HEXAWARE

TRAINING PROJECT REPORT

INVENTORY MANAGEMENT

OBJECTIVE : Analysis & Reporting System ofInventory Management.

PYTHON BATCH - 2

MEMBERS (TEAM - 14):

→ **Rushmitha Sreeja Kaluvakolan**
→ **Katika Mubeena**

Introduction

This mini-project focuses on developing an **Inventory Management Analysis and Reporting System** using **PySpark**. The project centers on building a data warehouse based on a star or snowflake schema, transforming OLTP source data into OLAP structures, applying data transformations, and generating insightful reports. By leveraging tools like Python and PySpark on the Databricks platform, the project integrates **Microsoft Azure Services** for efficient data storage, processing, and report generation.

The primary objective was to explore the synergy between **Big Data technologies** and **cloud services** for working with large datasets, gaining hands-on experience in implementing a robust data analysis pipeline.

Technologies Used

Programming and Frameworks

- **Python**
- **Apache Spark** (Version 15.4 LTS)

Microsoft Azure Services

1. **Azure Data Lake Storage**: Securely storing large volumes of data.
2. **Azure Databricks**: For data processing and analysis using:
    - Databricks Clusters
    - Databricks Catalog
    - Databricks Workspace
3. **Azure SQL Server**: Managing and querying structured data.
4. **SQL Database**: Facilitating operations on structured datasets.
5. **Azure Data Factory**: Automating data workflows and orchestrations.

This setup enabled efficient data analysis and processing while providing practical experience with cloud-based tools essential for modern data projects.

Abstract

In today’s competitive business landscape, efficient inventory management is vital for optimizing operational performance and customer satisfaction. This project focuses on developing an **Inventory Management System** using dimensional modeling to enable seamless data analysis, reporting, and decision-making. The system is designed to handle and integrate large-scale transaction data with key dimensions, such as customers, products, sellers, and time, into a structured schema for business intelligence and analytics.

The project architecture comprises a **central fact table** that records transactional details, including sales, costs, and profits, linked to multiple **dimension tables**—Time, Customers, Products, Sellers, and Transactions—through carefully designed relationships. Data from various sources is ingested into an Azure-based ecosystem leveraging **Azure Databricks** for processing and transformation. The **Time Table** provides a comprehensive breakdown of dates, facilitating insights into sales trends across time dimensions, such as years, months, and quarters.

This system utilizes **PySpark** for data cleaning, transformation, and ETL (Extract, Transform, Load) processes, ensuring scalability and efficiency. Reports such as **top customers by transaction value**, **state-wise transaction summaries**, and **seller performance metrics** are generated and stored back into Azure Data Lake in accessible formats (CSV/Parquet). The project integrates robust error handling mechanisms to address missing data, failed relationships, and performance bottlenecks.

By implementing this inventory management system, organizations can achieve improved data visibility, generate actionable insights, and support strategic decision-making, ultimately enhancing overall business performance. The   solution serves as a foundation for scalable analytics in inventory control and sales optimization.

DIMENSION TABLES

Customer Dimension

Fields in the Customer Dimension Table:

- CUSTOMER_ID
- Customer Name
- CUSTOMER_LOGIN_ID
- CUSTOMER_STREET_ADDRESS
- CUSTOMER_CITY
- CUSTOMER_STATE
- CUSTOMER_ZIP
- CUSTOMER_PHONE_NO.

Products Dimension

Fields in the Products Dimension Table:

- PRODUCT_ID
- CATEGORY_ID
- PRODUCT Name
- PRODUCT Brand
- Product Model No.
- PRODUCT_STOCK

SELLERS Dimension

Fields in the SELLERS Dimension Table:

- SELLER_ID
- SELLER_NAME

- SELLER_RATING
- SELLER_STREET_ADDRESS
- SELLER_CITY
- SELLER_STATE
- SELLER_ZIP
- SELLER_PHONE_NO.

TIME Dimension

Fields in the Time Dimension Table:

- date_key
- full_date
- day_of_week
- month_name
- quarter
- fb_year

TRANSACTIONS Dimension

Fields in the Transactions Dimension Table:

- TRANSACTION_ID
- TRANSACTION_DATE
- TRANSACTION_AMOUNT
- TRANSACTION_TYPE
- DISPATCH_DATE
- EXPECTED_DATE
- DELIVERY_DATE

FACT TABLE

Fields in the Fact Table:

- TRANSACTION_ID
- Date
- PRODUCT_ID
- CUSTOMER_ID
- SELLER_ID
- PRODUCT_COST_PRICE
- PRODUCT_SELLING_PRICE

- SELLER_ID
- SELLER_NAME
- SELLER_RATING
- SELLER_STREET_ADDRESS
- SELLER_CITY
- SELLER_STATE
- SELLER_ZIP
- SELLER_PHONE_NO.

TIME Dimension

Fields in the Time Dimension Table:

- date_key
- full_date
- day_of_week
- month_name
- quarter
- fb_year

TRANSACTIONS Dimension

Fields in the Transactions Dimension Table:

- TRANSACTION_ID
- TRANSACTION_DATE
- TRANSACTION_AMOUNT
- TRANSACTION_TYPE
- DISPATCH_DATE
- EXPECTED_DATE
- DELIVERY_DATE

REPORTS TO BE BUILT

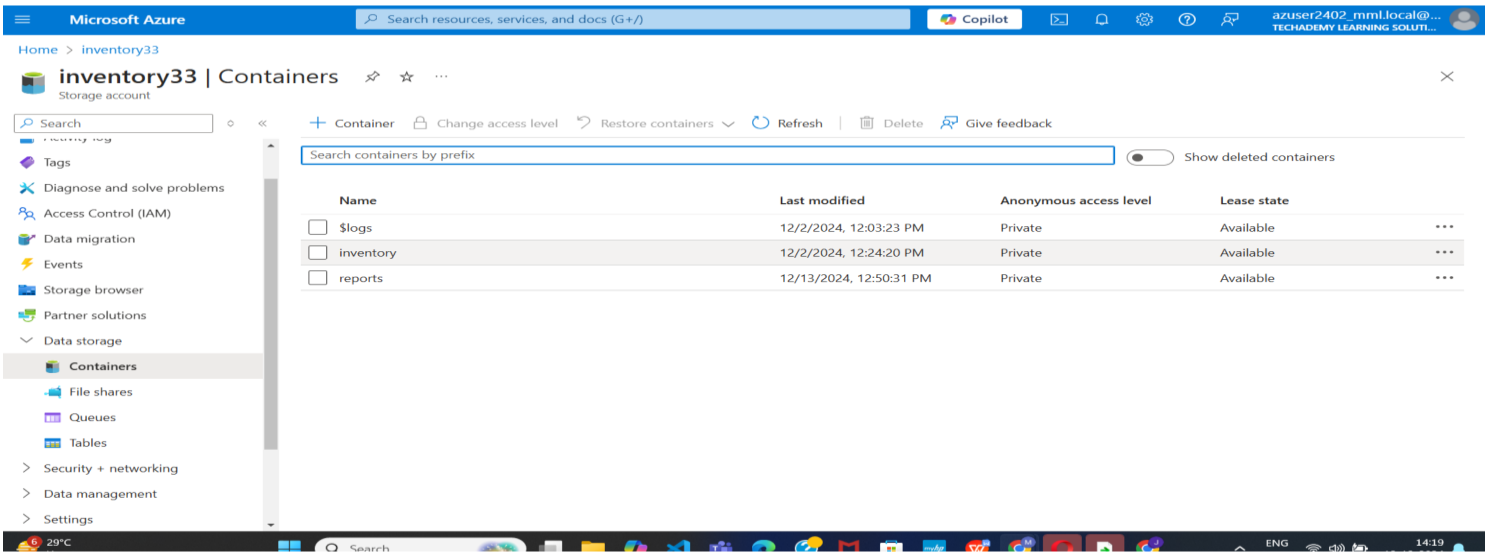
- Transaction Amount Wise Top 10 Customers
- State wise number of orders
- Preferred mode of payment across states
- Quarterly profit for a particular year
- Quarterly sales count of each product category

