1
3	Touring Bikes	1
4	Handlebars	2

This table doesn't require any transformations, so we'll directly push the data into the **Silver Layer** in **Parquet** format.

Step 16:

Transformation in Products Table:

The **Products** table has the following structure:

ProductKey	ProductSubcategoryKey	ProductSKU	ProductName	ModelName	ProductDescription	ProductColor
214	31	HL-U509-R	Sport-100 Helmet, Red	Sport-100	Universal fit, well-vented, lightweight, snap-on visor.	Red
215	31	HL-U509	Sport-100 Helmet, Black	Sport-100	Universal fit, well-vented, lightweight, snap-on visor.	Black
218	23	SO-B909-M	Mountain Bike Socks, M	Mountain Bike Socks	Combination of natural and synthetic fibers stays dry and provides just the right cushioning.	White
219	23	SO-B909-L	Mountain Bike Socks, L	Mountain Bike Socks	Combination of natural and synthetic fibers stays dry and provides just the right cushioning.	White

Transformation:

1. We extract the first word before the "-" in the **ProductSKU** column.
2. We extract the first word before the space in the **ProductName** column.

This is done using the `split()` function in Spark:

```
df_pro = df_pro.withColumn('ProductSKU',split(col('ProductSKU'),'-')[0])\
               .withColumn('ProductName',split(col('ProductName'),' ')[0])
```

After transformation, the table looks like this:

ProductKey	ProductSubcategoryKey	ProductSKU	ProductName	ModelName	ProductDescription	ProductColor
214	31	HL	Sport-100	Sport-100	Universal fit, well-vented, lightweight, snap-on visor.	Red
215	31	HL	Sport-100	Sport-100	Universal fit, well-vented, lightweight, snap-on visor.	Black
218	23	SO	Mountain	Mountain Bike Socks	Combination of natural and synthetic fibers stays dry and provides just the right cushioning.	White
219	23	SO	Mountain	Mountain Bike Socks	Combination of natural and synthetic fibers stays dry and provides just the right cushioning.	White

Finally, we push this transformed data to the **Silver Layer** in **Parquet** format.

Step 17:

Returns Table:

The **Returns** table looks like this:

ReturnDate	TerritoryKey	ProductKey	ReturnQuantity
2015-01-18	9	312	1
2015-01-18	10	310	1
2015-01-21	8	346	1
2015-01-22	4	311	1

This table needs **no transformation**. We'll directly push this data to the **Silver Layer** in **Parquet** format.

Step 18:

Territories Table:

The **Territories** table looks like this:

SalesTerritoryKey	Region	Country	Continent
1	Northwest	United States	North America
2	Northeast	United States	North America
3	Central	United States	North America
4	Southwest	United States	North America

Like the **Returns** table, this table needs **no transformation**. We will push it directly into the **Silver Layer** in **Parquet** format.

Step 19:

Sales Table:

The **Sales** table has this structure:

OrderDate	StockDate	OrderNumber	ProductKey	CustomerKey	TerritoryKey	OrderLineItem	OrderQu
2017-01-01	2003-12-13	SO61285	529	23791	1	2	2
2017-01-01	2003-09-24	SO61285	214	23791	1	3	1
2017-01-01	2003-09-04	SO61285	540	23791	1	1	1

2017-01-01	2003-09-28	SO61301	529	16747	1	2	2

Transformation:

1. Change the **StockDate** column's format from **date** to **timestamp**:

```
df_sales = df_sales.withColumn('StockDate',to_timestamp('StockDate'))
```

2. Replace the letter "S" with "T" in the **OrderNumber** column:

```
df_sales = df_sales.withColumn('OrderNumber',regexp_replace(col('OrderNumber'),'S','T'))
```

3. Create a new column, **multiply**, by multiplying **OrderLineItem** and **OrderQuantity**:

```
df_sales = df_sales.withColumn('multiply',col('OrderLineItem')*col('OrderQuantity'))
```

After these transformations, the **Sales** table looks like this:

OrderDate	StockDate	OrderNumber	ProductKey	CustomerKey	TerritoryKey	OrderLineItem
2017-01-01	2003-12-13T00:00:00.000+00:00	TO61285	529	23791	1	2
2017-01-01	2003-09-24T00:00:00.000+00:00	TO61285	214	23791	1	3
2017-01-01	2003-09-04T00:00:00.000+00:00	TO61285	540	23791	1	1
2017-01-01	2003-09-28T00:00:00.000+00:00	TO61301	529	16747	1	2

Then, we push this transformed data to the **Silver Layer** in **Parquet** format using:

```
df_sales.write.format('parquet')\
            .mode('append')\
            .option("path","abfss://silver@tarifstoragedatalake.dfs.core.windows.net/Adventur
            .save()
```

Step 20:

Analysis and Visualization in the Sales Table:

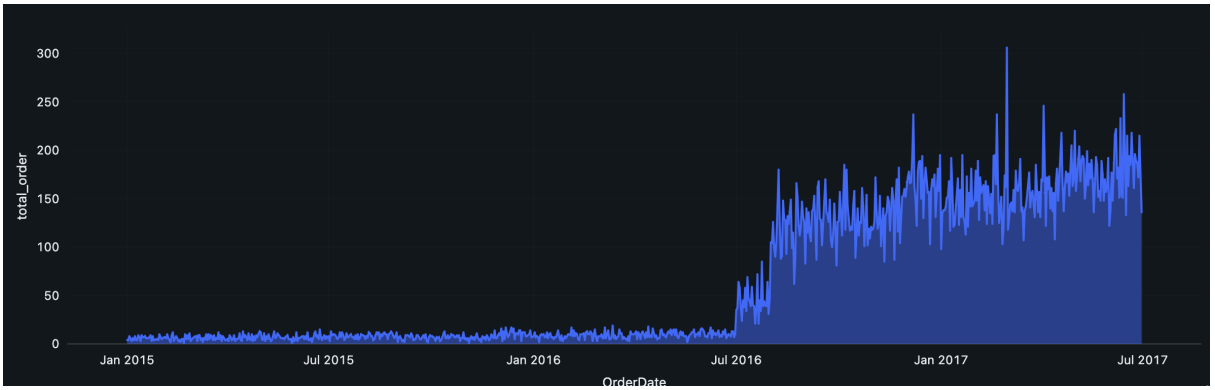
We perform an analysis to find how many orders were placed on each order date:

```
df_sales.groupBy('OrderDate').agg(count('OrderNumber').alias('total_order')).display()
```

The result is:

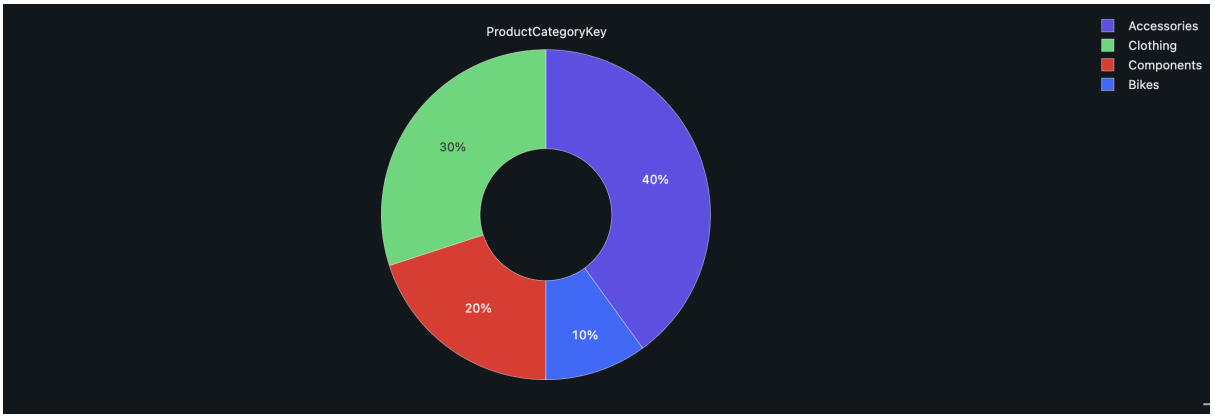
OrderDate	total_order
2017-01-06	151
2017-01-27	142
2017-02-26	119
2017-01-24	173

We can also visualize the data using Databricks' built-in charting options.



