Building an End-to-End Retail Analytics Platform

Capstone Project

By - Shivendra Tripathi (AS1550)

The Challenge for Modern Retail

- Retail companies struggle with integrating in-store POS transactions, online orders, and inventory systems into a single analytics-ready platform.
- This data is often siloed, making it difficult to get a single, analytics-ready view.
- **Key Business Needs:**
  - Batch processing for daily sales reconciliation.
  - Real-time ingestion for fraud detection & stock alerts.
  - A cloud-native platform that can scale for peak seasons.
  - A unified warehouse (Snowflake) for advanced reporting.
  - Strong data security for customer PII.

SIGMOID

Project Objectives & Key Deliverables

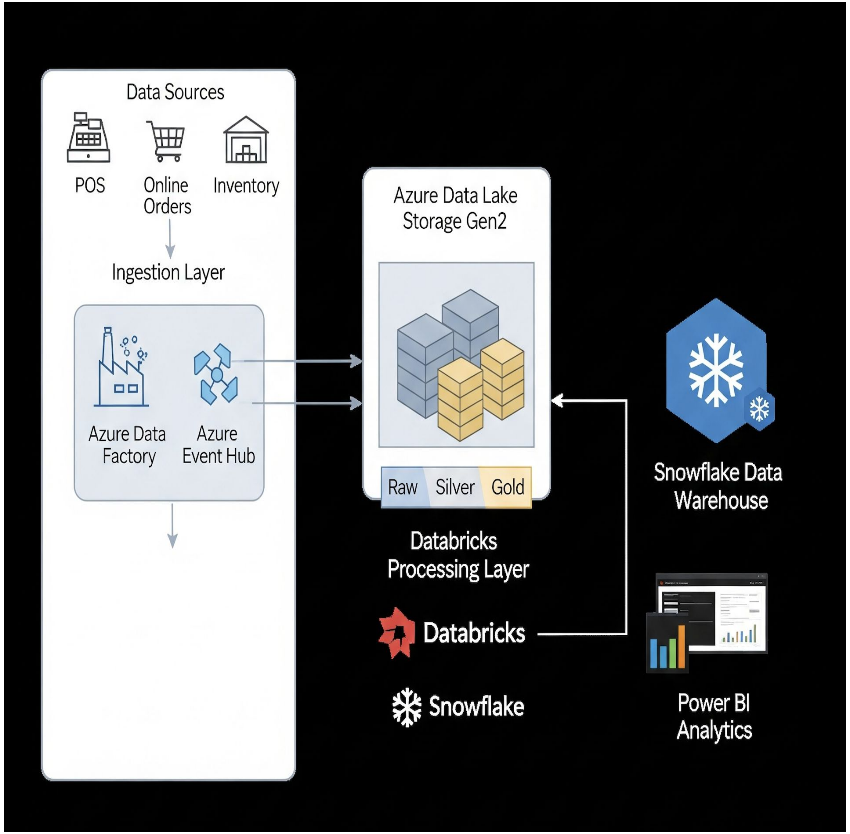
- The goal was to build a resilient, secure, and scalable data platform to meet key business user stories .

- **Key Deliverables:**
  - A Data Lake with Raw, Processed, and Curated zones for orders, customers, and products.
  - Azure Data Factory (ADF) pipelines for batch ingestion of POS data.
  - Databricks notebooks for cleansing, enrichment, and Delta Lake transformations.
  - A Snowflake warehouse with fact and dimension tables.
  - BI dashboards showing Daily/Monthly Sales, Top-selling products, and Inventory shortages.
  - Real-time alerts for suspicious transactions.
  - PII masking for customer email addresses and mobile numbers.

SIGMOID

The Solution: High-Level Architecture

- We implemented a modern Medallion Architecture on Azure.
- Data flows from source systems, is ingested by ADF, and lands in ADLS Gen2.
- Databricks processes the data from a Bronze, to a Silver, to a Gold layer.
- The final, curated Gold data is loaded into a Snowflake warehouse.
- Power BI connects to Snowflake for all reporting and analytics.



SIGMOID

The Foundation: ADLS Gen2 & The Medallion Architecture

The Medallion Architecture organizes data into three distinct quality zones:

Bronze (Raw Zone):

- Ingested data in its raw, source format (e.g., CSV, JSON).
- Serves as the single source of truth for all data.