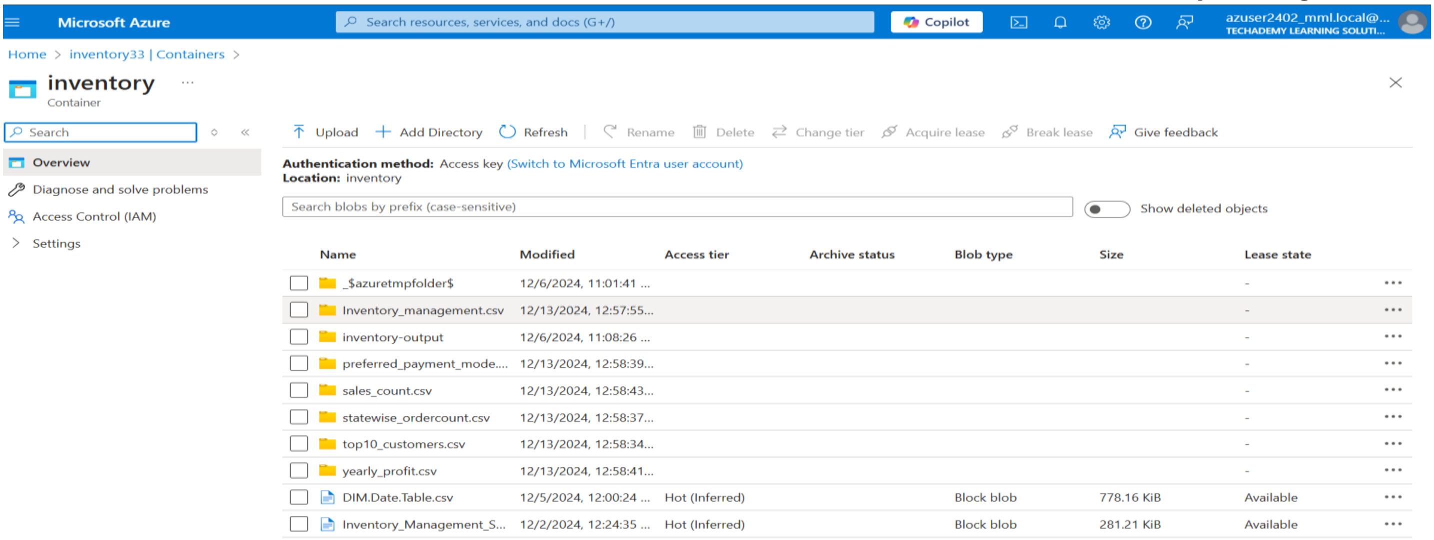
Workflow: Azure Data Lake Storage Setup

To efficiently manage data, we implemented Azure Data Lake Storage and structured it into two organized containers:

Inventory

- This container holds the raw source data in CSV format, which forms the foundation of the project.
- It includes critical datasets such as transactions, customers, and products, used for data processing and transformation.
- These files serve as the primary input for performing analytics and building the ETL pipeline.

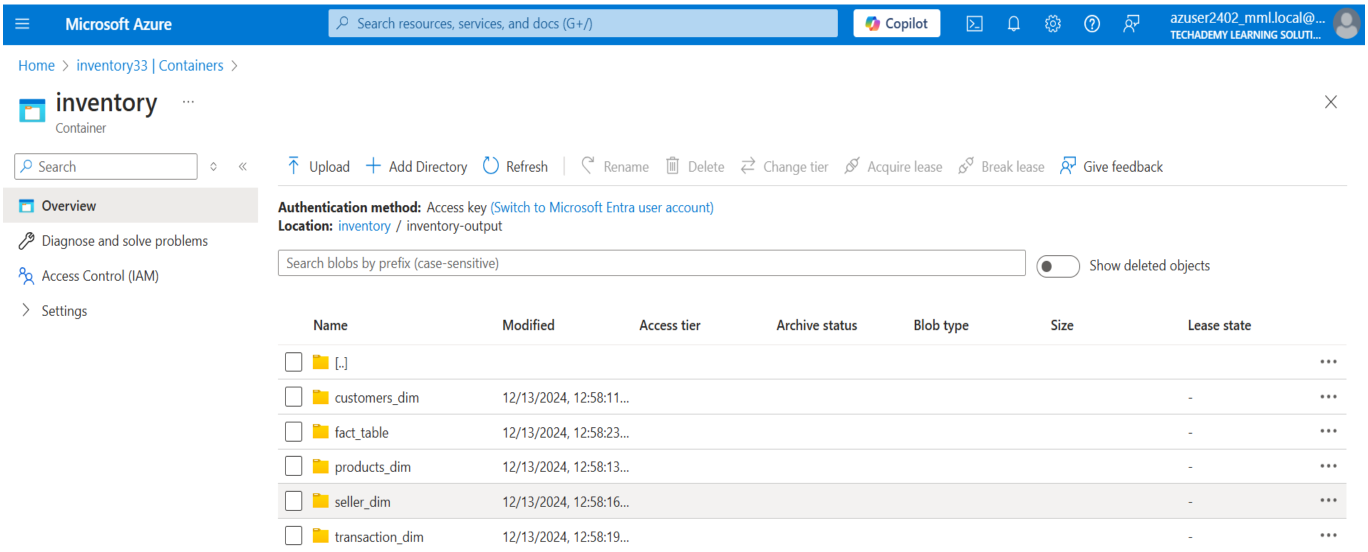




Output:

This container stores the transformed and processed CSV filesgenerated as output.

These output files are the results of our analysis and transformations.

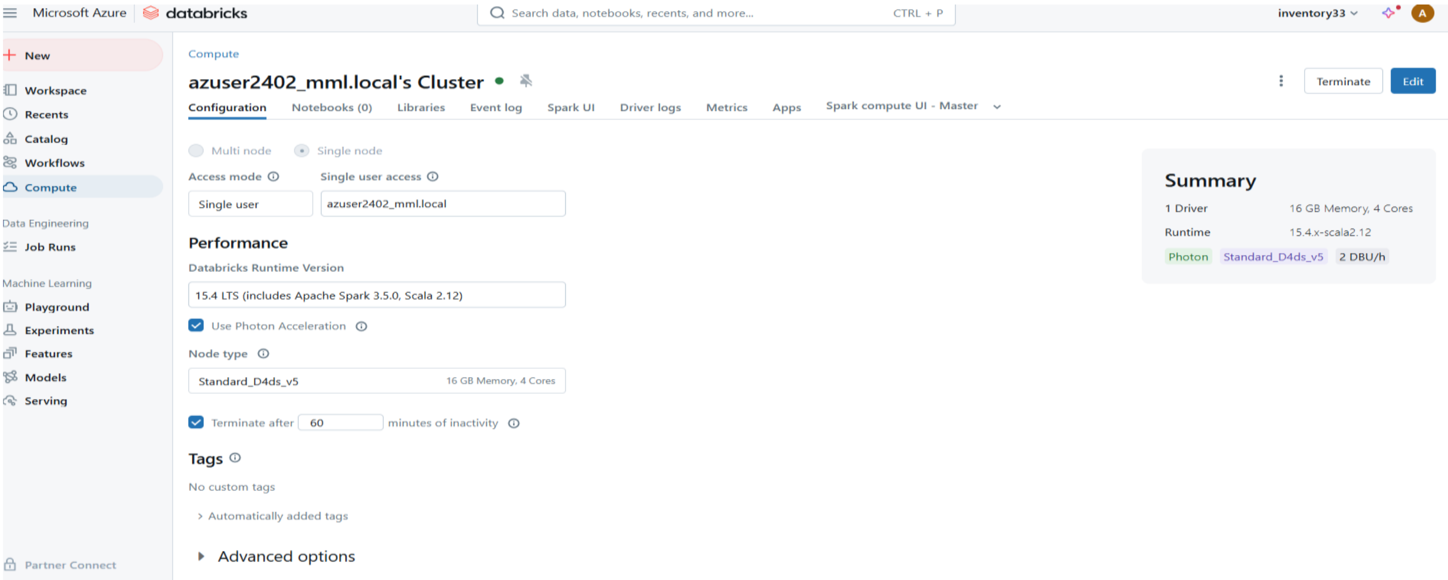


Azure Databricks Workspace

In the **Azure Databricks Workspace**, the following steps were performed:

1. **Cluster Creation**:
   - A **single-node cluster** was created to run and manage the notebooksefficiently.
   - This cluster served as the processing environment for

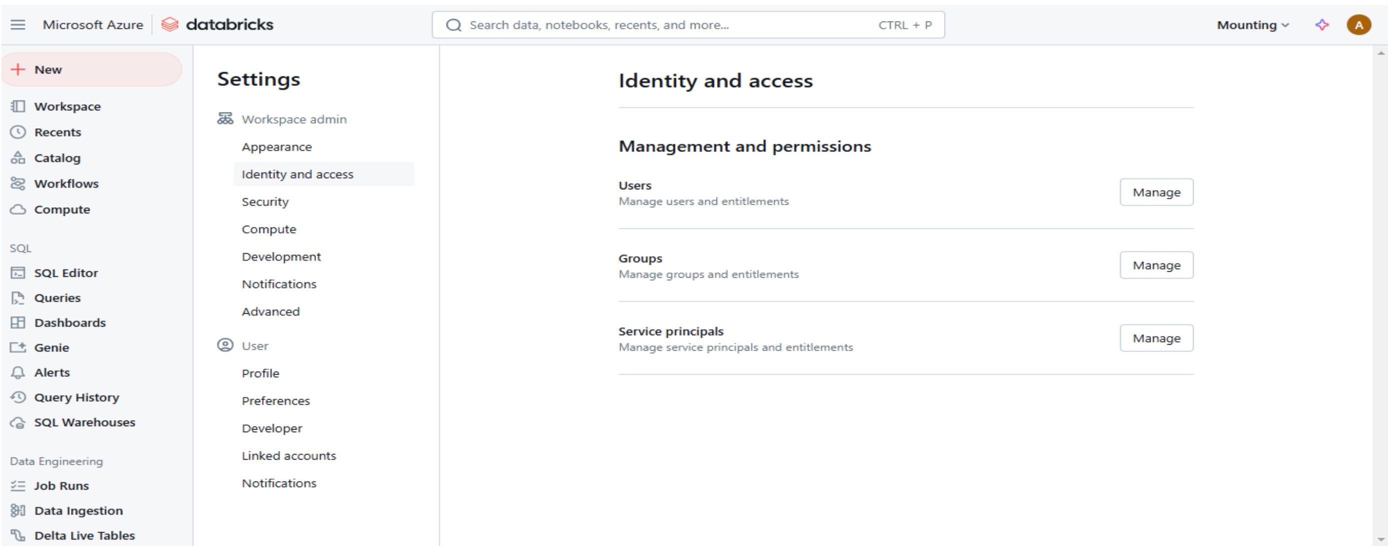
performingtransformations and analysis on the data.

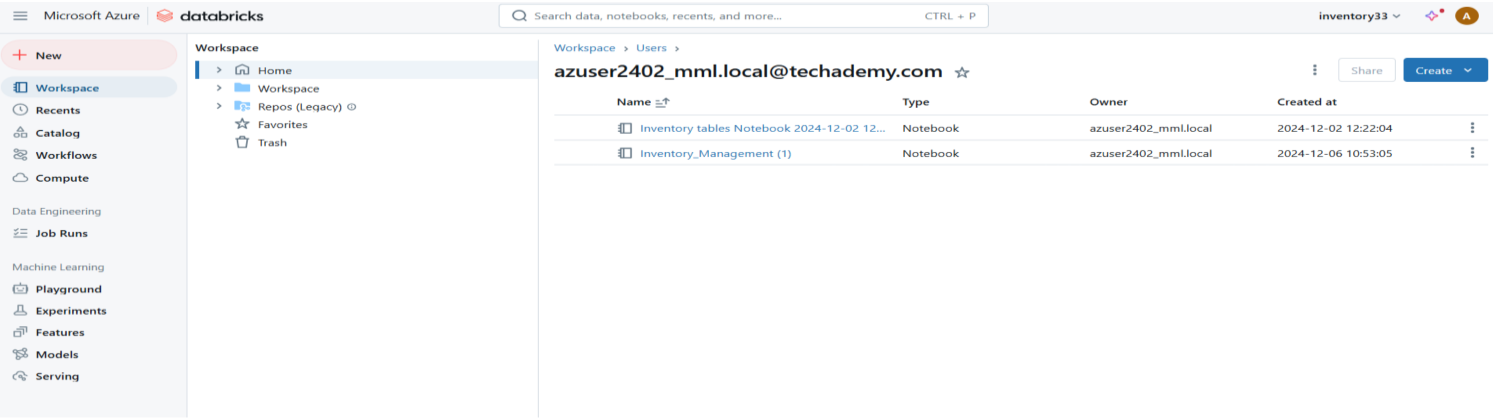


2. **Notebook Organization**:

A dedicated folder was created within the workspace to store all the notebooks.

These notebooks contained the PySpark code and other scripts used for dataprocessing and visualization.





Mounting the Azure Data Lake Storage (ADLS) Account

To facilitate seamless access to the data stored in Azure Data Lake Storage (ADLS), the storage containers were mounted to the Databricks File System (DBFS) using the Account Key authentication method.

Mounting Containers

Input Container

1. Serves as the source for accessing raw data files, such as CSV files, required for transformations and data processing.
2. Enables Databricks to efficiently read and process unstructured and semi-structured data.

Output Container

1. Acts as the destination for storing processed and transformed data.
2. Includes results like reports, aggregated datasets, and intermediate outputs, ensuring data is well-organized for further use.

By mounting these containers, the integration between Databricks and ADLS ensures a smooth and efficient data pipeline for transformation and analysis

Mount input container(Inventory)

```
12:57 PM (11s)                                   4                              Python

dbutils.fs.mount(
    source = "wasbs://inventory@inventory33.blob.core.windows.net",
    mount_point = "/mnt/inventory",
    extra_configs = {"fs.azure.account.key.inventory33.blob.core.windows.
    net":"OyFNfd0A5TkzgjbBgldfqqEXEpAdnDYK6zTGxIZ1sebVfjY4lCTeaZhkH4d2sa9kNqMgZyU0qDR9+ASth4M68A=="})

True
```

+ Code   + Text

Mounting Container From ADLS for Output

```
12:57 PM (11s)                                   6

dbutils.fs.mount(
    source="wasbs://inventory@inventory33.blob.core.windows.net/",
    mount_point="/mnt/reports",
    extra_configs={"fs.azure.account.key.inventory33.blob.core.windows.net":
    "OyFNfd0A5TkzgjbBgldfqqEXEpAdnDYK6zTGxIZ1sebVfjY4lCTeaZhkH4d2sa9kNqMgZyU0qDR9+ASth4M68A=="}
)
```

PRE-PROCESSING

Read the data from Mount Point

Markdown

```
12:57 PM (9s)                                    9                              Python

df = spark.read.csv('dbfs:/mnt/inventory/Inventory_Management_Src1.csv', header=True, inferSchema=True)

(2) Spark Jobs
df:  pyspark.sql.dataframe.DataFrame = [Date: string, TRANSACTION_ID: integer ... 30 more fields]
```

```
12:57 PM (4s)                                    10

df.write.format('parquet').mode('overwrite').save('/mnt/inventory/Inventory_management.csv',header=True)

(1) Spark Jobs
```

```
12:57 PM (1s)                                    11

df.show()

(1) Spark Jobs
+---------+--------------+-----------------+------------------+------------------+----------------+--------------+---------------+-----------+
```

Correcting the date format

12:57 PM (<1s) 26

```
from pyspark.sql.functions import col, year, quarter, month, dayofweek, monotonically_increasing_id
```

12:57 PM (1s) 27 Python

```
df2 = spark.read.csv('dbfs:/mnt/inventory/DIM.Date.Table.csv', header=True, inferSchema=True)
```

(2) Spark Jobs

df2: pyspark.sql.dataframe.DataFrame = [date_key: integer, full_date: timestamp ... 22 more fields]

