

# Ingestion and Transformation of Olympics Data

## Title:

This project is designed to showcase how various Azure services can be utilized to ingest, store, transform Olympics-related data. We have employed the following Azure services to achieve these goals:

- **Data Ingestion: Configure and run the ADF pipelines to ingest the Olympics data into Azure Data Lake Storage Gen2.**
- **Data Transformation: Execute the PySpark notebooks in Azure Databricks to perform data transformation tasks**

## Project Overview

The main objective of this project is to ingest Olympic data into Azure Data Lake Storage Gen2 using Azure Data Factory (ADF) pipelines and perform data transformation tasks using PySpark notebooks in Azure Databricks. The processed data will be stored back in Azure Data Lake Storage Gen2 for further analysis.

## Workflow:

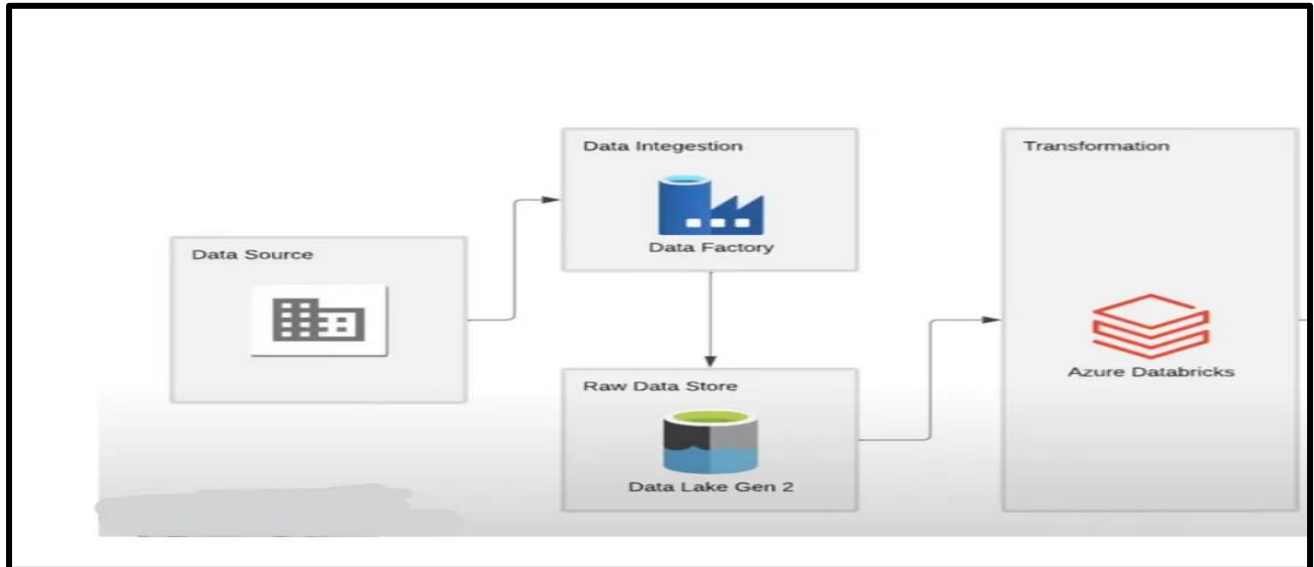
1. **Data Ingestion:**
  - Raw Olympic data from various sources is ingested into Azure Data Lake Storage Gen2 using ADF pipelines.
  - ADF is configured to handle data movement and transformation activities, ensuring data integrity and security during the ingestion process.
2. **Data Transformation:**
  - Azure Databricks is employed to execute PySpark notebooks that perform data transformations on the ingested Olympic data.
  - PySpark notebooks read data from Azure Data Lake Storage Gen2, apply necessary transformations, and store the processed data back in the data lake.

## Azure Resources/Tools Used:

- **Azure Data Factory (ADF):** Used for data ingestion, Azure Data Factory allows us to efficiently collect data from various sources and move it to the desired destination. In this project, ADF is responsible for bringing in Olympics data from external sources.
- **Azure Data Lake Storage Gen2:** This is where the ingested data is stored. Azure Data Lake Storage Gen2 provides a scalable and secure platform for storing large volumes of data. It enables us to manage, access, and analyse data effectively.
- **Azure Databricks:** For data transformation, we leverage Azure Databricks with PySpark. Databricks provides a collaborative environment for data engineers and data scientists to work together on big data projects. In our project, we use PySpark to clean, reshape, and process the raw Olympics data into a more usable format.
- **Python with PySpark:** PySpark is the Python API for Apache Spark, which is a fast and general-purpose cluster computing system. PySpark would be used within Azure Databricks notebooks for developing and executing data transformation tasks.

- **Microsoft Excel:** Microsoft Excel might be used for data preparation or as a data source for creating input datasets for the project.

## Architectural diagram



## Execution Overview

To ingest Olympics data into Azure Data Lake Storage Gen2, Azure Data Factory (ADF) pipelines are configured and executed. These pipelines are designed to efficiently transfer data from various sources to the designated Azure Data Lake Storage Gen2 destination. The configuration includes defining the data sources, mapping transformations, and specifying the target location within the Data Lake.

Once the data is ingested, the next step involves executing PySpark notebooks in Azure Databricks to perform data transformation tasks. These PySpark notebooks contain the logic and code necessary to clean, process, and transform the raw Olympics data into a structured and analyzable format. The notebooks leverage the distributed computing capabilities of Apache Spark to handle large datasets efficiently.

This end-to-end process ensures a seamless flow from data ingestion using ADF pipelines to data transformation using Azure Databricks, ultimately preparing the Olympics data for further analysis and insights within the Azure ecosystem. The combination of these Azure services facilitates a scalable and flexible approach to managing and processing large volumes of data for analytics and reporting purposes.

## How it works

### Data Ingestion:

#### 1) Source Data Identification:

- Identify the sources of Olympics data.
- These sources could include databases, Azure storage accounts, or any other data repositories where the Olympics data is stored.

#### 2) Azure Data Lake Storage Gen2 Configuration:

- Set up Azure Data Lake Storage Gen2 (ADLS Gen2) as the destination for storing the Olympics data.
- Go to Azure Portal and search for Storage Accounts.
- Click on + to create. Add the Subscription , Resource group name , storage account name
- Enable Hierarchical Namespace to organize data in a hierarchical structure, improving performance and manageability in Advanced setting
- Click on Create
- Deployment is in progress.
- Once the deployment is complete Click on Go to Resource.

#### 3) Azure Data Factory (ADF) Pipeline Configuration:

- Create a Azure data factory by selecting the Azure subscription, Resource group and give a name for data factory and Click on Create.
- Once the deployment is complete, Click on Go to Resource
- Click on Launch Studio.
- Click on Ingest Data
- Give the details of Source and Destination to perform Copy Activity.

#### 4) Run the Pipeline:

- Once the pipeline is configured, trigger the pipeline execution manually or let it run based on the defined schedule.
- Monitor the pipeline execution for any errors or issues and troubleshoot as needed.
- Verify that the data has been successfully ingested into ADLS Gen2.

### Data Transformation:

#### 1) Azure Databricks Environment Setup:

- Set up an Azure Databricks workspace in the Azure portal.
- Configure clusters within the Databricks workspace with appropriate specifications for running PySpark notebooks.

- Specify the cluster type, instance types, number of instances, and auto-scaling settings based on the workload requirements.
- It is a best practice to use personal compute to reduce the costing.

