Data Storage & Pipelines

Assignment 2

Real-Time Food Order Streaming to Data Lake

Building real-time streaming pipeline for Food Delivery platform

ROLL NO: 2025EM1100102 | NAME: SHAIK KHAJA NAYAB RASOOL

1. Overall Goal: Real-Time Orders → Data Lake

Whenever new food order is inserted into PostgreSQL.

- Producer service detects using change-data-capture (CDC)-style polling mechanism based on created_at.
- Publishes the received order details, as a JSON message to a Kafka topic.
- Consumer consumes order details by a Spark Structured Streaming job.
- Cleaned and stored order details get captured as Parquet files in a Data Lake which either can be Local path or S3 bucket, partitioned by date, ready for analytics.

2. PostgreSQL Input Table

Table schema: *food_delivery_db.orders*

Columns: orders includes *order_id, customer_name, restaurant_name, item, amount, order_status, and created_at*.

Role in pipeline.

- Acts as single source, system of record for all food orders.
- The created_at column is used as a high-watermark to detect new rows.
- Every new row inserted must eventually appear in Data Lake via Kafka + Spark, ensuring end-to-end propagation of fresh data.

Field	Data Type	Validation Criteria	Rule #
order_id	Long	Not Null	1
customer_name	String	Not Null + Not Empty (after trim)	3
restaurant_name	String	Not Null + Not Empty (after trim)	4
item	String	Not Null + Not Empty (after trim)	5
amount	Double	Not Null + Non-Negative (> 0)	2, 7
order_status	String	Not Null + Not Empty (after trim)	6
created_at	Timestamp	(No validation - used for date partition)	—

order_id	customer_name	restaurant_name	item	amount	order_status	created_at
388	Test Customer 3	Test Restaurant 3	Test Item 3	417.78	CANCELLED	2025-12-07 07:45:41.637356
387	Test Customer 2	Test Restaurant 2	Test Item 2	62.08	PLACED	2025-12-07 07:45:41.637356
386	Test Customer 1	Test Restaurant 1	Test Item 1	371.54	DELIVERED	2025-12-07 07:45:41.637356
385	Test Customer 2	Test Restaurant 2	Test Item 2	123.09	PREPARING	2025-12-07 07:39:14.108942
384	Test Customer 1	Test Restaurant 1	Test Item 1	231.14	PREPARING	2025-12-07 07:39:14.108942

order_id	customer_name	restaurant_name	item	amount	order_status	created_at
399	Test Customer 1	Test Restaurant 1	Test Item 1	51.24	CANCELLED	2025-12-07 16:25:06.953853
398	Test Customer 1	Test Restaurant 1	Test Item 1	0.00	PLACED	2025-12-07 16:25:02.70675
397	Test Customer 1	Test Restaurant 1		123.77	PLACED	2025-12-07 16:24:52.495215
396	Test Customer 1		Test Item 1	175.00	DELIVERED	2025-12-07 16:24:41.675195
395		Test Restaurant 1	Test Item 1	190.97	PLACED	2025-12-07 16:24:24.39243
394	Test Customer 1	Test Restaurant 1	Test Item 1	152.47	PLACED	2025-12-07 16:23:22.536728
393	Test Customer 1	Test Restaurant 1	Test Item 1	-4.00	DELIVERED	2025-12-07 16:23:11.864464

3. Architecture & Data Flow

3.1 PostgreSQL → Spark CDC Poller (Producer Logic)

Mechanism:

- A Spark (PySpark) job periodically polls PostgreSQL every few seconds.
- Selects only rows with created_at > last_processed_timestamp, ensuring incremental ingestion.
- Newly fetched rows are converted into JSON documents matching specified schema.

Outcome:

- Reliable CDC-style stream of order events from a simple relational table, without using heavy CDC tools.

3.2 Kafka Topic: <rollnumber>_food_orders_raw

Role:

- Functions as a durable, scalable message queue.
- Receives each new order as a JSON message from producer.

Outcome:

- Decouples database from consumer.
- Enables real-time streaming and replay capability if needed.

3.3 Spark Structured Streaming Consumer

Mechanism:

- Spark Structured Streaming job reads from the Kafka topic continuously.
- Parses JSON messages into a strongly typed DataFrame with correct schema.
- Applies data cleaning rules.

- Drops records with null order_id.
- Drops records with negative amount.

Outcome:

- Transforms raw events into clean, analytics-ready records in streaming fashion.

3.4 Data Lake (Parquet, Partitioned by Date)

Storage pattern:

- Writes out as Parquet files to a configured datalake path (Local or S3).
- Partitions by date=YYYY-MM-DD, derived from created_at.
- Uses append mode so new batches are added without overwriting old data.
- Uses checkpointing to track streaming progress and offsets.