DISPLAYING THE INPUT DATA

12:57 PM (2s) 17 Python

```python
customers_dim.show(5)
```

(2) Spark Jobs

CustomerID	CustomerName	CustomerLoginID	CustomerStreetAddress	CustomerCity	CustomerState	CustomerZip	CustomerPhoneNo
100088	Gisela Mayer	5088	P.O. Box 321, 480...	SAN DIEGO	CALIFORNIA	60330	023-479-2802
100068	Tana Blevins	5068	Ap #364-8638 Et S...	BOSTON	MASSAACHUSETTS	91583	081-828-4143
100074	Carly Parker	5074	9331 Ipsum. Ave	NEW YORK	NEWYORK	66948	016-408-0204
100003	Fritz Grant	5003	Ap #897-7736 Eges...	PHILADELPHIA	PENNSYLVANIA	65342	029-197-3614
100042	Leroy Pugh	5042	6658 Venenatis Rd.	AUSTIN	TEXAS	63417	050-412-9084

only showing top 5 rows

12:57 PM (<1s) 21

```python
products_dim.show(5)
```

(2) Spark Jobs

ProductID	CategoryID	ProductName	ProductBrand	ProductModelNo	ProductStock
3000	4000	CAMERA	NIKON	1100	10
3089	4006	PENDRIVE	TRANSCEND	9976	9
3098	4002	BAG	AMERICAN TOURISTER	37950	9
3042	4007	EARPHONES	SOUND MAGIC	11540	9
3028	4009	SUNGLASSES	GUCCI	6270	9

only showing top 5 rows

12:57 PM (<1s) 25

```python
seller_dim.show(5)
```

(2) Spark Jobs

SellerID	SellerName	SellerRating	SellerStreetAddress	SellerCity	SellerState	SellerZip	SellerPhoneNo
200049	Cherokee Richard	5	Ap #381-3851 Eget...	PARIS	ILE-DE-FRANCE	72816	01 33 96 49 60
200031	Robert Becker	1	182-4566 Gravida ...	ANGERS	PAYS DE LA LOIRE	90358	01 39 47 76 38
200001	Aaron Cooper	2	P.O. Box 446, 302...	LE MANS	PAYS DE LA LOIRE	94302	07 25 54 31 55
200021	Bo Page	3	Ap #257-6739 Lect...	GRENOBLE	RHONE-ALPES	65860	03 18 25 76 17
200096	Mary Lloyd	1	6075 Mus. Av.	REIMS	CHAMPAGNE-ARDENNE	80672	01 41 65 21 82

only showing top 5 rows

12:57 PM (<1s) 34 Python

```
time_date_dim.show()
```

(1) Spark Jobs

```
+----+-------+-------+-----------+--------+------+
|YEAR|QUARTER|  MONTH|DAY_OF_WEEK|    DATE|TimeID|
+----+-------+-------+-----------+--------+------+
|2008|      1|January|          2|20080101|     0|
|2008|      1|January|          3|20080102|     1|
|2008|      1|January|          4|20080103|     2|
|2008|      1|January|          5|20080104|     3|
|2008|      1|January|          6|20080105|     4|
|2008|      1|January|          7|20080106|     5|
|2008|      1|January|          1|20080107|     6|
|2008|      1|January|          2|20080108|     7|
|2008|      1|January|          3|20080109|     8|
|2008|      1|January|          4|20080110|     9|
|2008|      1|January|          5|20080111|    10|
|2008|      1|January|          6|20080112|    11|
|2008|      1|January|          7|20080113|    12|
|2008|      1|January|          1|20080114|    13|
|2008|      1|January|          2|20080115|    14|
|2008|      1|January|          3|20080116|    15|
```

CREATING FACT AND DIMENSIONAL TABLES

Creating Seller DIM table

Markdown

12:57 PM (<1s) 23 Python

```
seller_dim = df11.select(
    "SELLER_ID",
    "SELLER_NAME",
    "SELLER_RATING",
    "SELLER_STREET_ADDRESS",
    "SELLER_CITY",
    "SELLER_STATE",
    "SELLER_ZIP",
    "SELLER_PHONE_NO"
)
```

seller_dim: pyspark.sql.dataframe.DataFrame = [SELLER_ID: integer, SELLER_NAME: string ... 6 more fields]

12:57 PM (<1s) 24

```
seller_dim = seller_dim.withColumnRenamed("SELLER_ID", "SellerID") \
                       .withColumnRenamed("SELLER_NAME", "SellerName") \
                       .withColumnRenamed("SELLER_RATING", "SellerRating") \
                       .withColumnRenamed("SELLER_STREET_ADDRESS", "SellerStreetAddress") \
                       .withColumnRenamed("SELLER_CITY", "SellerCity") \
                       .withColumnRenamed("SELLER_STATE", "SellerState") \
                       .withColumnRenamed("SELLER_ZIP", "SellerZip") \
                       .withColumnRenamed("SELLER_PHONE_NO", "SellerPhoneNo")
```

seller_dim: pyspark.sql.dataframe.DataFrame = [SellerID: integer, SellerName: string ... 6 more fields]

creating dim table

12:57 PM (<1s) 39 Python

```
transaction_dim = df11.select(
    "TRANSACTION_ID",
    "TRANSACTION_DATE",
    "TRANSACTION_AMOUNT",
    "TRANSACTION_TYPE",
    "DISPATCH_DATE",
    "EXPECTED_DATE",
    "DELIVERY_DATE"
)
```

transaction_dim: pyspark.sql.dataframe.DataFrame = [TRANSACTION_ID: integer, TRANSACTION_DATE: string ... 5 more fields]

12:57 PM (<1s) 40

```
transaction_dim = transaction_dim.withColumnRenamed("TRANSACTION_ID", "TransactionID") \
                                 .withColumnRenamed("TRANSACTION_DATE", "TransactionDate") \
                                 .withColumnRenamed("TRANSACTION_AMOUNT", "TransactionAmount") \
                                 .withColumnRenamed("TRANSACTION_TYPE", "TransactionType") \
                                 .withColumnRenamed("DISPATCH_DATE", "DispatchDate") \
                                 .withColumnRenamed("EXPECTED_DATE", "ExpectedDate") \
                                 .withColumnRenamed("DELIVERY_DATE", "DeliveryDate")
```

transaction_dim: pyspark.sql.dataframe.DataFrame = [TransactionID: integer, TransactionDate: string ... 5 more fields]

output

Markdown

12:57 PM (<1s) 27 Python

```
seller_dim.show(5)
```

(2) Spark Jobs

```
+--------+----------------+------------+--------------------+----------+------------------+---------+--------------+
|SellerID|      SellerName|SellerRating| SellerStreetAddress|SellerCity|       SellerState|SellerZip| SellerPhoneNo|
+--------+----------------+------------+--------------------+----------+------------------+---------+--------------+
|  200049|Cherokee Richard|           5|Ap #381-3851 Eget...|     PARIS|     ILE-DE-FRANCE|    72816|01 33 96 49 60|
|  200031|   Robert Becker|           1|182-4566 Gravida ...|    ANGERS| PAYS DE LA LOIRE|    90358|01 39 47 76 38|
|  200001|    Aaron Cooper|           2|P.O. Box 446, 302...|   LE MANS| PAYS DE LA LOIRE|    94302|07 25 54 31 55|
|  200021|         Bo Page|           3|Ap #257-6739 Lect...|  GRENOBLE|       RHONE-ALPES|    65860|03 18 25 76 17|
|  200096|      Mary Lloyd|           1|       6075 Mus. Av.|     REIMS|CHAMPAGNE-ARDENNE|    80672|01 41 65 21 82|
+--------+----------------+------------+--------------------+----------+------------------+---------+--------------+
only showing top 5 rows
```

creating time DIM table

12:57 PM (<1s) 33 Python

```
time_date_dim = df2.select(
    "fb_year",
    "QUARTER",
    "month_name",
    "DAY_OF_WEEK",
    "DATE_KEY"
)
```

time_date_dim: pyspark.sql.dataframe.DataFrame = [fb_year: integer, QUARTER: integer ... 3 more fields]

12:57 PM (<1s) 34

```
time_date_dim = time_date_dim.withColumn("TimeID", monotonically_increasing_id())
```

time_date_dim: pyspark.sql.dataframe.DataFrame = [fb_year: integer, QUARTER: integer ... 4 more fields]