Silver (Processed Zone):

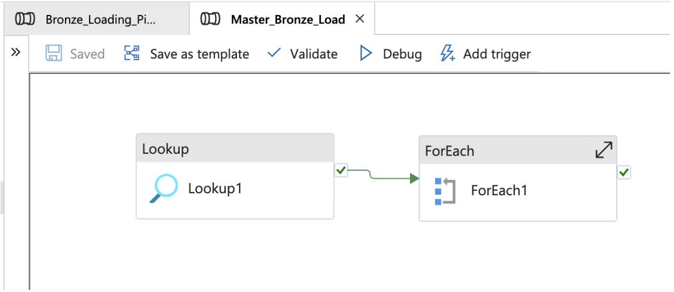
- Contains cleansed, validated, and enriched data.
- PII data is masked here to enforce security.
- This is where we created our clean dimension tables (`DimCustomers`, `DimStores`).

Gold (Curated Zone):

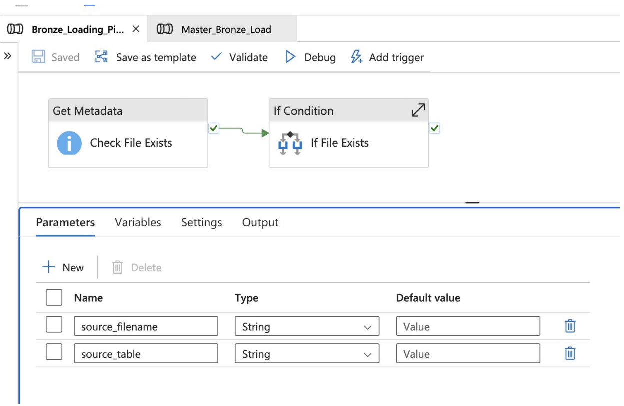
- Business-ready, aggregated tables optimized for analytics.
- This layer directly feeds our BI dashboards.

SIGMOID

Step 1: Automated Batch Ingestion (ADF)



Master Pipeline



Worker Pipeline

- We used Azure Data Factory (ADF) for robust, automated ingestion of batch data.
- A 'Master' pipeline was built to read a JSON configuration file and loop.
- It calls a 'Worker' pipeline for each table, making the process reusable and easy to maintain.
- Azure Monitor alerts were configured to send an email on any pipeline failure.
- ADF was configured with Github for version control.

Step 2: Transformation with Databricks (Bronze -> Silver)

CustomerID	FirstName	LastName	Email	Phone
1	Vivaan	Singh	XXXX@example.com	XXXXXXXX7923
2	Pooja	Verma	XXXX@example.com	XXXXXXXX0525
3	Advika	Patel	XXXX@example.com	XXXXXXXX3256
4	Ananya	Chettri	XXXX@example.com	XXXXXXXX6825
5	Vihaan	Singh	XXXX@example.com	XXXXXXXX5019
6	Riya	Bose	XXXX@example.com	XXXXXXXX1978
7	Kiran	Menon	XXXX@example.com	XXXXXXXX6473
8	Vikash	Patel	XXXX@example.com	XXXXXXXX5938
9	Sneha	Das	XXXX@example.com	XXXXXXXX9081
10	Sneha	Nair	XXXX@example.com	XXXXXXXX0147
11	Sneha	Yadav	XXXX@example.com	XXXXXXXX0825
12	Aarohi	Chettri	XXXX@example.com	XXXXXXXX0484
13	Pooja	Iyer	XXXX@example.com	XXXXXXXX9272
14	Kiran	Nair	XXXX@example.com	XXXXXXXX9853
15	Vikash	Das	XXXX@example.com	XXXXXXXX6184

- Core transformation logic was built in Databricks notebooks.
- **Cleansing:** Data types were corrected and null values were handled.
- **Security:** Fulfilled a key security deliverable by masking PII columns (e.g., `john.doe@email.com` -> `****@email.com`).
- **Enrichment:** We joined the orders table with customers, stores, and products to create a single, wide `silver_fact_orders` table.