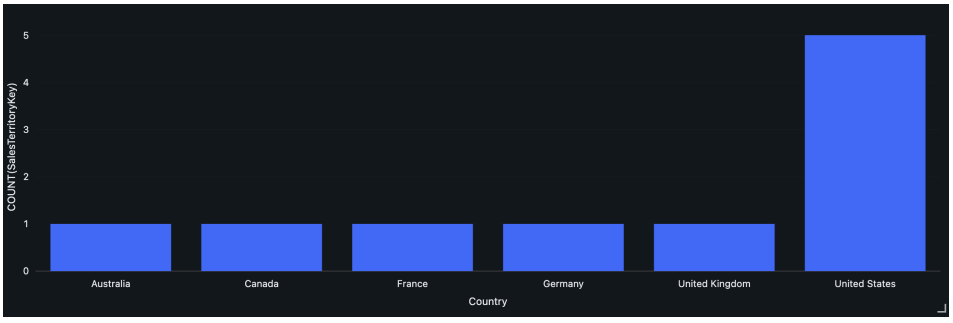
Step 21:

You successfully created a **Pie chart** for the `Product Table` by displaying the `df_procat` dataframe and using the visualization options in Databricks. This allows you to quickly analyze the distribution of product categories.



Step 22:

Next, you created a **Bar chart** for the `Territory Table` by displaying the `df_ter` dataframe and clicking the visualization option in Databricks. The bar chart will help visualize the different territories, aiding in business decisions.



Step 23:

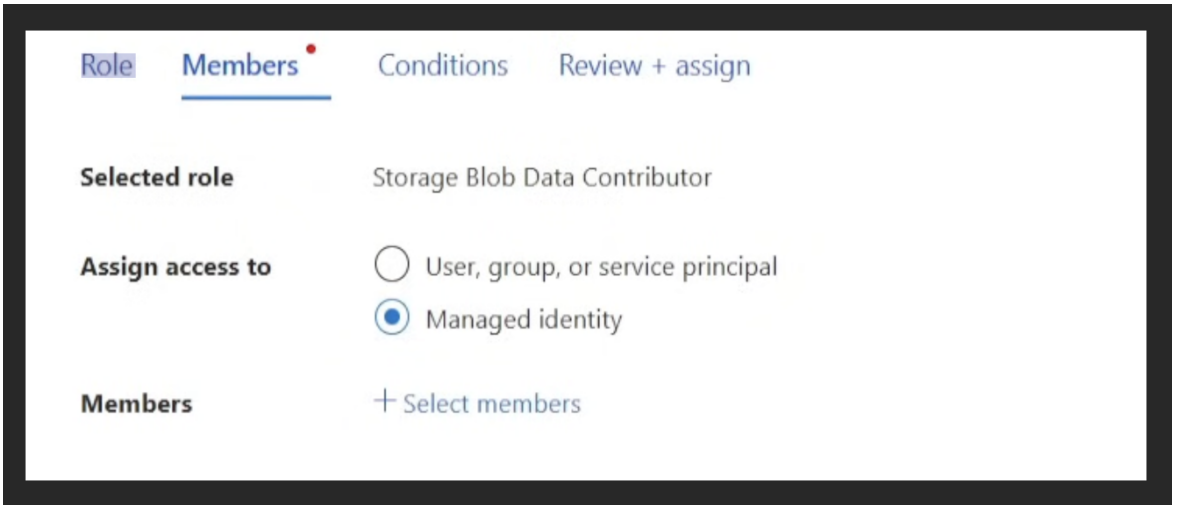
You've moved on to working with **Azure Synapse Analytics**:

1. First, create the **Azure Synapse Analytics resource**.
2. It's important to create the **default ADLS Gen2** during the creation of this resource.
3. Set the **admin login & password** for the SQL pool.
4. Disable **virtual network**.

Step 24:

After creating the resource:

1. Go to the **Resource Group**.
2. Select **external data lake** (not the new data lake created in Synapse).
3. Navigate to **Access Control (IAM)**.
4. Assign the role `Storage Blob Data Contributor` to the **Synapse Managed Identity**.
5. In the **Assign access to** section, choose **Managed identity**, not **User, group, or service principal**.
6. In **Select member**, choose the **Synapse workspace** that was already created.



