Case Study: TechRetail Solutions Architecture

Members: Madhur Agrawal, Arsalan Malik, Dhiraj Paudel

Room - 3

Background

- TechRetail, a mid-sized retail company, wants to create a data pipeline to collect retail data from various sources, process it using advanced analytics, and visualize the results in a dashboard. The goal is to gain insights into sales trends and improve decision-making. The company wants to leverage Azure Databricks for data processing and Microsoft Fabric for data integration and visualization.

Objectives:

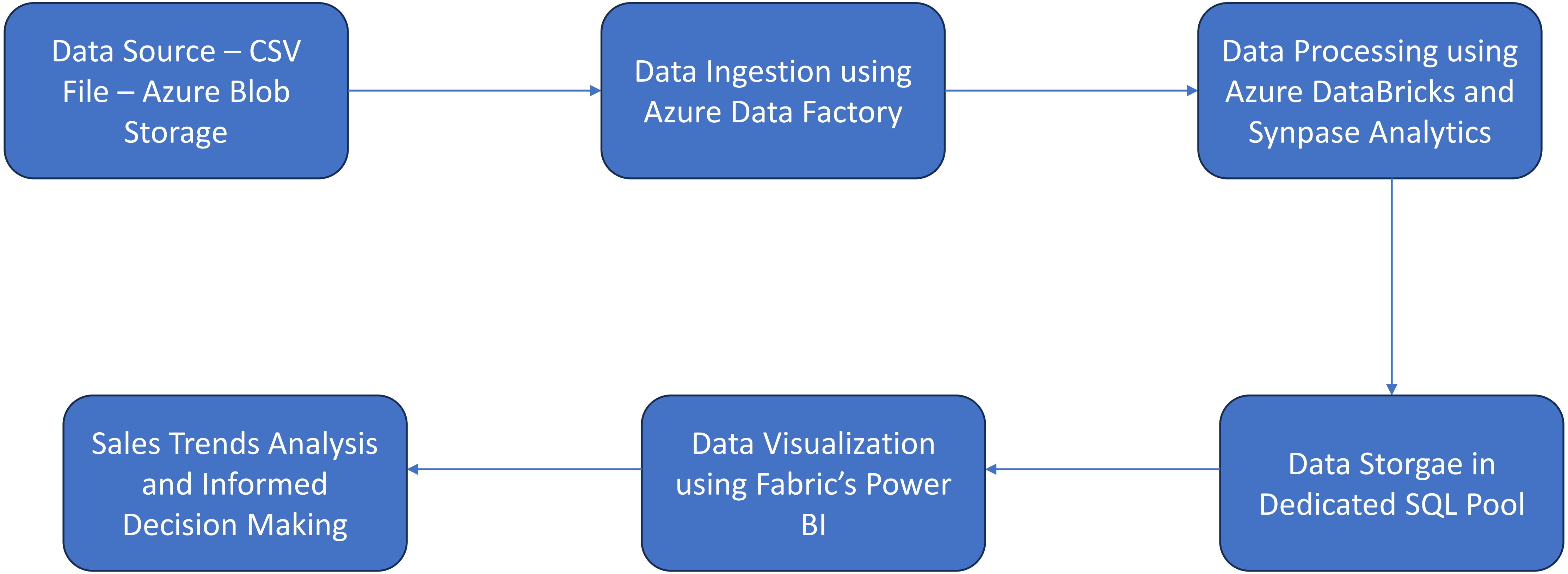
Data Ingestion

Data Processing

Data Storage

Data visualization

Solutions Architecture for TechRetail



Data Source – CSV Stored in Azure Blob Storage

The data is provided in the CSV format which is can be stored in the Azure Blob Storage. The Blob storage is a perfect choice to store the data which can be ingested with several services like Azure Data Factory.

The Blob Storage also provides a scalable and reliable storage solution.

Data Ingestion – Azure Data Factory (ADF)

- Using ADF to automate the data ingestion process from the CSV file.
- ADF can pull data from various sources, including Azure Blob Storage (if the CSV file is stored there) or on other on-premises locations if needed.
- <u>Azure Blob Storage:</u> This will be the staging area where CSV files are initially stored before ingestion. Blob Storage provides high scalability and security for storing raw data files.

Data Exploration and Cleaning

- Using Azure Synapse Analytics for quick data exploration and switch to Spark pools within Synapse to explore further data using Python.
- As data has been directly loaded into Azure Blob, it can be directly accessed within Azure Synapse Studio.
- Utilization of various SQL queries or Spark Notebooks that can handle missing values, duplicate values, data type corrections and other preprocessing task.