SIGMOID

Step 3: Building the Gold Layer (For BI)

```python
# 1. Daily Sales Summary
df_silver_base = spark.table("silver_fact_orders")
df_gold_daily_sales = df_silver_base \
    .withColumn("OrderDate", date_format(col("OrderDateTime"), "yyyy-MM-dd")) \
    .groupBy("OrderDate", "StoreName", "Region", "Category") \
    .agg(
        round(sum("OrderTotal"),2).alias("TotalRevenue"),
        sum("Quantity").alias("TotalQuantity")
    )
```

```
+----------+---------+-------+-----------+------------+-------------+
|OrderDate |StoreName|Region |Category   |TotalRevenue|TotalQuantity|
+----------+---------+-------+-----------+------------+-------------+
|2025-05-06|Store-006|North  |Electronics|271.72      |4            |
|2025-06-17|Store-010|West   |Grocery    |446.33      |7            |
|2025-06-13|Store-011|North  |Electronics|1557.32     |46           |
|2025-04-07|Store-010|West   |Home       |446.53      |9            |
|2025-07-05|Store-005|West   |Apparel    |386.69      |7            |
|2025-07-08|Store-008|Central|Electronics|404.41      |6            |
|2025-07-12|Store-008|Central|Health     |156.2       |3            |
|2025-06-12|Store-012|West   |Electronics|366.42      |5            |
|2025-07-07|Store-001|North  |Home       |250.95      |4            |
|2025-05-06|Store-002|East   |Apparel    |129.96      |3            |
+----------+---------+-------+-----------+------------+-------------+
only showing top 10 rows
```

- The Gold layer is designed for performance. It contains pre-aggregated summary tables.
- Querying the full `silver_fact_orders` table is slow for dashboards.
- We created tables like `gold_daily_sales_summary` by pre-calculating metrics (e.g., `SUM(Revenue)`) by date and store.
- This makes our Power BI dashboards load instantly and answers the Store Manager's user story.

SIGMOID

Real-Time Ingestion (Event Hubs & Streaming)

	body
1	{"event_time": "2025-08-15T00:00:00", "event_type": "order_created", "payload": {"OrderID": "S17552160000", "StoreID"...
2	{"event_time": "2025-08-15T00:00:30", "event_type": "order_created", "payload": {"OrderID": "S17552160301", "StoreID"...
3	{"event_time": "2025-08-15T00:01:00", "event_type": "inventory_update", "payload": {"StoreID": 7, "ProductID": 233, "Delt...
4	{"event_time": "2025-08-15T00:01:30", "event_type": "inventory_update", "payload": {"StoreID": 9, "ProductID": 8, "Delta...
5	{"event_time": "2025-08-15T00:02:00", "event_type": "inventory_update", "payload": {"StoreID": 9, "ProductID": 1, "Delta...
6	{"event_time": "2025-08-15T00:02:30", "event_type": "order_created", "payload": {"OrderID": "S17552161505", "StoreID"...
7	{"event_time": "2025-08-15T00:03:00", "event_type": "inventory_update", "payload": {"StoreID": 9, "ProductID": 231, "Delt...
8	{"event_time": "2025-08-15T00:03:30", "event_type": "order_created", "payload": {"OrderID": "S17552162107", "StoreID"...
9	{"event_time": "2025-08-15T00:04:00", "event_type": "order_created", "payload": {"OrderID": "S17552162408", "StoreID"...
10	{"event_time": "2025-08-15T00:04:30", "event_type": "order_created", "payload": {"OrderID": "S17552162709", "StoreID"...
11	{"event_time": "2025-08-15T00:05:00", "event_type": "inventory_update", "payload": {"StoreID": 8, "ProductID": 243, "Delt...
12	{"event_time": "2025-08-15T00:05:30", "event_type": "order_created", "payload": {"OrderID": "S175521633011", "StoreID...
13	{"event_time": "2025-08-15T00:06:00", "event_type": "order_created", "payload": {"OrderID": "S175521636012", "StoreID...
14	{"event_time": "2025-08-15T00:06:30", "event_type": "order_created", "payload": {"OrderID": "S175521639013", "StoreID...
15	{"event_time": "2025-08-15T00:07:00", "event_type": "order_created", "payload": {"OrderID": "S175521642014", "StoreID...

- To meet the requirement for real-time alerts, we built a parallel streaming pipeline.
- A '*event-producer*' notebook simulated live `order_created` and `inventory_update` events by sending them to an Azure Event Hub.
- A '*streaming-consumer*' notebook then used Databricks Structured Streaming to read this live data from the Event Hub.