Step 25:

You then went to **Synapse** and started setting up the database:

1. In the **Develop section**, select **SQL script**.
2. Go to the **Data section** and select **SQL Database**.
3. Choose **Serverless SQL pool** (not Dedicated SQL pool).
4. Create the database.

This setup will allow you to query data without needing a dedicated SQL pool, offering flexibility and cost savings.

Step 26:

You've assigned the necessary permissions again for the **external data lake**:

1. Go to the **Resource Group** and select **external data lake**.
2. Under **Access Control (IAM)**, assign the role `Storage Blob Data Contributor`.
3. Select **User, group, or service principal**.
4. Choose the **Synapse workspace** as the member, then **Review + Assign**.

This ensures that the required permissions are in place for accessing the data in Azure Synapse.

Step 27:

Now, you created a **SQL View** in **Azure Synapse Analytics** using the `OPENROWSET()` function to access Parquet files stored in an Azure Data Lake. Here's the SQL syntax you used:

```
CREATE VIEW gold.calendar
AS
SELECT
    *
FROM
    OPENROWSET
        (
```

```
            BULK 'https://tarifstoragedatalake.blob.core.windows.net/silver/AdventureWorks_Ca
            FORMAT = 'PARQUET'
        ) as QUER1
```

Gold_Create_Views.sql

You used this to create views for all the necessary files, making it easier for everyone (managers, data analysts, etc.) to access the data.

Step 28:

You're now working on **External Tables** in **Synapse Analytics**. To do that, you need to create:

1. **Credentials**: Save login details to access the data.
2. **External Data Source**: Points to where the data is stored (like in Azure Data Lake).
3. **External File Format**: Defines the type of file (e.g., Parquet, CSV).

Microsoft's Azure documentation has predefined scripts to help with these tasks.

Step 29:

For **Credentials**:

1. First, you created a **Master Key** for encryption (to protect sensitive data) with:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'password'
```

This is just to set up the encryption at the database level.

2. You created a **database-scoped credential** using:

```
CREATE EXTERNAL DATA SOURCE source_silver
WITH
(
LOCATION = 'https://tarifstoragedatalake.blob.core.windows.net/silver',
CREDENTIAL = cred_tarif
)
```

This credential uses the **Managed Identity** to access resources securely.

Step 30:

For **External Data Sources**:

1. You created the **silver data source**:

```
CREATE EXTERNAL DATA SOURCE source_silver
WITH
(
LOCATION = 'https://tarifstoragedatalake.blob.core.windows.net/silver',
CREDENTIAL = cred_tarif
)
```

2. You created the **gold data source**:

```
CREATE EXTERNAL DATA SOURCE source_gold
WITH
(
LOCATION = 'https://tarifstoragedatalake.blob.core.windows.net/gold',
```

```
CREDENTIAL = cred_tarif
)
```

This sets up the external data sources, allowing you to query and interact with the data stored in the Azure Data Lake.

Step 31:

For **External File Format**:

You created an external file format using **Parquet** and specified **Snappy Compression** for efficient storage and retrieval:

```
CREATE EXTERNAL FILE FORMAT format_parquet
WITH
(
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
)
```

This ensures that your Parquet files are optimized for compression, which improves performance when querying.

Step 32:

You created an **External Table** in the **Gold layer** of the Data Lake, referencing the `sales` table and using the defined external data source and file format:

```
CREATE EXTERNAL TABLE gold.extsales
WITH
(
    LOCATION = 'extsales',
    DATA_SOURCE = source_gold,
    FILE_FORMAT = format_parquet
)
AS
SELECT * FROM gold.sales;
```

This makes it easier to query the data in the gold layer of your Data Lake directly within Synapse Analytics.

Step 33:

For **Power BI Integration**:

1. You copied the **Serverless SQL Endpoint** from Azure Synapse.
2. In Power BI:
    - Go to **Get Data**.
    - Select **Azure > Azure Synapse Analytics**.
    - Paste the SQL endpoint.
    - Load the `gold.extsales` table.

Now, Azure Synapse Analytics is connected to Power BI, enabling data analysts or data scientists to visualize and analyze the data.