## **2) PySpark Notebook Development:**

- Create a new notebook within the Azure Databricks workspace.
- Set the configuration to Azure Data Lake Storage gen2 account by providing name of the storage account, name of container and key.
- Develop PySpark notebooks to perform various data transformation tasks on the ingested Olympics data. Tasks may include aggregation, filtering, distinct operations, data normalization, joining datasets, etc.
- Write PySpark code to implement the desired data transformation logic.
- Ensure that the code is well-documented and includes comments for clarity and maintainability.
- Test the PySpark code within the notebook to verify that it produces the expected output.

## **3) Notebook Execution and Scheduling:**

- Execute the developed PySpark notebooks within the Azure Databricks environment to perform data transformation tasks on the ingested Olympics data.
- Schedule the execution of notebooks at specified intervals or triggers to ensure regular updates to the transformed data.
- Configure notebook parameters and input/output paths as necessary for scheduled execution.
- Review the transformed data output to ensure it meets the desired transformation requirements.

# **Tasks performed**

## **Data Ingestion with ADF Pipelines:**

- Configure Azure Data Factory (ADF) pipelines to ingest Olympics data.
- Define the necessary data sources and sinks, specifying Azure Data Lake Storage Gen2 as the destination.
- Set up activities within the pipeline to orchestrate the data movement.
- Schedule or trigger the pipeline execution as per our requirements.

## **Data Transformation with PySpark Notebooks in Azure Databricks:**

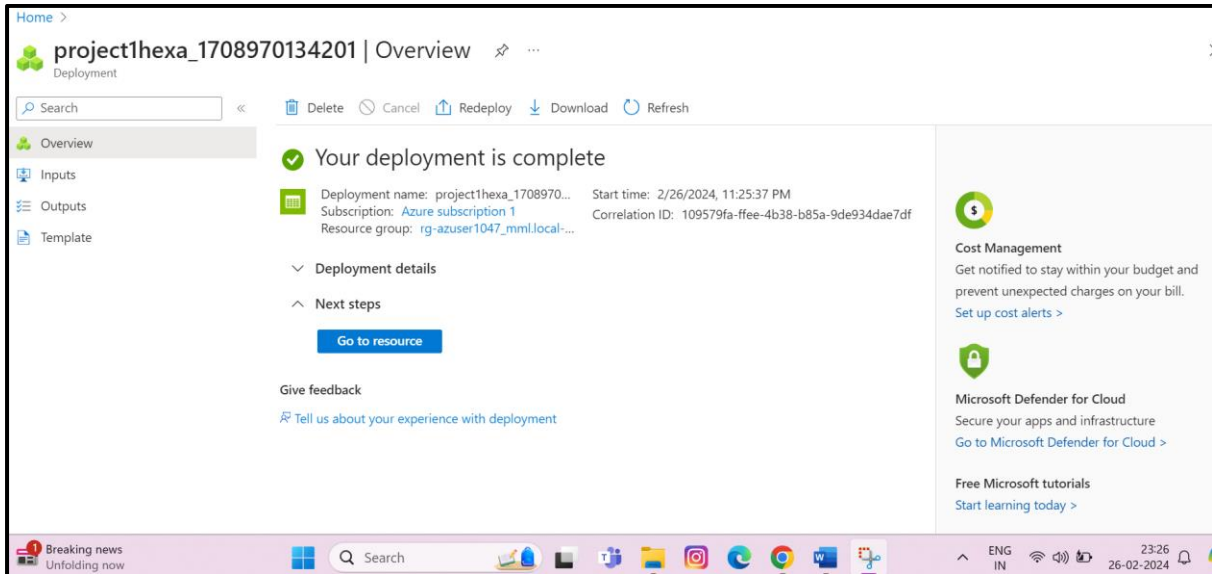
- Utilize Azure Databricks for data transformation using PySpark.
- Create Azure Databricks workspace in Azure portal.
- Develop PySpark notebooks to perform specific transformation tasks like sorting, filtering, aggregations, distinct columns etc., on the ingested Olympics data.
- Leverage the scalability of Databricks for efficient distributed processing.

- Execute the notebooks, ensuring they handle the data according to your transformation logic.