Step 5: Real-Time Logic (Fraud & Inventory)

```python
# Defining the MERGE function
def upsert_inventory(micro_batch_df, batch_id):
    inventory_table = DeltaTable.forPath(spark, inventory_table_path)
    inventory_table.alias("target") \
        .merge(
            micro_batch_df.alias("source"),
            "target.StoreID = source.StoreID AND target.ProductID = source.ProductID"
        ) \
        .whenMatchedUpdate(
            set = { "OnHandQty": col("target.OnHandQty") + col("source.TotalDeltaQty") }
        ) \
        .whenNotMatchedInsert(
            values = {
                "StoreID": col("source.StoreID"),
                "ProductID": col("source.ProductID"),
                "OnHandQty": col("source.TotalDeltaQty")
            }
        ) \
        .execute()
```

```sql
%sql
SELECT OnHandQty AS Before_Quantity
FROM bronze_inventory
WHERE StoreID = 6 AND ProductID = 40
ORDER BY SnapshotDate DESC
LIMIT 1
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [B

Table +

	Before_Quantity
1	73

```sql
%sql
SELECT OnHandQty AS After_Quantity
FROM gold_current_inventory
WHERE StoreID = 6 AND ProductID = 40
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [Af

Table +

	After_Quantity
1	69

The consumer notebook split the stream to handle two use cases:

1. Fraud Detection:

- The stream filtered for `order_created` events.
- A rule was applied (e.g., `OrderValue > 3000`) to flag abnormal orders instantly.
- These alerts were written in real-time to the `gold_fraud_alerts` table.

2. Inventory Updates:

- The stream filtered for `inventory_update` events.
- It used a `MERGE INTO` command to update the `gold_current_inventory` table in near-real-time.

Step 6: The Unified Data Warehouse (Snowflake)

RETAIL_DB.ANALYTICS   Settings

```sql
CREATE OR REPLACE WAREHOUSE COMPUTE_WH
WITH
WAREHOUSE_SIZE='MEDIUM'
AUTO_SUSPEND=900
AUTO_RESUME=TRUE;

CREATE OR REPLACE DATABASE RETAIL_DB;
CREATE OR REPLACE SCHEMA ANALYTICS;

SHOW TABLES;
```

Results   Chart

	cre	name	database_name	schema_name	kind
1	2025-	DIM_CUSTOMERS	RETAIL_DB	ANALYTICS	TABLE
2	2025-	DIM_PAYMENTS	RETAIL_DB	ANALYTICS	TABLE
3	2025-	DIM_PRODUCTS	RETAIL_DB	ANALYTICS	TABLE
4	2025-	DIM_STORES	RETAIL_DB	ANALYTICS	TABLE
5	2025-	FACT_ORDERS	RETAIL_DB	ANALYTICS	TABLE
6	2025-	GOLD_CURRENT_INVEN	RETAIL_DB	ANALYTICS	TABLE
7	2025-	GOLD_CUSTOMER_TIER	RETAIL_DB	ANALYTICS	TABLE
8	2025-	GOLD_DAILY_FRAUD_SU	RETAIL_DB	ANALYTICS	TABLE
9	2025-	GOLD_DAILY_SALES_SU	RETAIL_DB	ANALYTICS	TABLE
10	2025-	GOLD_FRAUD_ALERTS	RETAIL_DB	ANALYTICS	TABLE
11	2025-	GOLD_MONTHLY_PROD	RETAIL_DB	ANALYTICS	TABLE
12	2025-	GOLD_SALES_BY_CHAN	RETAIL_DB	ANALYTICS	TABLE

- All 12 of our final Silver and Gold tables were loaded from Databricks into Snowflake.
- **Why Snowflake?** It serves as our 'Unified Warehouse' and the single source of truth for all business users.
- It separates compute and storage, so our Power BI dashboard queries (`BI_WH`) don't slow down data science workloads (`DS_WH`).
- It provides a simple, high-performance SQL endpoint for Power BI.

