Project 1

E-commerce System Data Pipeline Projects

3.1 Project: Orders & Payments Ingestion and JSON Flattening

By team 9 :
Ahmed Hesham

Laila Mohamed

Youseef Elgammal

1. Overview

The goal of this project is to build a data pipeline that ingests semi-structured JSON Orders data, transforms it into a structured relational format, and makes it queryable for analytics.

We used:
- Python → to generate sample JSON files.
- Apache NiFi → for data ingestion.
- HDFS → for storage.
- Apache Spark (PySpark) → for data transformation.
- Apache Hive → for staging and SQL-based analytics.

2. JSON Data Generation (Python)

Before ingestion, we generated JSON order files using Python. Each JSON file represents an order and contains order_id, payment_status, and an items array (with product_id and sales_quantity).

2.Python Script

This script created 100 JSON files, each representing one order with nested items.

```
[student@localhost ~]$ mkdir -p ~/sample_orders/input
[student@localhost ~]$ python3 - <<'EOF'
> import json, os, random, datetime
>
> input_dir = os.path.expanduser('~/sample_orders/input')
>
> products = [101, 102, 103, 104, 105]
>
> for i in range(1, 101):  # 100 files
>     order = {
>         "order_id": i,
>         "customer_id": random.randint(1000, 2000),
>         "order_date": datetime.date.today().isoformat(),
>         "total_value": random.randint(20, 500),
>         "payment_status": random.choice(["PAID", "PENDING", "FAILED"]),
>         "items": [
>             {
>                 "product_id": random.choice(products),
>                 "sales_quantity": random.randint(1, 5)
>             }
>             for _ in range(random.randint(1, 3))  # 1-3 items per order
>         ]
>     }
>     file_path = os.path.join(input_dir, f"order_{i}.json")
>     with open(file_path, "w") as f:
>         json.dump(order, f)
> EOF
[student@localhost ~]$
[student@localhost ~]$ pyspark
```