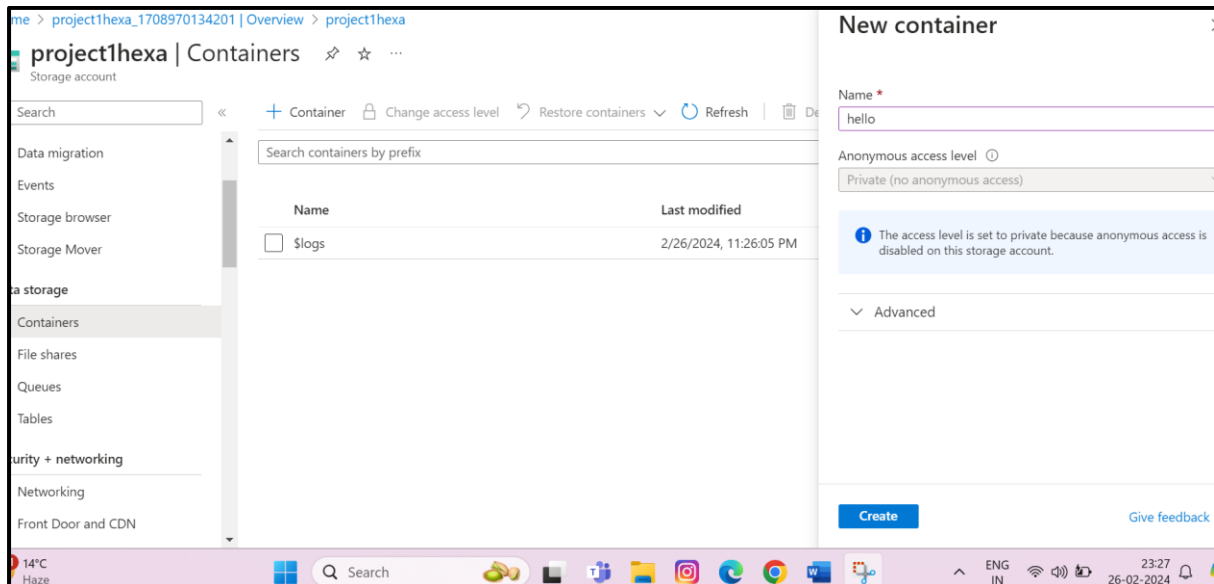
## RESULTS

### Setting Up Source and destination Storage Accounts

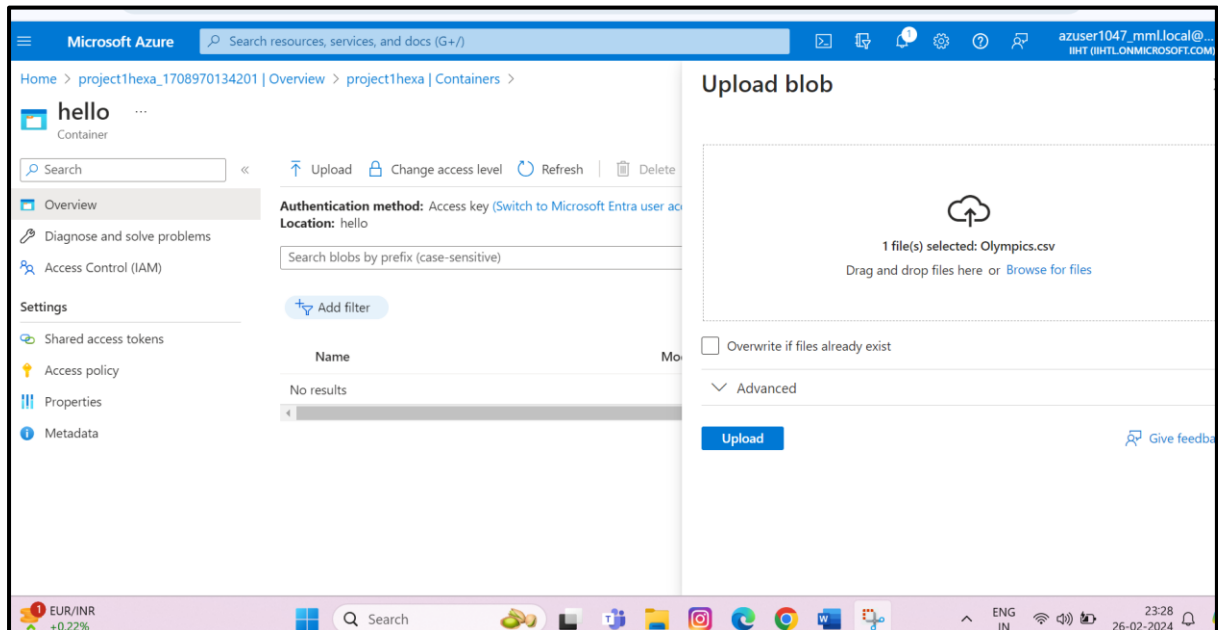
**First a Blob storage is created:**



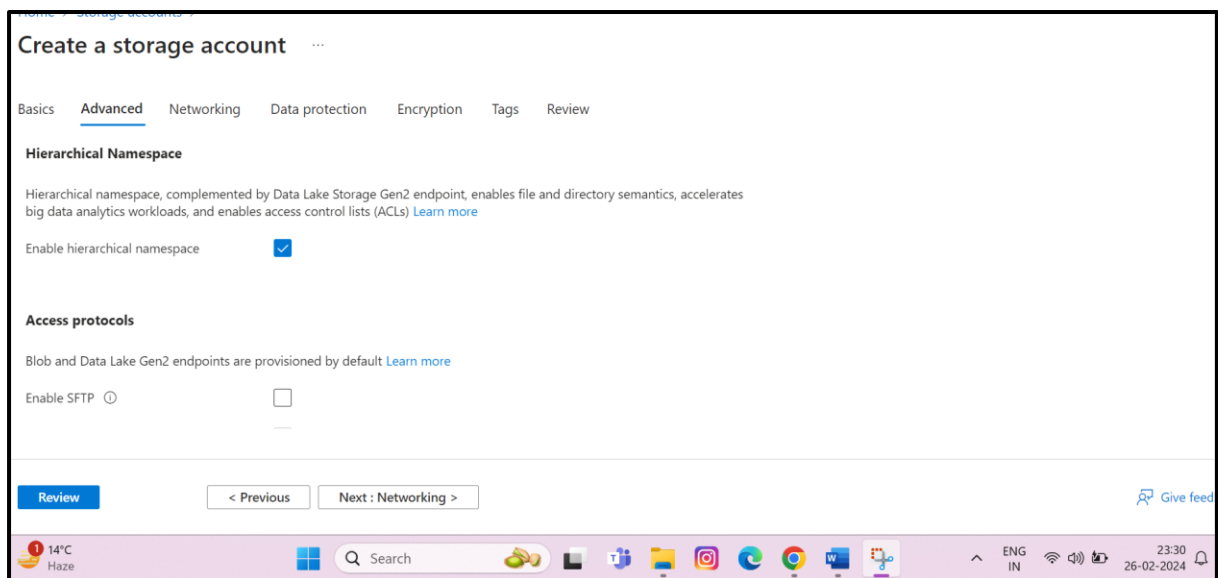
**Inside the blob storage a Container is created:**



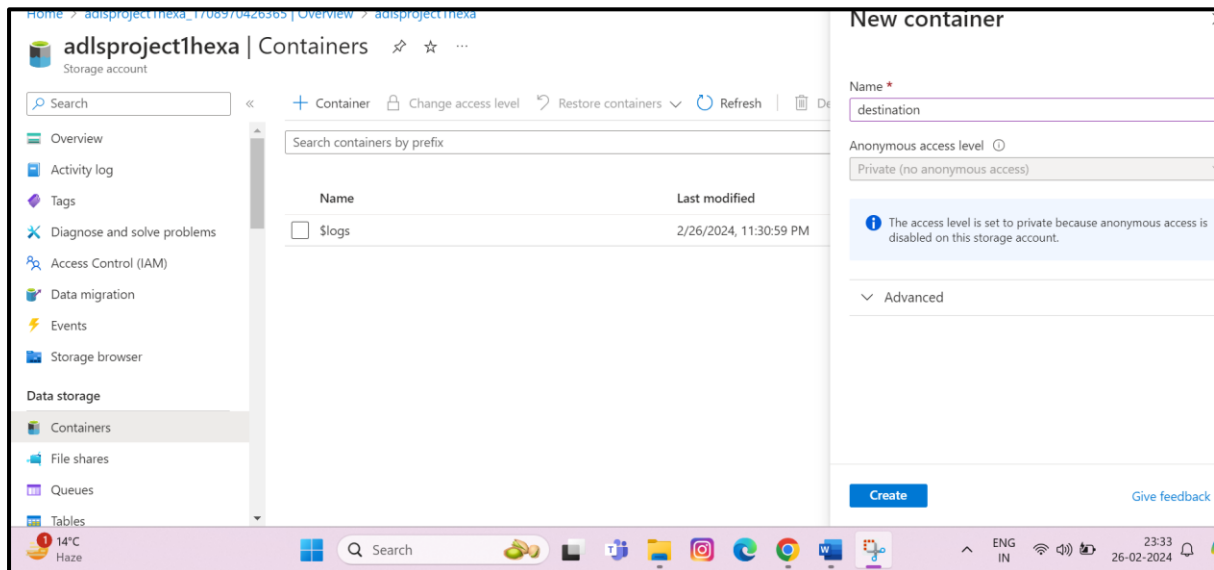
After that the source file is uploaded in the container:



Now creating an Aure Data Lake Storage GEN2(ADLS) account by enabling hierarchical namespace:

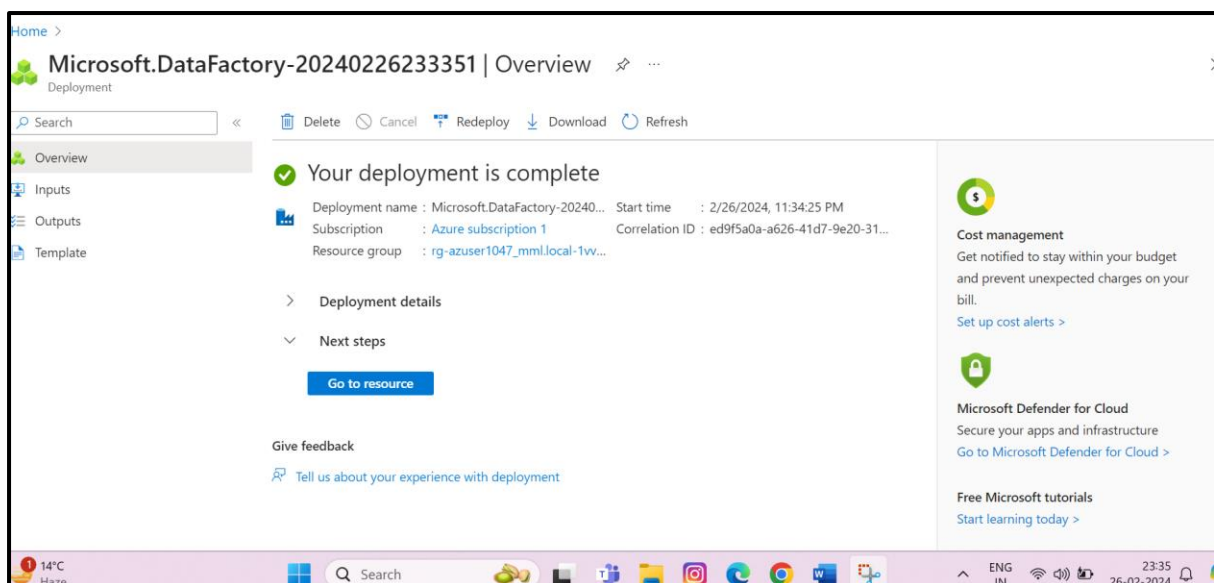


**Creating a container in ADLS where file will be copied through copy action:**

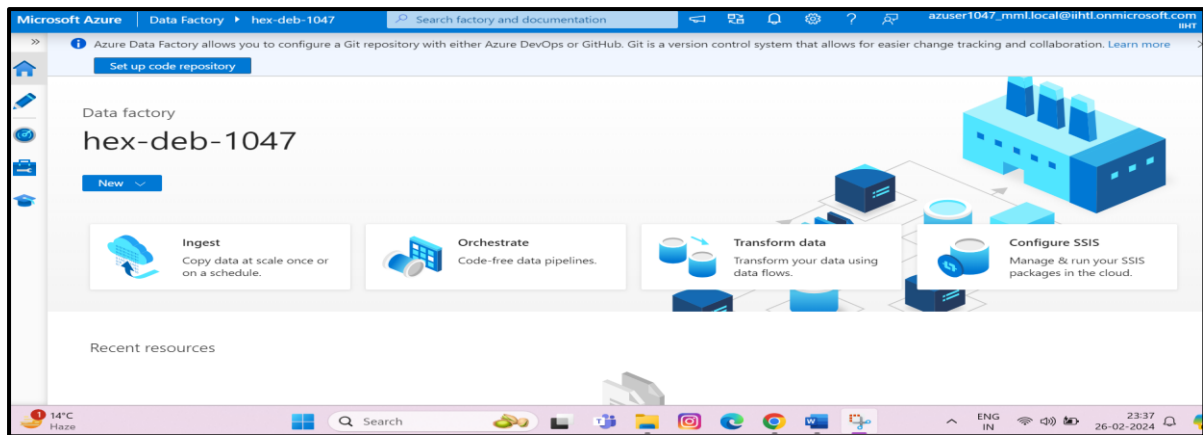


## Data Ingestion

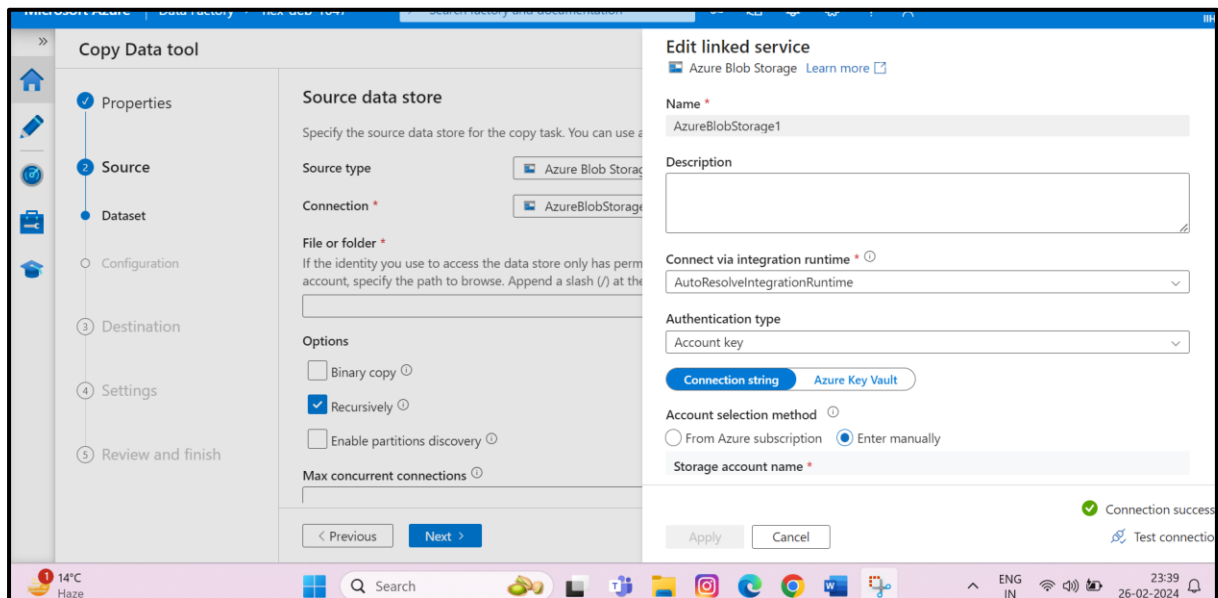
**Data factory created successfully:**



**ADF portal:**

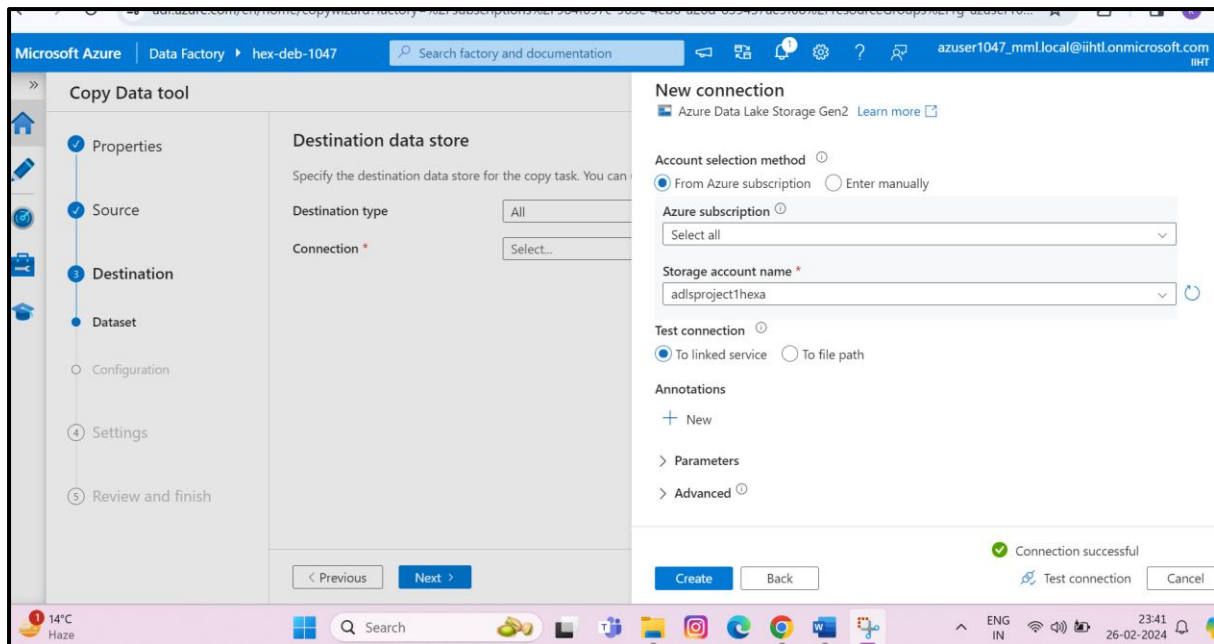