SIGMOID

Dashboard 1: Real-Time Operations Dashboard

STOREID	PRODUCTID	ONHANDQTY	REORDERPOINT
1.00	46.00	44.00	45.00
1.00	241.00	34.00	41.00
2.00	40.00	9.00	41.00
2.00	46.00	47.00	57.00
2.00	131.00	32.00	41.00
3.00	22.00	43.00	44.00
3.00	61.00	30.00	47.00
3.00	118.00	30.00	47.00
3.00	190.00	29.00	40.00
3.00	245.00	33.00	49.00
4.00	88.00	18.00	38.00
5.00	103.00	42.00	51.00
6.00	136.00	45.00	50.00
6.00	203.00	55.00	56.00
7.00	24.00	40.00	41.00
7.00	161.00	41.00	46.00
8.00	25.00	24.00	33.00
9.00	207.00	22.00	47.00
12.00	52.00	41.00	51.00
12.00	213.00	20.00	46.00

This dashboard provides live alerts for the operations team.

Suspicious Transactions:

- A live table of all orders flagged for fraud (from `gold_fraud_alerts`).

Inventory Shortages:

- A filtered table showing *only* items that need re-stocking.
- We built this using a DAX measure (`IsShortage`) to filter where `OnHandQty < ReorderPoint`.

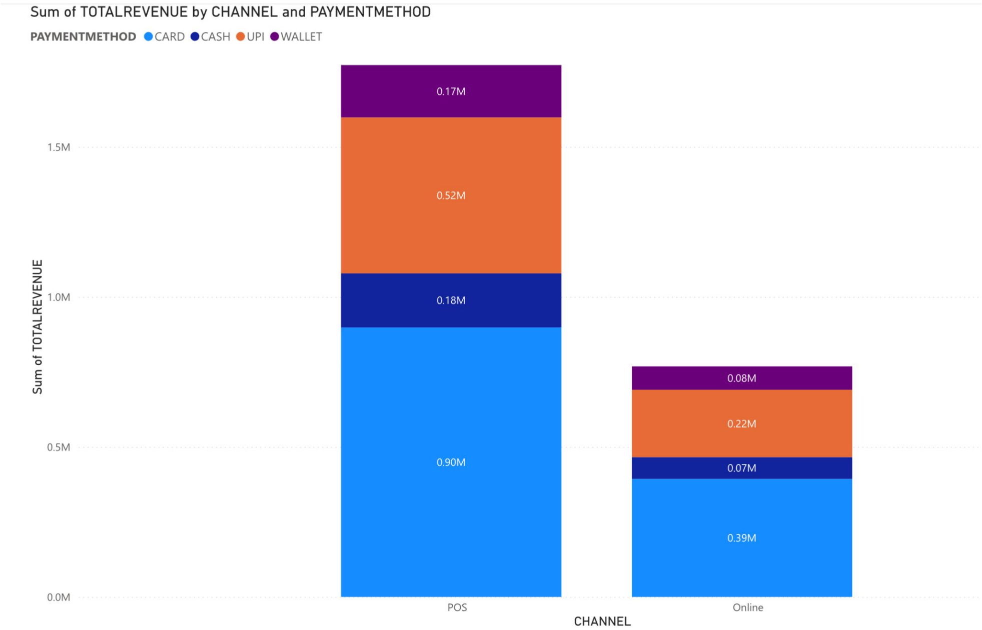
SIGMOID

Dashboard 2: Payment Channel Dashboard

This dashboard provides deeper insights for marketing and finance.

It shows:

- Sales by Channel (Online vs. POS).
- Popularity of Payment Methods (Cash, Card, UPI).



SIGMOID

Project Summary & Skills Developed

- **Successfully built** an end-to-end, automated, and scalable retail data platform.
- **Data Engineering:** Designed batch (ADF) and streaming (Event Hub) pipelines.
- **Data Transformation:** Implemented a robust Medallion Architecture using Databricks and Delta Lake.
- **Data Warehousing:** Modeled and loaded a Star Schema (Silver) and Data Marts (Gold) into Snowflake.
- **Data Security:** Implemented PII masking for customer data.
- **BI & Analytics:** Delivered actionable dashboards in Power BI that serve multiple user stories.

SIGMOID

Future Work & Conclusion

This platform is now a solid foundation for more advanced analytics.

Future Work:

- Implement **Snowpark ML** to build a demand forecasting model using our `gold_daily_sales_summary` table.
- Create a detailed **Cost Estimation** playbook for all Azure and Snowflake services used.

Conclusion: We successfully transformed siloed, raw data from multiple sources into a unified, secure, and real-time source of truth that directly drives business decisions.

SIGMOID

Thank You!