12:57 PM (<1s) 35

```
time_date_dim = time_date_dim.withColumnRenamed("fb_Year", "YEAR") \
                             .withColumnRenamed("Quarter", "QUARTER") \
                             .withColumnRenamed("Month_name", "MONTH") \
                             .withColumnRenamed("Day_Of_Week", "DAY_OF_WEEK") \
                             .withColumnRenamed("Date_key", "DATE")
```

time_date_dim: pyspark.sql.dataframe.DataFrame = [YEAR: integer, QUARTER: integer ... 4 more fields]

Output

12:57 PM (<1s) 38 Python

```
time_date_dim.show()
```

(1) Spark Jobs

```
|2008|      1|January|          4|20080103|     2|
|2008|      1|January|          5|20080104|     3|
|2008|      1|January|          6|20080105|     4|
|2008|      1|January|          7|20080106|     5|
|2008|      1|January|          1|20080107|     6|
|2008|      1|January|          2|20080108|     7|
|2008|      1|January|          3|20080109|     8|
|2008|      1|January|          4|20080110|     9|
|2008|      1|January|          5|20080111|    10|
|2008|      1|January|          6|20080112|    11|
|2008|      1|January|          7|20080113|    12|
|2008|      1|January|          1|20080114|    13|
|2008|      1|January|          2|20080115|    14|
|2008|      1|January|          3|20080116|    15|
|2008|      1|January|          4|20080117|    16|
|2008|      1|January|          5|20080118|    17|
|2008|      1|January|          6|20080119|    18|
|2008|      1|January|          7|20080120|    19|
+----+-------+-------+-----------+--------+------+
only showing top 20 rows
```

12:57 PM (<1s) 21

```
products_dim = products_dim.dropDuplicates(["ProductID"])
```

products_dim: pyspark.sql.dataframe.DataFrame = [ProductID: integer, CategoryID: integer ... 4 more fields]

Output

12:57 PM (<1s) 23 Python

```
products_dim.show(5)
```

(2) Spark Jobs

```
+---------+----------+-----------+-----------------+--------------+------------+
|ProductID|CategoryID|ProductName|     ProductBrand|ProductModelNo|ProductStock|
+---------+----------+-----------+-----------------+--------------+------------+
|     3000|      4000|     CAMERA|            NIKON|          1100|          10|
|     3089|      4006|   PENDRIVE|        TRANSCEND|          9976|           9|
|     3098|      4002|        BAG|AMERICAN TOURISTER|        37950|           9|
|     3042|      4007|  EARPHONES|      SOUND MAGIC|         11540|           9|
|     3028|      4009| SUNGLASSES|            GUCCI|          6270|           9|
+---------+----------+-----------+-----------------+--------------+------------+
only showing top 5 rows
```

Creating the Fact Table

12:57 PM (<1s) 48 Python

```
inventory_fact = df11.select(
    "TRANSACTION_ID",
    "PRODUCT_ID",
    "CUSTOMER_ID",
    "SELLER_ID",
    "TRANSACTION_DATE",
    "PRODUCT_COST_PRICE",
    "PRODUCT_SELLING_PRICE"
)
```

inventory_fact: pyspark.sql.dataframe.DataFrame = [TRANSACTION_ID: integer, PRODUCT_ID: integer ... 5 more fields]

12:57 PM (<1s) 49

```
inventory_fact = inventory_fact.join(
    time_dim.select("TimeID", "Date"),
    inventory_fact.TRANSACTION_DATE == time_dim.Date,
    "left"
).drop("Date", "TRANSACTION_DATE")
```

inventory_fact: pyspark.sql.dataframe.DataFrame = [TRANSACTION_ID: integer, PRODUCT_ID: integer ... 5 more fields]

12:57 PM (<1s) 50

```
inventory_fact = inventory_fact.withColumn("FactID", monotonically_increasing_id())
```

inventory_fact: pyspark.sql.dataframe.DataFrame = [TRANSACTION_ID: integer, PRODUCT_ID: integer ... 6 more fields]

12:57 PM (<1s) 51

```
inventory_fact = inventory_fact.select(
    "FactID", "TimeID", "TRANSACTION_ID", "PRODUCT_ID",
    "CUSTOMER_ID", "SELLER_ID",
    "PRODUCT_COST_PRICE", "PRODUCT_SELLING_PRICE"
)
```

inventory_fact: pyspark.sql.dataframe.DataFrame = [FactID: long, TimeID: long ... 6 more fields]

12:57 PM (1s) 52 Python

```
inventory_fact.show(5)
```

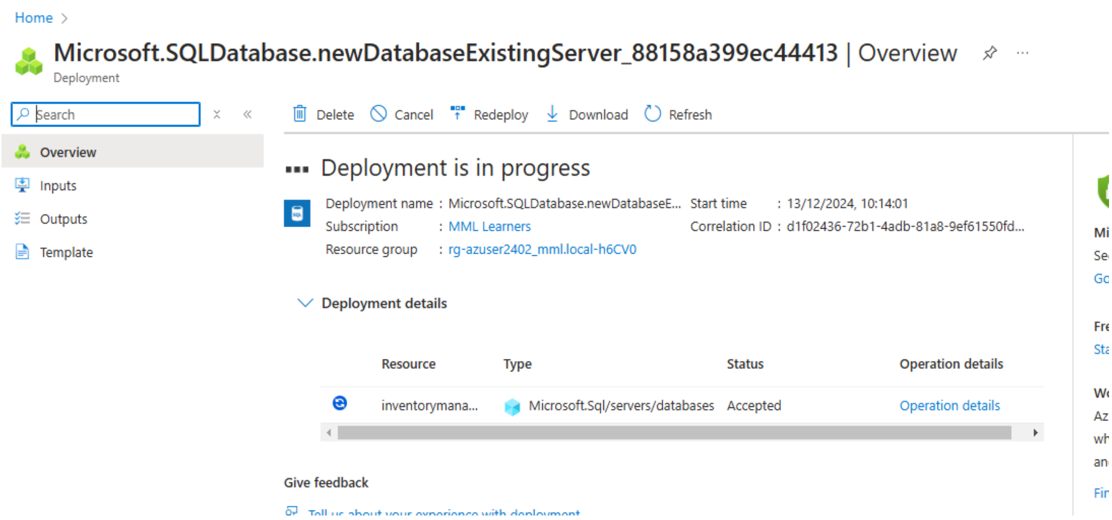
(4) Spark Jobs

```
+------+------+--------------+----------+-----------+---------+------------------+---------------------+
|FactID|TimeID|TRANSACTION_ID|PRODUCT_ID|CUSTOMER_ID|SELLER_ID|PRODUCT_COST_PRICE|PRODUCT_SELLING_PRICE|
+------+------+--------------+----------+-----------+---------+------------------+---------------------+
|     0|   734|          5000|      3100|     100060|   200074|             32000|                35000|
|     1|   734|          5001|      3001|     100038|   200091|              2450|                 2450|
|     2|   734|          5002|      3012|     100045|   200048|              3500|                 3500|
|     3|   322|          5003|      3089|     100090|   200055|               600|                  650|
|     4|   875|          5004|      3023|     100097|   200025|             22000|                21500|
+------+------+--------------+----------+-----------+---------+------------------+---------------------+
```

STORING IN SQL DATABASE

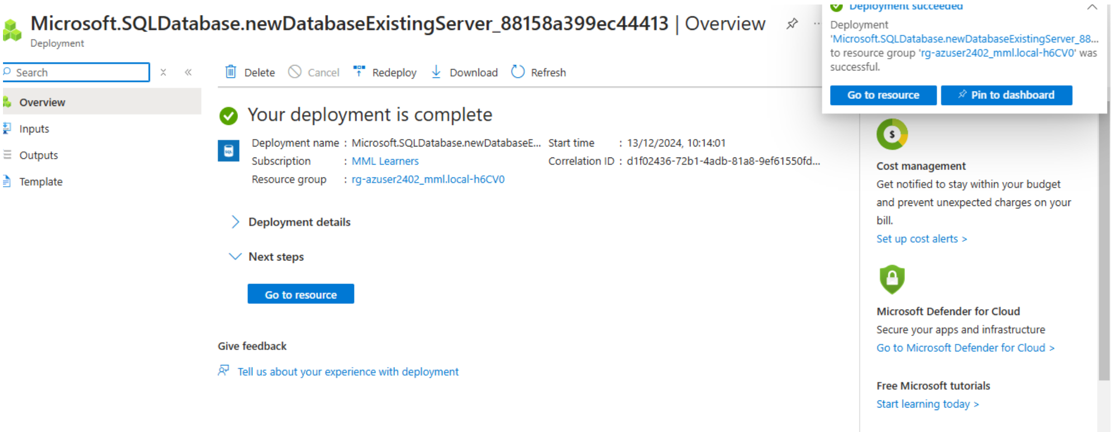
First We created an Azure SQL Database Server

Azure SQL Server is a logical server that acts as a container for hosting multiple Azure SQL
Databases. It provides the foundation for managing and scaling your relational databases in
the cloud.