Data Processing – Azure Databricks & Azure Synapse Analytics

- Use Azure Databricks for advanced analytics and data processing. Databricks is ideal for handling large datasets, performing complex transformations, and running machine learning algorithms if required
- As an alternative to Databricks, Synapse offers integrated ETL capabilities with both Spark and SQL-based processing, seamlessly connecting with Azure Data Factory for transformation tasks.

Data Storage in the Dedicated SQL Pool of Synapse Analytics

- It serves as the storage layer for processed data, offering optimized analytics capabilities and scalable, multi-tier storage options. Azure Data Lake Storage is suitable for large datasets and complex queries.
- It can also store processed data in a SQL Pool for fast querying, ideal for analytical workloads that require quick SQL-based data access. It also integrates well with visualization tools like Power

Data Visualization using Microsoft Fabric's Power BI

- Microsoft's Fabric Platform provide Power BI solution which can be used for the visualization of the data which is analysed and processed by DataBricks and Synapse Analytics.
- It can directly connect to Azure Synapse which can be used to create the dashboards for the insights from the processed data.
- It can be used to monitor retail insights and trends in the data.
- Scheduled Refresh option is also available to update the visualization based on the updates with data

Analysis Performed over the Processed Data

- **Trend Analysis**
  - How key things (sales, expenses, customer feedback) change over time
  - Line charts to plot values over time and spot incline or decline pattern.
- **Top/Bottom Performers**
  - Identification of top performing and underperforming products
  - Basic table visual to rank entities based on sales and revenue followed by sorting to spot the highest and lowest values.

Analysis Performed over the Processed Data

- **Growth Analysis**
  - Measurement of growth between two time periods (month – month , year – year)
  - Simple calculations to show how sales have changed over specific period
- **Average Calculations**
  - Average of a metric (average sales, average revenue , average ratings)
  - Table to display the average of the metrics for further analysis

Analysis Performed over the Processed Data

- **Min / Max**
  - Identification of smallest and largest values of the dataset
  - Can be used to show max /mins like Maximum / Minimum Sales value.
- **Basic Proportion Analysis(Yes/No)**
  - Customer feedback or cases can be resolved or unresolved , can be used to visualize what percentage of cases have been resolved and what are still open
  - Pie chart to display the percentage of each outcome for quick and easy understanding of distribution.

Informed Decision Making

- The visualized data can be used for the informed decision making by the stakeholders and the business owners.
- The various market trends can be analysed using the interactive dashboards of Power BI which can help in making decisions to help improve sales and grow the business of TechRetail.

Implementation Details

Snapshot of the Code

11:55 AM (3s) 1 Python

```python
# Import necessary libraries
import pandas as pd
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, isnan, count, year, month, dayofweek, hour
from pyspark.sql.types import IntegerType, FloatType, StringType, DateType,
TimestampType

# Initialize Spark session
spark = SparkSession.builder.appName("RetailDataCleaning").getOrCreate()

# Load the CSV file into a Spark DataFrame
file_path = "/FileStore/tables/retail_data.csv"  # Update with your DBFS path
df = spark.read.csv(file_path, header=True,inferSchema=True)

# Display the data structure and some initial records
df.printSchema()
df.show(5)
```

(4) Spark Jobs

- missing_counts: pyspark.sql.dataframe.DataFrame = [Transaction_ID: long, Customer_ID: long ... 28 more fields]
- df: pyspark.sql.dataframe.DataFrame = [Transaction_ID: integer, Customer_ID: integer ... 28 more fields]
- missing_counts_after: pyspark.sql.dataframe.DataFrame = [Transaction_ID: long, Customer_ID: long ... 28 more fields]

Table +

	Transaction_ID	Customer_ID	Name	Email	Phone	Address	City	State	Zipcode
1	333	308	382	347	362	315	248	281	3

1 row | 36.31 seconds runtime

Refreshed 4 hours ago

Snapshot of the Code

12:01 PM (1s) 3

```python
# Convert `Age`, `Income`, and `Ratings` to numeric types
df = df.withColumn("Age", col("Age").cast(IntegerType())) \
       .withColumn("Income", col("Income").cast(FloatType())) \
       .withColumn("Ratings", col("Ratings").cast(FloatType()))

# Convert `Date` and `Time` columns to date and timestamp formats
df = df.withColumn("Date", col("Date").cast(DateType())) \
       .withColumn("Time", col("Time").cast(TimestampType()))

# Display updated schema
df.printSchema()
```

df: pyspark.sql.dataframe.DataFrame = [Transaction_ID: integer, Customer_ID: integer ... 28 more fields]

```
|-- Age: integer (nullable = true)
|-- Gender: string (nullable = true)
|-- Income: float (nullable = true)
|-- Customer_Segment: string (nullable = true)
|-- Date: date (nullable = true)
|-- Year: integer (nullable = true)
|-- Month: string (nullable = true)
|-- Time: timestamp (nullable = true)
|-- Total_Purchases: integer (nullable = true)
|-- Amount: double (nullable = true)
|-- Total_Amount: double (nullable = true)
|-- Product_Category: string (nullable = true)
|-- Product_Brand: string (nullable = true)
|-- Product_Type: string (nullable = true)
|-- Feedback: string (nullable = false)
```