**Connecting to source linked service i.e. blob storage:**

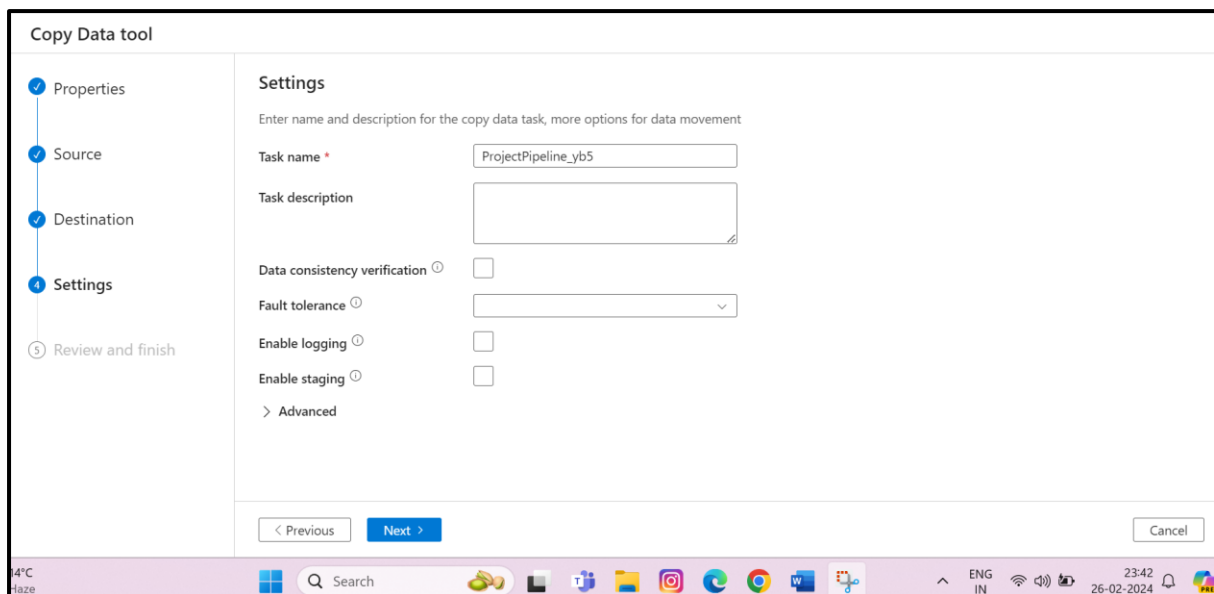


**Adding destination storage account i.e. data lake store gen2:**

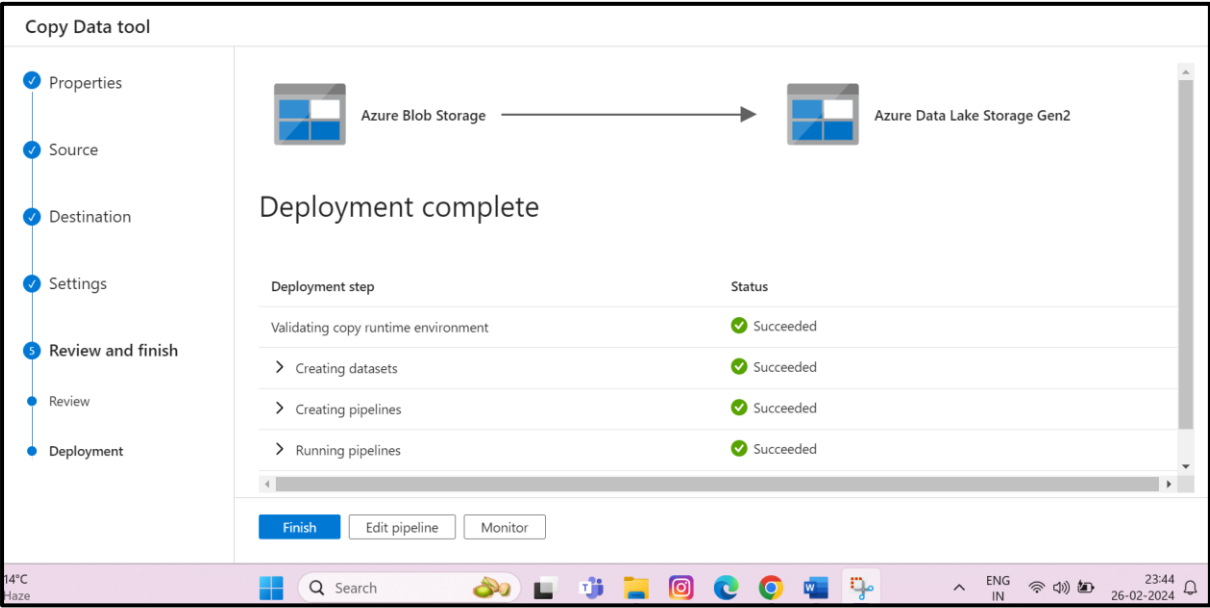




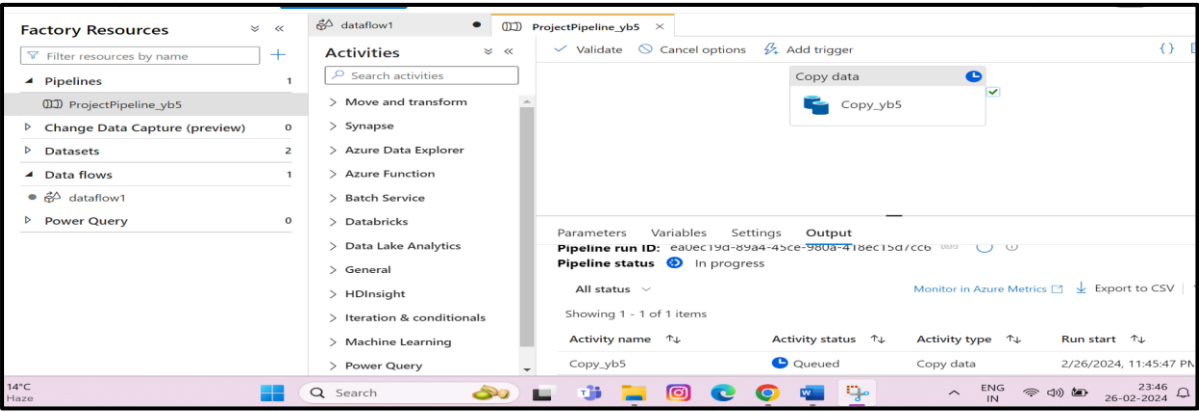
## Pipeline Creation:



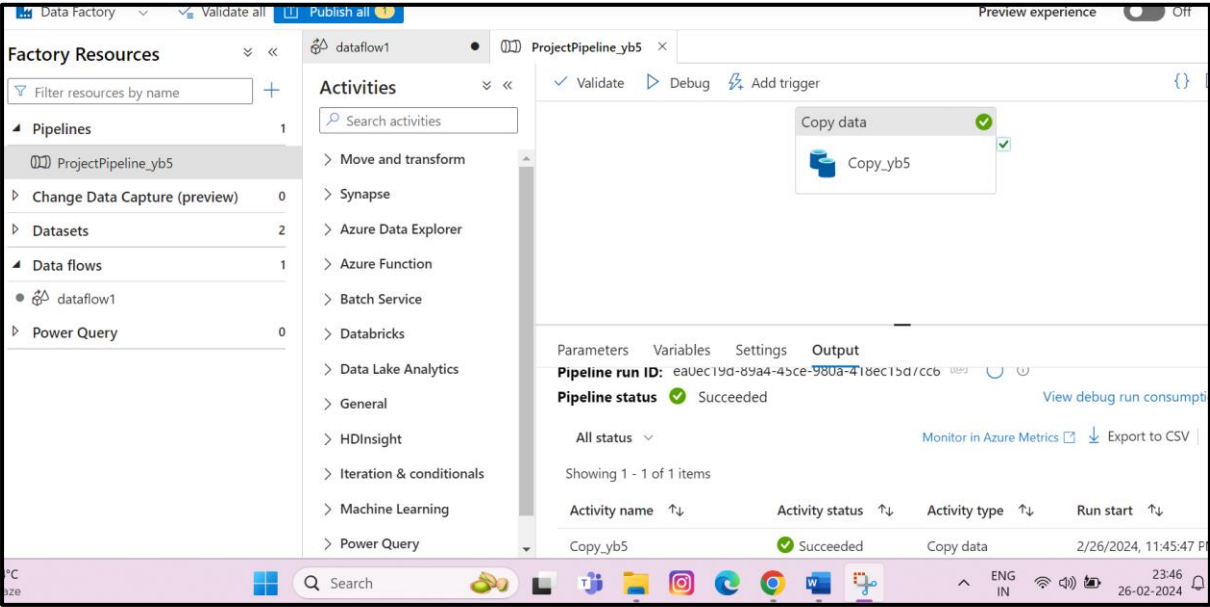
Deployment successful:



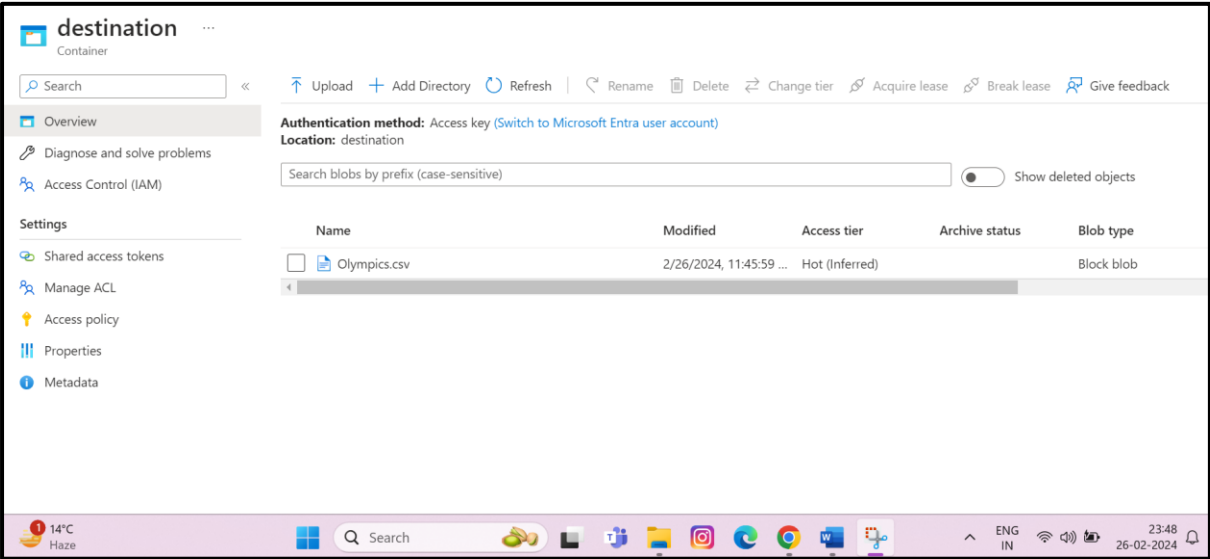
Debugging the pipeline by going to the author tab:



Debugging complete and copy action succeeded:

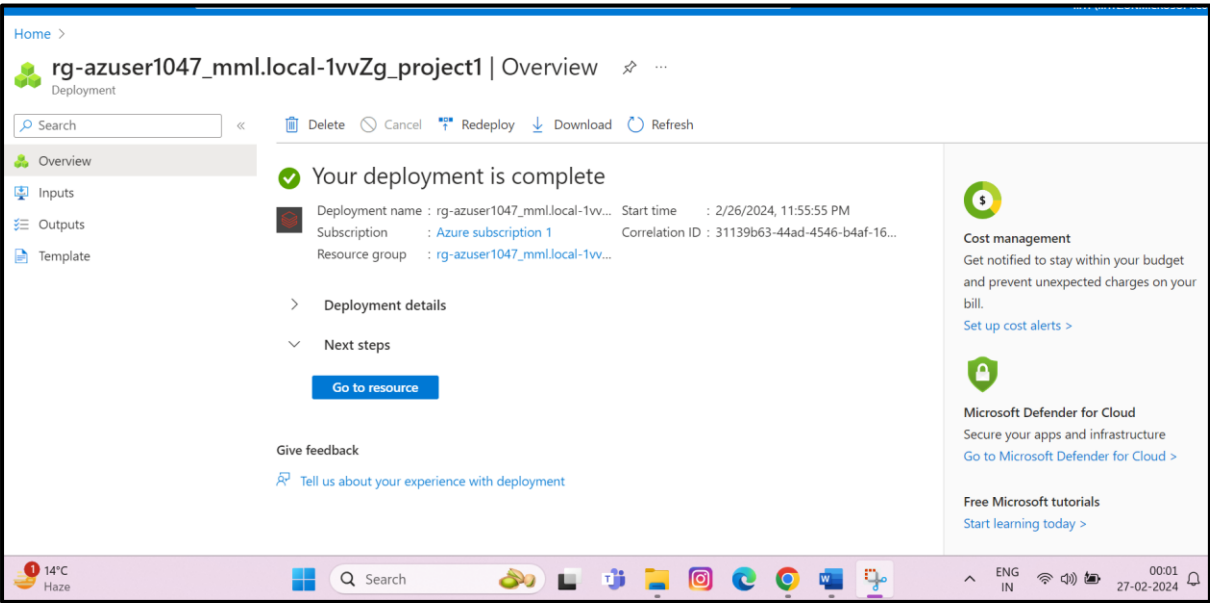


File is copied in the destination folder successfully:



Data Transformations

Creating a Databricks workspace:



## Created a cluster:

Microsoft Azure | databricks | Search data, notebooks, recents, and more... | CTRL + P | project1 | azuser1047\_mml.local@iihtl.onmi

**hello.cluster** [Status: Green Checkmark]

Configuration | Notebooks (0) | Libraries | Event log | Spark UI | Driver logs | Metrics | Apps | Spark compute UI - Master

Policy: Personal Compute

Access mode: Single user access | Single user | azuser1047\_mml.local

Performance

Databricks Runtime Version: 14.3 LTS ML (includes Apache Spark 3.5.0, Scala 2.12)