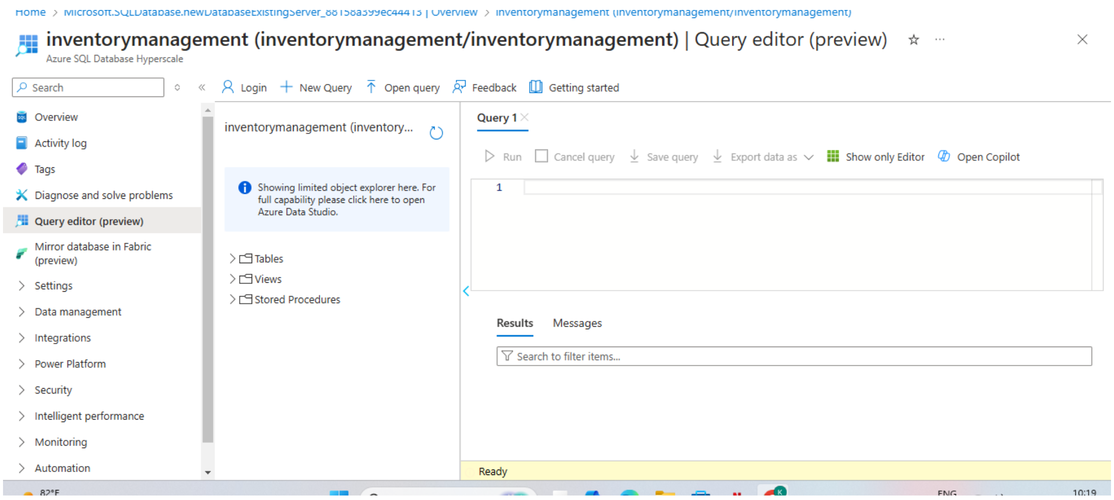


Creating SQL Database

Azure SQL Database is a fully managed, relational database service built on Microsoft's SQL
Server technology, designed for scalability, performance, and availability in the cloud. It
supports a wide range of workloads, from small-scale applications to large-scale enterprise
systems, making it an ideal choice for modern application development.



Storing all Dimensions and Fact Tables in Database



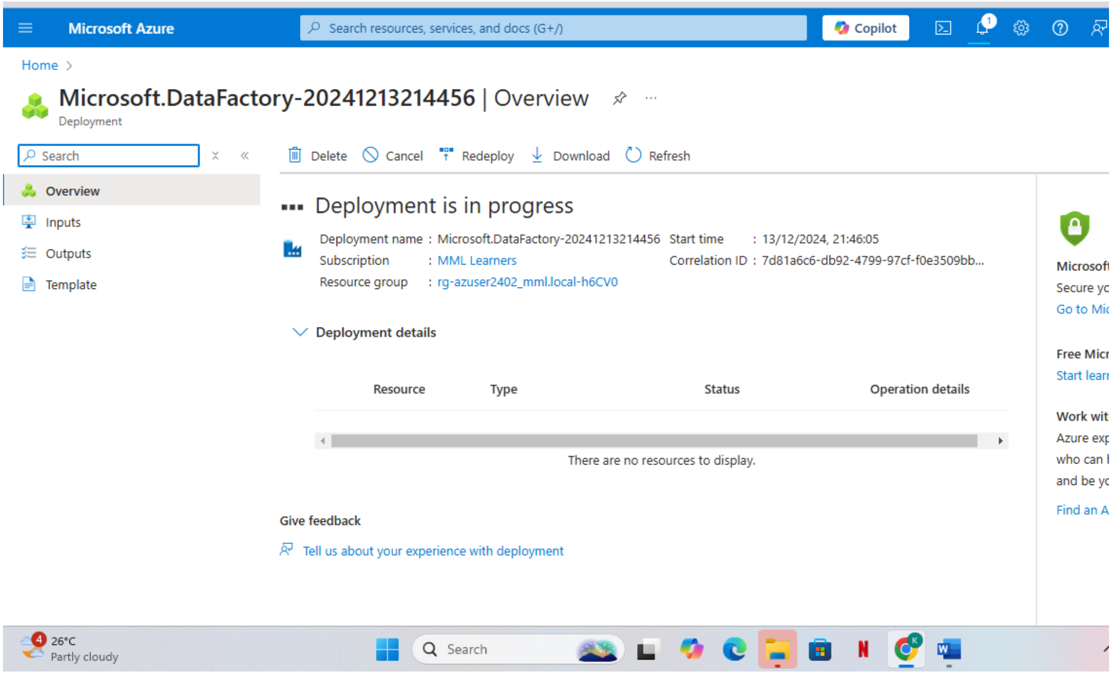
Azure Data Factory (ADF)

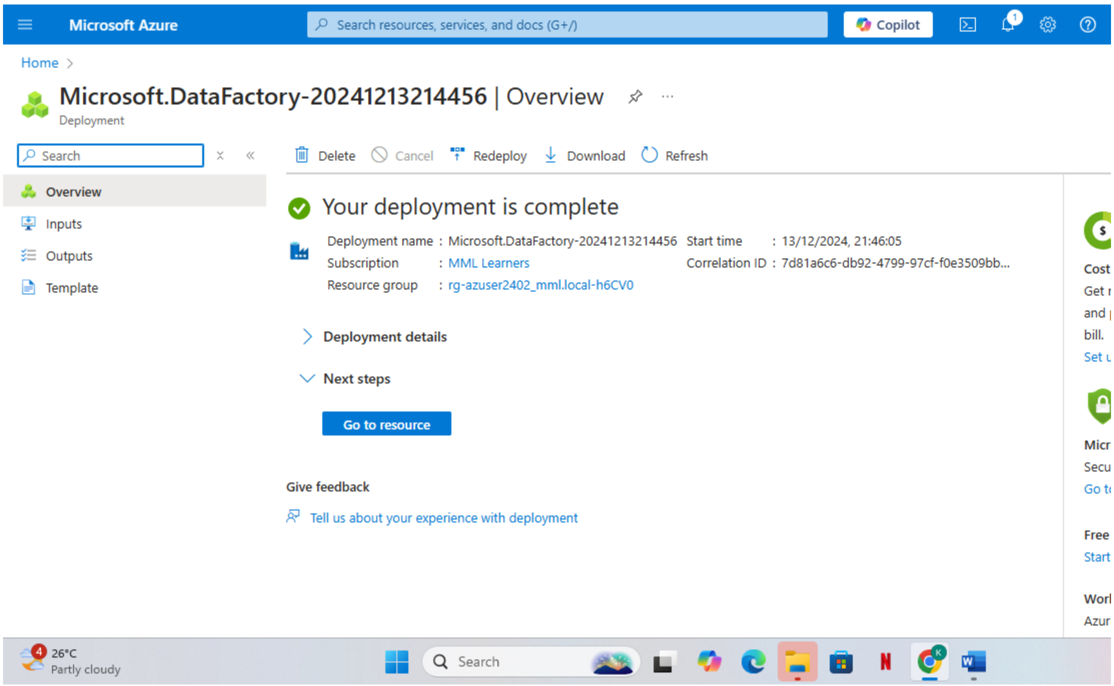
Azure Data Factory (ADF) is a cloud-based data integration and orchestration service provided by Microsoft Azure. It enables the creation, scheduling, and monitoring of data pipelines to efficiently transform and move data between diverse sources and destinations.

Key Features of Azure Data Factory:

- **Seamless Data Movement:** Facilitates data integration across on-premises, cloud, and hybrid environments.
- **ETL and ELT Pipelines:** Supports scalable and efficient **Extract, Transform, Load (ETL)** and **Extract, Load, Transform (ELT)** workflows, empowering organizations to process large volumes of data with ease.
- **Automation and Monitoring:** Provides tools for scheduling and monitoring data pipelines to ensure reliable and automated workflows.