Snapshot of the Code

12:02 PM (2s) 4

```
# Ensure consistent capitalization for `Gender`, `Country`, `Order_Status`, and other key columns
df = df.withColumn("Gender", when(col("Gender").isin("male", "Male"), "Male")
                             .when(col("Gender").isin("female", "Female"), "Female")
                             .otherwise("Unknown"))

# Standardize `Country` column values (example for 'USA', 'UK' variations)
df = df.withColumn("Country", when(col("Country").isin("US", "USA", "United States"), "USA")
                              .when(col("Country").isin("UK", "United Kingdom"), "UK")
                              .otherwise(col("Country")))

# Verify the transformations
df.select("Gender", "Country", "Order_Status").distinct().show()
```

(2) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [Transaction_ID: integer, Customer_ID: integer ... 28 more fields]

```
|   Male|Australia|  Processing|
| Female|       UK|     Pending|
|   Male|  Germany|   Delivered|
| Female|       UK|     Shipped|
| Female|   Canada|  Processing|
|   Male|  Germany|  Processing|
|   Male|   Canada|  Processing|
|   Male|       UK|  Processing|
|   Male|       UK|     Shipped|
|   Male|       UK|        NULL|
| Female|       UK|  Processing|
|Unknown|       UK|     Shipped|
|   Male|       UK|   Delivered|
| Female|       UK|        NULL|
|Unknown|       UK|   Delivered|
| Female|Australia|     Pending|
|Unknown|       UK|     Pending|
|   Male|  Germany|     Shipped|
+-------+---------+------------+
only showing top 20 rows
```

Snapshot of the Code

12:01 PM (1s) 3

```
# Convert `Age`, `Income`, and `Ratings` to numeric types
df = df.withColumn("Age", col("Age").cast(IntegerType())) \
       .withColumn("Income", col("Income").cast(FloatType())) \
       .withColumn("Ratings", col("Ratings").cast(FloatType()))

# Convert `Date` and `Time` columns to date and timestamp formats
df = df.withColumn("Date", col("Date").cast(DateType())) \
       .withColumn("Time", col("Time").cast(TimestampType()))

# Display updated schema
df.printSchema()
```

df: pyspark.sql.dataframe.DataFrame = [Transaction_ID: integer, Customer_ID: integer ... 28 more fields]

```
|-- Age: integer (nullable = true)
|-- Gender: string (nullable = true)
|-- Income: float (nullable = true)
|-- Customer_Segment: string (nullable = true)
|-- Date: date (nullable = true)
|-- Year: integer (nullable = true)
|-- Month: string (nullable = true)
|-- Time: timestamp (nullable = true)
|-- Total_Purchases: integer (nullable = true)
|-- Amount: double (nullable = true)
|-- Total_Amount: double (nullable = true)
|-- Product_Category: string (nullable = true)
|-- Product_Brand: string (nullable = true)
|-- Product_Type: string (nullable = true)
|-- Feedback: string (nullable = false)
```

Snapshot of the Code

12:02 PM (2s) 6

```python
# Create Age Bins for segmentation
df = df.withColumn("Age_Group", when(col("Age") < 18, "<18")
                   .when((col("Age") >= 18) & (col("Age") <= 25), "18-25")
                   .when((col("Age") >= 26) & (col("Age") <= 35), "26-35")
                   .when((col("Age") >= 36) & (col("Age") <= 45), "36-45")
                   .when((col("Age") >= 46) & (col("Age") <= 60), "46-60")
                   .otherwise(">60"))

# Extract additional time-based features from the Date column
df = df.withColumn("Year", year(col("Date"))) \
       .withColumn("Month", month(col("Date"))) \
       .withColumn("Day_of_Week", dayofweek(col("Date")))

# Extract hour from the Time column for hourly analysis
df = df.withColumn("Hour", hour(col("Time")))

# Show the new columns
df.select("Age_Group", "Year", "Month", "Day_of_Week", "Hour").show(5)
```