Node type: Standard\_DS3\_v2 | 14 GB Memory, 4 Cores

Summary

1 Driver | 14 GB Memory, 4 Cores

Runtime: 14.3.x-cpu-ml-scala2.12

Standard\_DS3\_v2 | 0.75 DBU/h

## Connecting Databricks to source file through ADLS for transformations:

Microsoft Azure | databricks | Search data, notebooks, recents, and more... | CTRL + P | project1 | azuser1047\_mml.local@iihtl.onmi

**Project1** | Python | Last edit was 2 minutes ago | New cell UI: ON

File | Edit | View | Run | Help

Run all | hello.cluster | Schedule | Share

Cell 1

```
spark.conf.set(f"fs.azure.sas.destination.adlsproject1hexa.blob.core.windows.net", "sv=2022-11-02&ss=bfqt&srt=sco&sp=rwdlacupyx&se=2024-02-27T02:48:55Z&st=2024-02-26T18:48:55Z&spr=https&sig=MSHXI%2BmaRQ4metmz4bdXf7iDkuOnofeXa5RJcJFVaa4%3D")
```

## Displaying the schema:

Microsoft Azure | databricks | Search data, notebooks, recents, and more... | CTRL + P | project1 | azuser1047\_mml.local@iihtl.onmi

**Project1** | Python | 2/27/2024, 12:21:41 AM | Last edit was 2 minutes ago | New cell UI: ON

File | Edit | View | Run | Help

Run all | hello.cluster | Schedule | Share

Cell 2

```
df = spark.read.csv('wasbs://destination@adlsproject1hexa.blob.core.windows.net/Olympics.csv', header=True, inferSchema=True)
df.show()
```

(3) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 5 more fields]

Games	Year	Country	Type	Cost per event	Cost per athlete	event
Tokyo	1964	Japan	Summer	1.7	0.1	NULL
Munich	1972	Germany	Summer	5.2	0.1	NULL
Montreal	1976	Canada	Summer	30.8	1.0	NULL
Moscow	1980	Soviet Union	Summer	31.2	1.2	NULL
Los Angeles	1984	United States	Summer	3.3	0.1	NULL
Barcelona	1992	Spain	Summer	37.7	1.0	NULL
Atlanta	1996	United States	Summer	15.3	0.4	NULL
Sydney	2000	Australia	Summer	16.8	0.5	NULL
Athens	2004	Greece	Summer	9.8	0.3	NULL
Beijing	2008	China	Summer	22.5	0.6	NULL

## Transformations Done On the Schema

- Renaming columns and dropping null values:

The screenshot shows a Databricks notebook interface for 'Project1'. The code in Cell 3 performs two transformations on a DataFrame: renaming 'Cost per event' to 'Cost\_per\_event' and 'Cost per athlete' to 'Cost\_per\_athlete', and then dropping the 'event' column. The output shows the resulting DataFrame with 5 columns: Games, Year, Country, Type, Cost\_per\_event, and Cost\_per\_athlete. The data is sorted by Year.

```
df = df.withColumnRenamed("Cost per event", "Cost_per_event").withColumnRenamed("Cost per athlete", "Cost_per_athlete")
df_cleaned = df.drop("event")
df_cleaned.show()
```

(1) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 5 more fields]

df\_cleaned: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 4 more fields]

Games	Year	Country	Type	Cost_per_event	Cost_per_athlete
Tokyo	1964	Japan	Summer	1.7	0.1
Munich	1972	Germany	Summer	5.2	0.1
Montreal	1976	Canada	Summer	30.8	1.0
Moscow	1980	Soviet Union	Summer	31.2	1.2
Los Angeles	1984	United States	Summer	3.3	0.1
Barcelona	1992	Spain	Summer	37.7	1.0
Atlanta	1996	United States	Summer	15.3	0.4
Sydney	2000	Australia	Summer	16.8	0.5
Athens	2004	Greece	Summer	9.8	0.3
Beijing	2008	China	Summer	22.5	0.6

- Sort in ascending order:

The screenshot shows a Databricks notebook interface for 'Project1'. The code in Cell 4 drops the 'event' column and sorts the resulting DataFrame by 'Year' in ascending order. The output shows the resulting DataFrame with 6 columns: Games, Year, Country, Type, Cost\_per\_event, and Cost\_per\_athlete. The data is sorted by Year.

```
df1 = df.drop("event")
df1.sort("Year").show()
```

(1) Spark Jobs

df1: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 4 more fields]

Games	Year	Country	Type	Cost_per_event	Cost_per_athlete
Tokyo	1964	Japan	Summer	1.7	0.1
Innsbruck	1964	Austria	Winter	0.6	0.02
Grenoble	1968	France	Winter	25.4	0.8
Munich	1972	Germany	Summer	5.2	0.1
Sapporo	1972	Japan	Winter	3.4	0.1
Montreal	1976	Canada	Summer	30.8	1.0
Innsbruck	1976	Austria	Winter	3.2	0.1
Moscow	1980	Soviet Union	Summer	31.2	1.2
Lake Placid	1980	United States	Winter	11.5	0.4
Los Angeles	1984	United States	Summer	3.3	0.1
Calgary	1988	Canada	Winter	24.1	0.8
Barcelona	1992	Spain	Summer	37.7	1.0

- **Sort in Descending Order:**

The screenshot shows a Jupyter Notebook interface with a Python kernel. The code in the cell is `df1.sort("Year",ascending=False).show()`. The output displays a DataFrame with 16 rows of Olympic Games data, sorted by Year in descending order. The columns are Games, Year, Country, Type, Cost\_per\_event, Cost\_per\_athlete, and event. The data includes games from Rio 2016 down to Albertville 1992.

Games	Year	Country	Type	Cost_per_event	Cost_per_athlete	event
Rio	2016	Brazil	Summer	14.9	0.4	NULL
Sochi	2014	Russia	Winter	223.4	7.9	NULL
London	2012	United Kingdom	Summer	49.5	1.4	NULL
Vancouver	2010	Canada	Winter	29.5	1.0	NULL
Beijing	2008	China	Summer	22.5	0.6	NULL
Torino	2006	Italy	Winter	52.0	1.7	NULL
Athens	2004	Greece	Summer	9.8	0.3	NULL
Salt Lake City	2002	United States	Winter	32.3	1.1	NULL
Sydney	2000	Australia	Summer	16.8	0.5	NULL
Nagano	1998	Japan	Winter	32.7	1.0	NULL
Atlanta	1996	United States	Summer	15.3	0.4	NULL
Lillehammer	1994	Norway	Winter	36.5	1.3	NULL
Barcelona	1992	Spain	Summer	37.7	1.0	NULL
Albertville	1992	France	Winter	35.0	1.1	NULL

- **Filtering:**

The screenshot shows a Jupyter Notebook interface with a Python kernel. The code in the cell is `df1= df.drop("event")` followed by `df1.filter("Cost_per_event > 50").show()`. The output displays a DataFrame with 2 rows of data, filtered by Cost\_per\_event > 50. The columns are Games, Year, Country, Type, Cost\_per\_event, and Cost\_per\_athlete. The data includes Torino 2006 and Sochi 2014.