- **Flexibility:** Supports a wide range of data formats, connectors, and transformations, making it a versatile solution for modern data workflows.



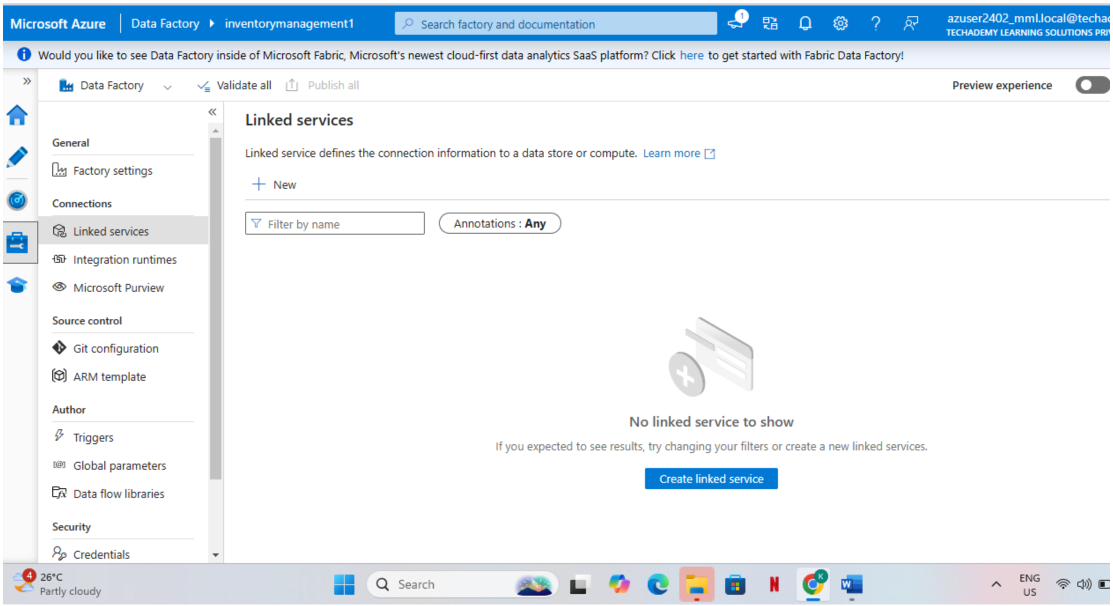


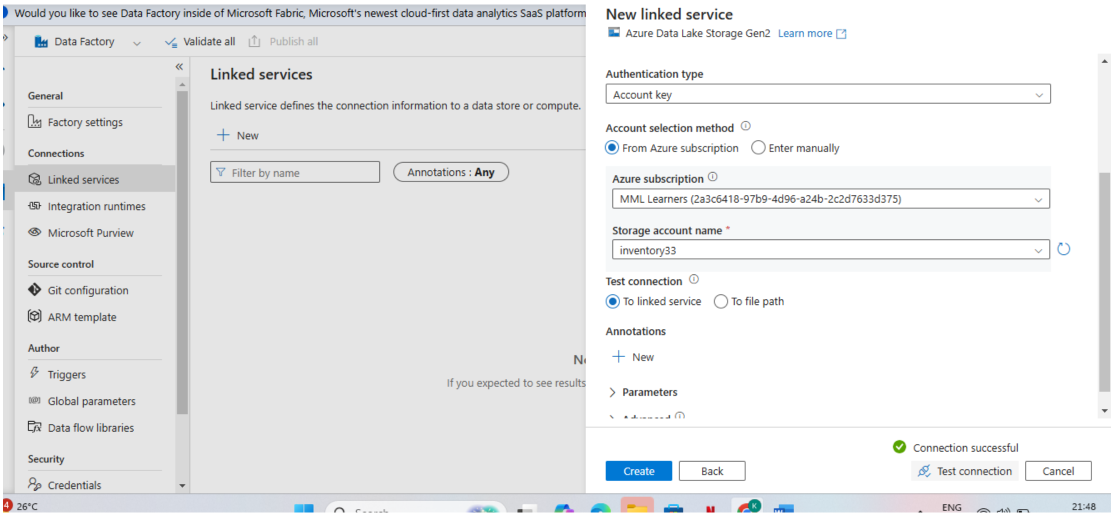
Creating an Azure Data Lake Storage (ADLS) Service Connection with Azure Data Factory (ADF)

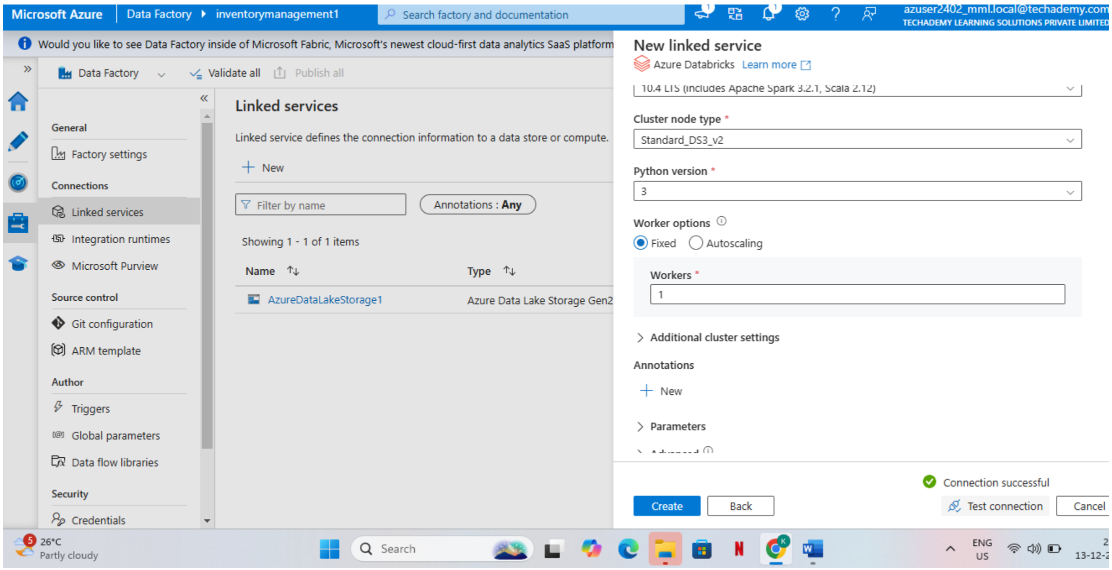
Azure Data Lake Storage (ADLS) is a high-performance, enterprise-grade storage solution optimized for big data analytics. To enable Azure Data Factory (ADF) to interact with data stored in ADLS, a **service connection (linked service)** needs to be created.

This connection serves as a secure bridge, allowing ADF to seamlessly read, write, and manage data in ADLS as part of data integration and ETL workflows. By establishing this connection, ADF pipelines can efficiently perform tasks such as:

- **Data Ingestion**: Importing raw data from various sources into ADLS.
- **Data Transformation**: Processing and transforming data for analytical or operational purposes.
- **Data Export**: Moving processed data from ADLS to downstream systems or destinations







Creating Database Service Connection in Azure Data Factory (ADF)

Establishing a database service connection in Azure Data Factory (ADF) enables seamless integration with relational and non-relational databases. This connection allows ADF to move, transform, and process data effectively. In ADF, **service connections (linked services)** serve as connectors to external data sources, empowering pipelines to interact with databases hosted on Azure (e.g., Azure SQL Database, SQL Server), on-premises, or third-party cloud environments.