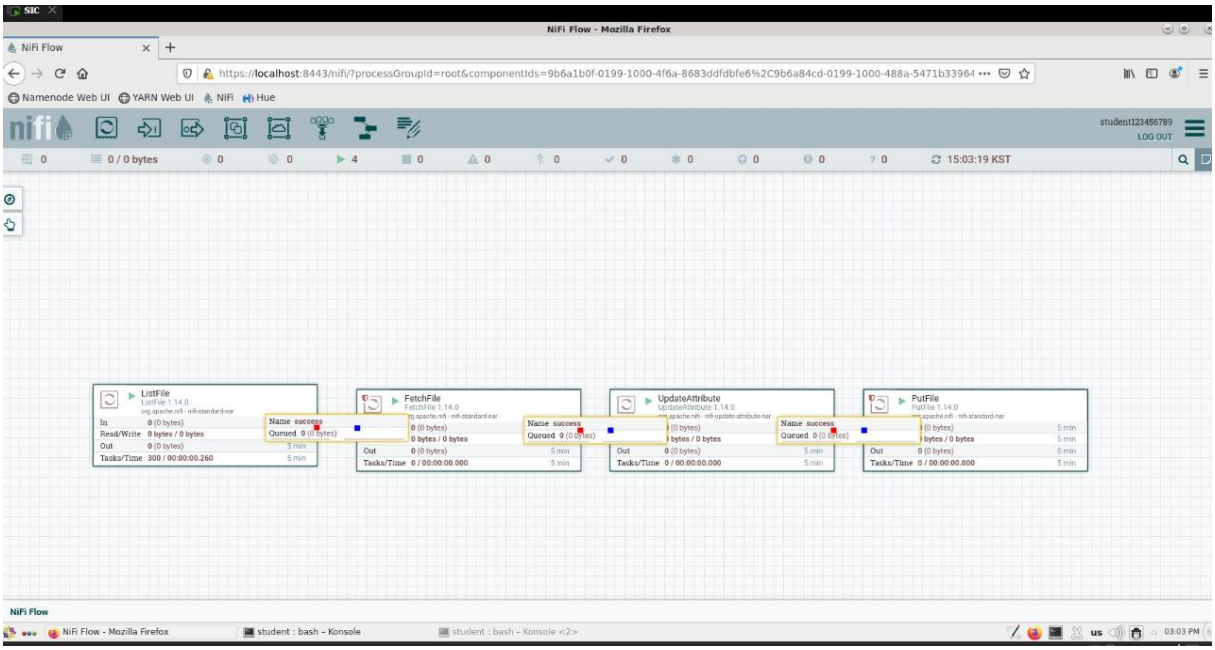
3. NiFi: Data Ingestion

We built a NiFi flow to move JSON files from input → output with metadata.

Flow: ListFile → FetchFile → UpdateAttribute → PutFile

- ListFile → scans /home/<username>/sample_orders/input.
- FetchFile → fetches JSON file contents.
- UpdateAttribute → adds ingestion_time.
- PutFile → writes ingested JSON files into /home/<username>/sample_orders/output.

At this stage → JSON files were successfully ingested and ready for big data processing.



4. HDFS: Storage

The ingested JSON files were uploaded into HDFS:

hdfs dfs -mkdir -p /data/orders
hdfs dfs -put ~/sample_orders/output/* /data/orders/

5. Spark: Transformation

We used PySpark to:

1. Read the JSON files from HDFS.
2. Explode the nested items array.
3. Flatten fields (order_id, product_id, sales_quantity, payment_status).
4. Add a generated unique order_line_id.
5. Write the transformed data back to HDFS in Parquet format for Hive.

```
In [4]: order_lines_df = df.withColumn("item", explode(col("items")))
```

```
In [7]: staging_df = order_lines_df.select(
            concat_ws("_", col("order_id").cast("string"), col("item.product_id").cast("string")).alias("order_line_id"),
            col("order_id"),
            col("item.product_id").alias("product_id"),
            col("item.sales_quantity").alias("sales_quantity"),
            col("payment_status")
        )
```

```
In [8]: staging_df.show(20, truncate=False)
```

```
+-------------+--------+----------+--------------+--------------+
|order_line_id|order_id|product_id|sales_quantity|payment_status|
+-------------+--------+----------+--------------+--------------+
|10_105       |10      |105       |5             |PENDING       |
|10_101       |10      |101       |1             |PENDING       |
|10_105       |10      |105       |3             |PENDING       |
|12_103       |12      |103       |4             |PENDING       |
|12_104       |12      |104       |3             |PENDING       |
|12_101       |12      |101       |3             |PENDING       |
|39_102       |39      |102       |1             |PENDING       |
|39_102       |39      |102       |4             |PENDING       |
|39_103       |39      |103       |3             |PENDING       |
|49_104       |49      |104       |5             |PENDING       |
|49_104       |49      |104       |1             |PENDING       |
|49_103       |49      |103       |2             |PENDING       |
|27_105       |27      |105       |4             |PENDING       |
|27_104       |27      |104       |5             |PENDING       |
|27_105       |27      |105       |4             |PENDING       |
|56_105       |56      |105       |3             |FAILED        |
|56_105       |56      |105       |4             |FAILED        |
|56_101       |56      |101       |3             |FAILED        |
|66_103       |66      |103       |5             |FAILED        |
|66_101       |66      |101       |3             |FAILED        |
+-------------+--------+----------+--------------+--------------+
only showing top 20 rows
```

6. Hive: Staging & Queries

We created an External Hive Table over the Parquet data in HDFS. This allowed us to run SQL queries for insights (e.g., count orders by payment status, top products, total sales per order).

7. Final Pipeline Summary

1. Python → Generated sample JSON order files.
2. NiFi → Ingested JSON files, added ingestion timestamp, moved to output directory.
3. HDFS → Stored raw JSON files + transformed Parquet files.
4. Spark → Flattened and transformed JSON data.
5. Hive → Enabled SQL-based analytics on structured order data.