Games	Year	Country	Type	Cost_per_event	Cost_per_athlete
Torino	2006	Italy	Winter	52.0	1.7
Sochi	2014	Russia	Winter	223.4	7.9

- **Distinct:**

Project1

Python

☆

File

Edit

View

Run

Help

Last edit was now

New cell UI: ON

Run all

hello.cluster

Schedule

Share

Cell 7

Python

Just now (2s)

```
df1= df.drop("event")
df1.select("Cost_per_athlete").distinct().show()
```

(2) Spark Jobs

df1: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 4 more fields]

Cost_per_athlete
1.4
1.7
0.1
1.0
0.6
0.8
0.5
1.3
7.9
0.4

Windows Taskbar

Search

Taskbar Icons

System Tray

## GroupBy with Aggregate Functions

- **Sum:**

Project1Python ↕ ☆

FileEditViewRunHelpLast edit was nowNew cell UI: ON ↕

▶ Run allhello.cluster ↕ScheduleShare

Cell 8Python ✨ ⌵ ⋮

▶ ↕ ✓ Just now (1s)

```
df1= df.drop("event")
df1.groupby("Type").sum("Cost_per_event").show()
```

▶ (2) Spark Jobs

▶ 📄 df1: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 4 more fields]

+-----+	
Type	sum(Cost_per_event)
+-----+	
Summer	238.70000000000002
Winter	509.6
+-----+	

[Shift+Enter] to run and move to next cell

[Esc H] to see all keyboard shortcuts

- **Min:**

The screenshot shows a Databricks notebook interface for 'Project1'. The top bar includes a 'Python' language selector, a star icon, and buttons for 'Run all', 'hello.cluster', 'Schedule', and 'Share'. The menu bar contains 'File', 'Edit', 'View', 'Run', and 'Help'. A status bar at the bottom shows system icons and the time '00:37' on '27-02-2024'.

The notebook content shows a previous cell's output as a table:

Summer	238.70000000000002
Winter	509.6

Below this is 'Cell 9' containing the following Python code:

```
df1= df.drop("event")
df1.groupBy("Type").min("Cost_per_event").show()
```

The output of Cell 9 shows '(2) Spark Jobs' and a DataFrame summary:

```
df1: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 4 more fields]
```

The resulting table is:

Type	min(Cost_per_event)
Summer	1.7
Winter	0.6

At the bottom of the cell, it says '[Shift+Enter] to run and move to next cell'.

- **Max:**

The screenshot shows a Databricks notebook interface for 'Project1'. The top bar includes a 'Python' language selector, a star icon, and buttons for 'Run all', 'hello.cluster', 'Schedule', and 'Share'. The menu bar contains 'File', 'Edit', 'View', 'Run', and 'Help'. A status bar at the bottom shows system icons and the time '00:38' on '27-02-2024'.

The notebook content shows 'Cell 10' containing the following Python code:

```
df1= df.drop("event")
df1.groupBy("Type").max("Cost_per_event").show()
```

The output of Cell 10 shows '(2) Spark Jobs' and a DataFrame summary:

```
df1: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 4 more fields]
```

The resulting table is:

Type	max(Cost_per_event)
Summer	49.5
Winter	223.4

At the bottom of the cell, it says '[Shift+Enter] to run and move to next cell' and '[Esc H] to see all keyboard shortcuts'.



- **Count:**

Project1Python☆

FileEditViewRunHelpLast edit was 1 minute agoNew cell UI: ON

Run allhello.clusterScheduleShare

+ Code+ Text

Just now (1s)Cell 11Python

df1= df.drop("event,")  
df1.groupBy("Type").count().show()

▶ (2) Spark Jobs

▶ df1: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 5 more fields]

+-----+  
| Type|count|  
+-----+  
|Summer| 12|  
|Winter| 13|  
+-----+

[Shift+Enter] to run and move to next cell  
[Esc H] to see all keyboard shortcuts

Search

ENG IN

00:39 27-02-2024

- **Mean:**

Project1Python☆

FileEditViewRunHelpLast edit was nowNew cell UI: ON

Run allhello.clusterScheduleShare

+ Code+ Text

Just now (1s)Cell 12Python

df1= df.drop("event")  
df1.groupBy("Type").mean("Cost\_per\_event").show()

▶ (2) Spark Jobs

▶ df1: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 4 more fields]

+-----+  
| Type|avg(Cost\_per\_event)|  
+-----+  
|Summer| 19.89166666666667|  
|Winter| 39.2|  
+-----+

[Shift+Enter] to run and move to next cell  
[Esc H] to see all keyboard shortcuts

Search

ENG IN

00:40 27-02-2024

- Avg:

The screenshot shows a Databricks notebook interface for 'Project1'. The top bar includes a 'Python' language selector, a star icon, and buttons for 'Run all', 'hello.cluster', 'Schedule', and 'Share'. The menu bar contains 'File', 'Edit', 'View', 'Run', and 'Help'. Below the menu, it says 'Last edit was 1 minute ago' and 'New cell UI: ON'. The notebook content area shows a code cell labeled 'Cell 13' with the following Python code:

```
df1 = df.drop("event")
df1.groupby("Type").avg("Cost_per_event").show()
```

Below the code, it indicates '(2) Spark Jobs' and shows the execution details for 'df1: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 4 more fields]'. The output is a table:

Type	avg(Cost_per_event)
Summer	19.89166666666667
Winter	39.2

At the bottom of the notebook, there are instructions: '[Shift+Enter] to run and move to next cell' and '[Esc H] to see all keyboard shortcuts'. The Windows taskbar at the very bottom shows the search bar, various application icons, and system status icons including language (ENG IN), network, and time (00:41, 27-02-2024).

- Agg

The screenshot shows a Databricks notebook interface for 'Project1'. The top bar includes a 'Python' language selector, a star icon, and buttons for 'Run all', 'hello.cluster', 'Schedule', and 'Share'. The menu bar contains 'File', 'Edit', 'View', 'Run', and 'Help'. Below the menu, it says 'Last edit was now' and 'New cell UI: ON'. The notebook content area shows a code cell labeled 'Cell 14' with the following Python code:

```
from pyspark.sql.functions import sum, min

df1 = df.drop("event")
df1.groupby("Type").agg(sum(df["Cost_per_event"]), min(df["Cost_per_athlete"])).show()
```

Below the code, it indicates '(2) Spark Jobs' and shows the execution details for 'df1: pyspark.sql.dataframe.DataFrame = [Games: string, Year: integer ... 4 more fields]'. The output is a table:

Type	sum(Cost_per_event)	min(Cost_per_athlete)
Summer	238.70000000000002	0.1
Winter	509.6	0.02

At the bottom of the notebook, there are instructions: '[Shift+Enter] to run and move to next cell' and '[Esc H] to see all keyboard shortcuts'. The Windows taskbar at the very bottom shows the search bar, various application icons, and system status icons including language (ENG IN), network, and time (00:42, 27-02-2024).

## CONCLUSION

In conclusion, the integration of Azure Data Factory (ADF) pipelines for data ingestion and Azure Databricks for data transformation provides a comprehensive solution for handling Olympics data within the Azure ecosystem. ADF simplifies and automates the process of bringing in data from diverse sources and depositing it into Azure Data Lake Storage Gen2. This serves as a reliable and scalable data repository.

On the other hand, Azure Databricks, with its PySpark capabilities, facilitates efficient and parallelized data transformations, ensuring that the raw Olympics data is transformed into a structured and analyzable format. The combination of these services not only streamlines the end-to-end data processing workflow but also takes advantage of the cloud's scalability and computational power.

By leveraging these Azure services, organizations can achieve a seamless and robust pipeline for managing, ingesting, and transforming large volumes of data, allowing for enhanced analytics, reporting, and insights. This approach aligns with modern data engineering best practices and empowers users to derive valuable insights from the Olympics data while benefitting from the flexibility and scalability offered by the Azure cloud platform.