Service Connection:

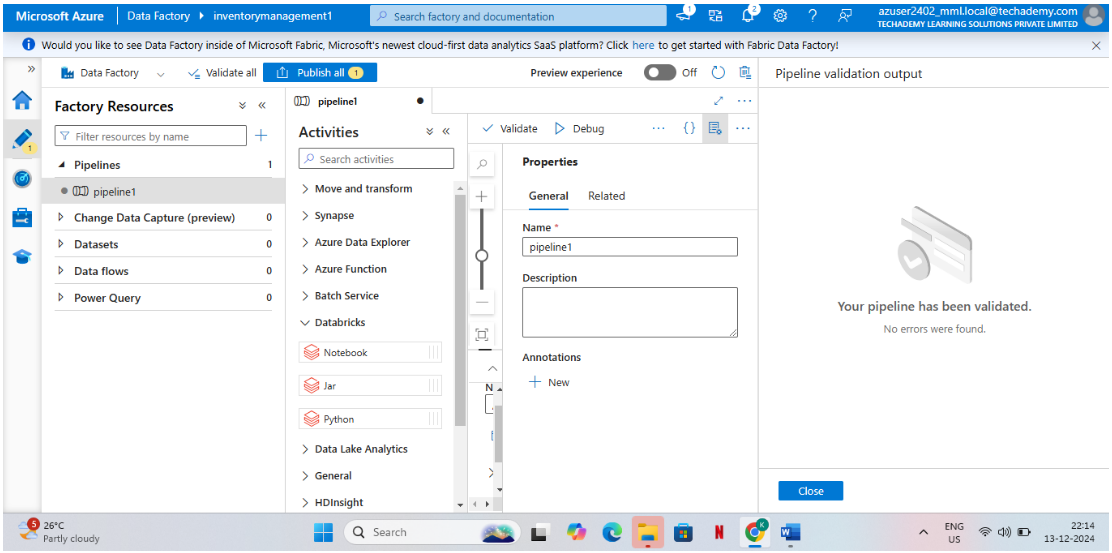
1.

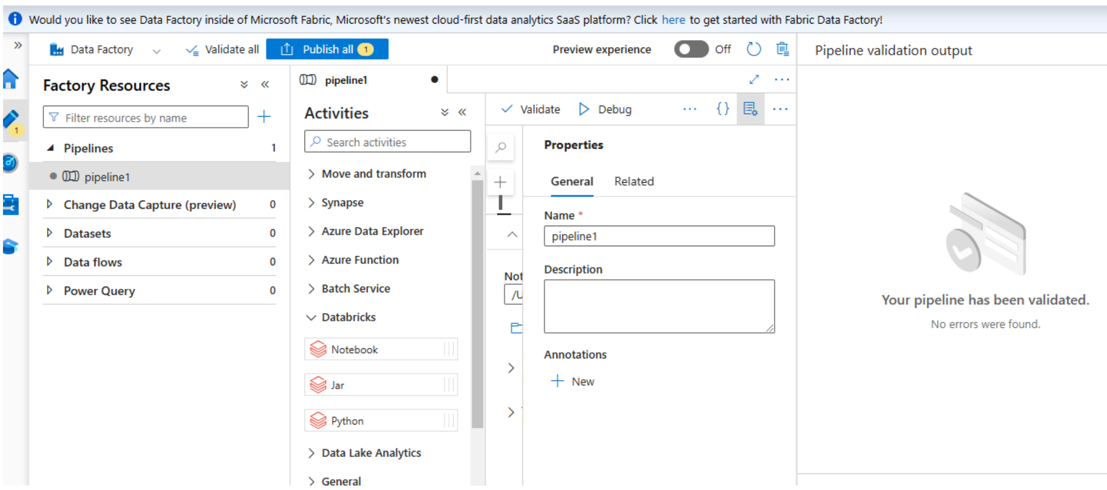
1. Provides ADF with the credentials and configuration to securely connect to database instances.
2. Enables ADF to perform read and write operations as part of ETL (Extract, Transform, Load) or ELT (Extract, Load, Transform) workflows.

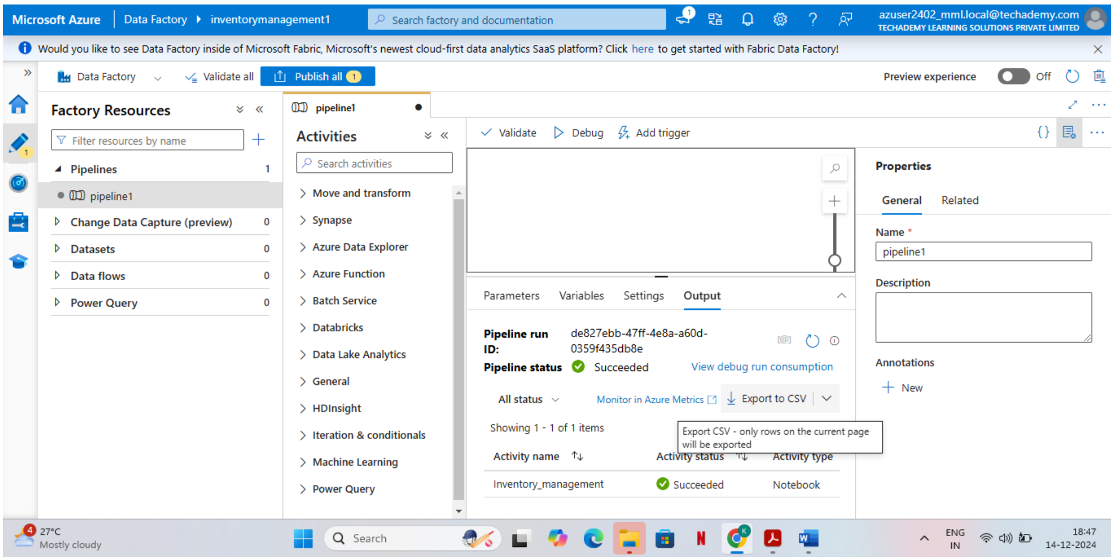
Integration Runtime (IR):

1. Acts as the compute infrastructure to move and transform data across cloud and on-premises data stores.
2. Facilitates secure and scalable data transfer and transformation for database connections.

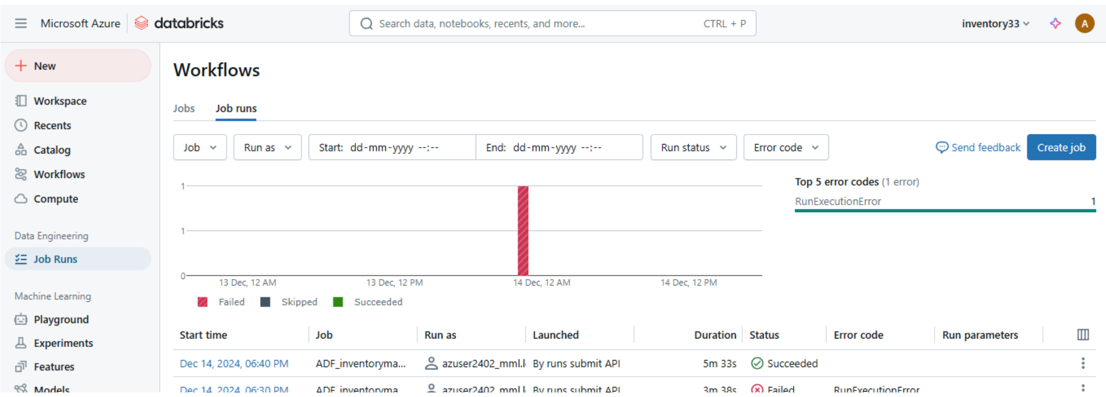
Pipeline is created to execute the Notebook







Job runs in Azure Data bricks



Conclusion

The "Inventory Management System" project successfully leveraged powerful data processing tools such as PySpark and Azure services to analyze and derive meaningful insights from large datasets. By transforming raw data into structured fact and dimension tables, the system implemented a scalable and efficient data pipeline capable of supporting comprehensive analysis. The integration of **Delta Lake** and **Azure Databricks** enabled the execution of complex transformations and machine learning models at scale, delivering real-time analytics to enhance decision-making.

The incorporation of **Azure Data Factory** further streamlined the data processing and report generation workflows by automating tasks and minimizing manual intervention. This ensured the system was not only robust but also scalable to handle increasing data volumes effectively. The generated reports—spanning product sales metrics, customer trends, and transaction statistics—offered actionable insights into inventory and product performance. These insights can be leveraged to optimize sales tracking, improve inventory control, and inform strategic planning for future business growth.

Ultimately, this project demonstrated how the integration of big data tools and cloud services can transform raw data into actionable insights, enabling stakeholders to make data-driven decisions. By combining scalability, automation, and advanced analytics, the system empowers organizations to gain a deeper understanding of their inventory metrics and drive improved operational performance.