(1) Spark Jobs

(1) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [Transaction_ID: integer, Customer_ID: integer ... 28 more fields]

```
-------+------------+---------+---------------+---------------+------------+-------+-------+-----------------+
|       8691788|      37249|Michelle Harrington|  Ebony39@gmail.com|1414786801|   3959 Amanda Burgs|  Dortmund|         Berlin|  77985|  Germa
ny| 21|  Male|  NULL|         Regular|NULL|2023|September|2024-11-11 22:03:55|              3|108.0287567|   324.08627|         Clothing|
Nike|       Shorts|Excellent|        Same-Day|    Debit Card|     Shipped|     5.0|   Cycling shorts|
|       2174773|      69749|       Kelsey Hill|   Mark36@gmail.com|6852899987|  82072 Dawn Centers|Nottingham|        England|  99071|
UK| 19|Female|  NULL|         Premium|NULL|2023| December|2024-11-11 08:42:04|              2|403.3539073| 806.7078147|      Electronics|
Samsung|       Tablet|Excellent|        Standard|   Credit Card|  Processing|     4.0|       Lenovo Tab|
|       6679610|      30192|      Scott Jensen|  Shane85@gmail.com|8362160449|   4133 Young Canyon|   Geelong|New South Wales|  75929|Austral
ia| 48|  Male|  NULL|         Regular|NULL|2023|    April|2024-11-11 04:06:29|              3|354.4775997| 1063.432799|            Books|Pengui
n Books|  Children's|  Average|        Same-Day|   Credit Card|  Processing|     2.0| Sports equipment|
|       7232460|      62101|     Joseph Miller|   Mary34@gmail.com|2776751724|8148 Thomas Creek...|  Edmonton|        Ontario|  88420|   Cana
da| 56|  Male|  NULL|         Premium|NULL|2023|      May|2024-11-11 14:55:17|              7|352.4077173| 2466.854021|       Home Decor|    Hom
e Depot|       Tools|Excellent|        Standard|        PayPal|  Processing|     4.0|    Utility knife|
|       4983775|      27901|     Debra Coleman|Charles30@gmail.com|9098267635|5813 Lori Ports S...|   Bristol|        England|  48704|
UK| 22|  Male|  NULL|         Premium|NULL|2024|  January|2024-11-11 16:54:07|              2|124.2765245| 248.5530491|          Grocery|
Nestle|   Chocolate|       Bad|        Standard|          Cash|     Shipped|     1.0|Chocolate cookies|
+--------------+-----------+-------------------+-------------------+----------+--------------------+----------+---------------+-------+------
--+---+------+------+----------------+----+----+---------+-------------------+---------------+-----------+------------+-----------------+------
-------+------------+---------+---------------+---------------+------------+-------+-------+-----------------+
only showing top 5 rows
```

Snapshot of the Code

12:02 PM (1s) 5 Python

```python
# Define thresholds for detecting outliers in numerical columns (e.g., Age, Income)
age_upper_limit = 100
income_upper_limit = 200000

# Filter out or replace unrealistic values in Age and Income
df = df.withColumn("Age", when((col("Age") < 0) | (col("Age") > age_upper_limit), None).otherwise(col("Age")))
df = df.withColumn("Income", when((col("Income") < 0) | (col("Income") > income_upper_limit), None).otherwise(col("Income")))

# Replace extreme values in `Ratings` (keeping it between 1-5)
df = df.withColumn("Ratings", when(col("Ratings") > 5, 5).when(col("Ratings") < 1, 1).otherwise(col("Ratings")))

# Show updated data after handling outliers
df.show(5)
```

df: pyspark.sql.dataframe.DataFrame = [Transaction_ID: integer, Customer_ID: integer ... 31 more fields]

```
+---------+----+-----+-----------+----+
|Age_Group|Year|Month|Day_of_Week|Hour|
+---------+----+-----+-----------+----+
|    18-25|NULL| NULL|       NULL|  22|
|    18-25|NULL| NULL|       NULL|   8|
|    46-60|NULL| NULL|       NULL|   4|
|    46-60|NULL| NULL|       NULL|  14|
|    18-25|NULL| NULL|       NULL|  16|
+---------+----+-----+-----------+----+
only showing top 5 rows
```

Snapshot of the Code

```
# Save the cleaned and transformed data back to Azure Blob or any designated storage
output_path = "/FileStore/tables/retail_data.csv"  # Update with your DBFS path
df.write.mode("overwrite").parquet(output_path)
print("Data cleaning and transformation completed. File saved to:", output_path)
```

Snapshot of the Visuals