Outcome:

- Creates a query-efficient, columnar, partitioned store of all orders for downstream BI, reporting or ad-hoc analysis.
- Partitioning by date enables fast filtering and time-based analysis.

3.5 Key Architectural Properties

- Incremental ingestion, ensures only new rows are pulled from PostgreSQL.

(created_at > 'last_timestamp') OR (created_at = 'last_timestamp' AND order_id > last_order_id)

- Continuous streaming: Spark Structured Streaming handles ongoing Kafka consumption and state via checkpoints.
- Durable analytics store: Parquet + partitioning in a Data Lake yields efficient, scalable storage for historical and real-time data.
- Config-driven: All operational parameters (DB connection, Kafka, paths, batch interval) are centralized in *orders_stream.yml* for consistency and portability.

4. Part 1 — PostgreSQL Setup (db/orders.sql)

What you do:

- Define orders table schema exactly as specified.
- Insert at least 10 initial sample records.
- Ensure created_at is set correctly (e.g., using NOW()).

How it supports the pipeline:

- Establishes baseline dataset and table structure required by producer queries.
- Initial records are used to verify correct table creation and basic read connectivity.
- Later, insert additional records to simulate new real-time orders and validate incremental behaviour.

5. Part 2 — CDC Simulation & Kafka Producer (producers/orders_cdc_producer.py)

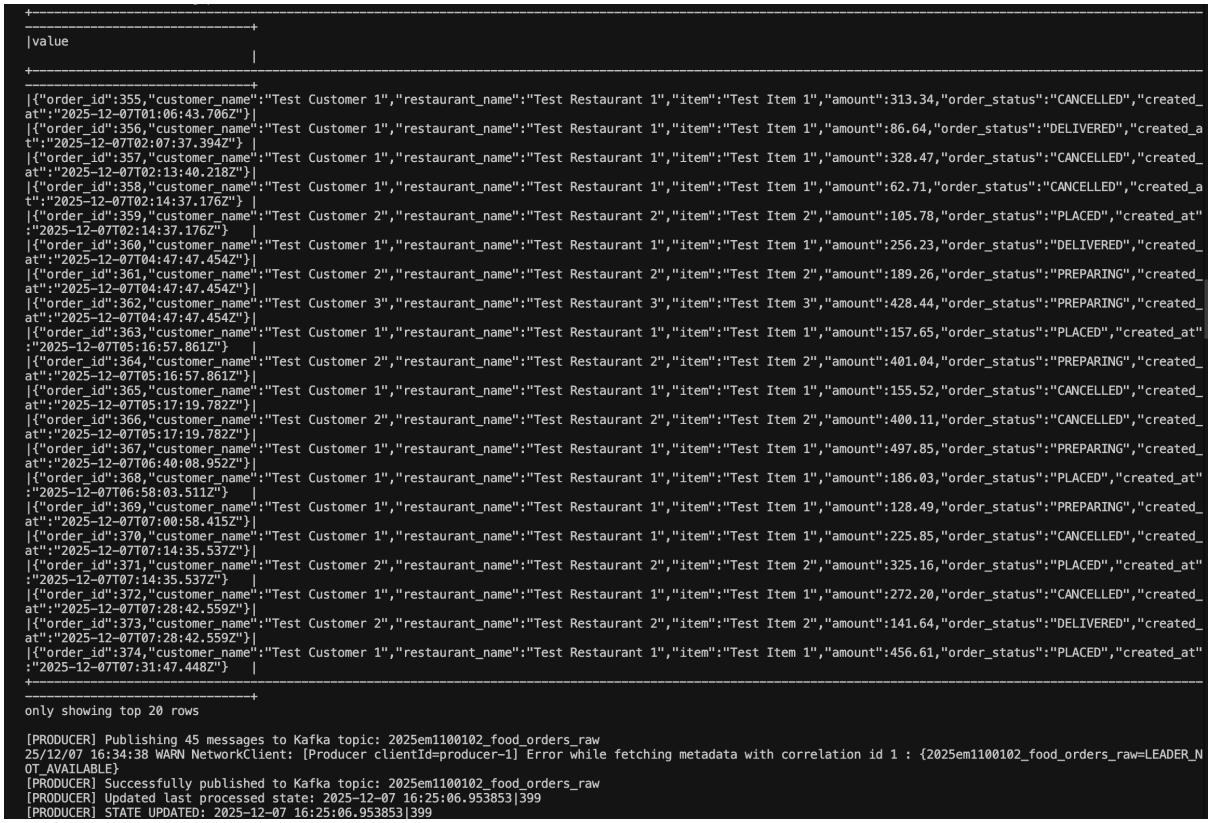
Core responsibilities

Connect to PostgreSQL using JDBC parameters from orders_stream.yml.

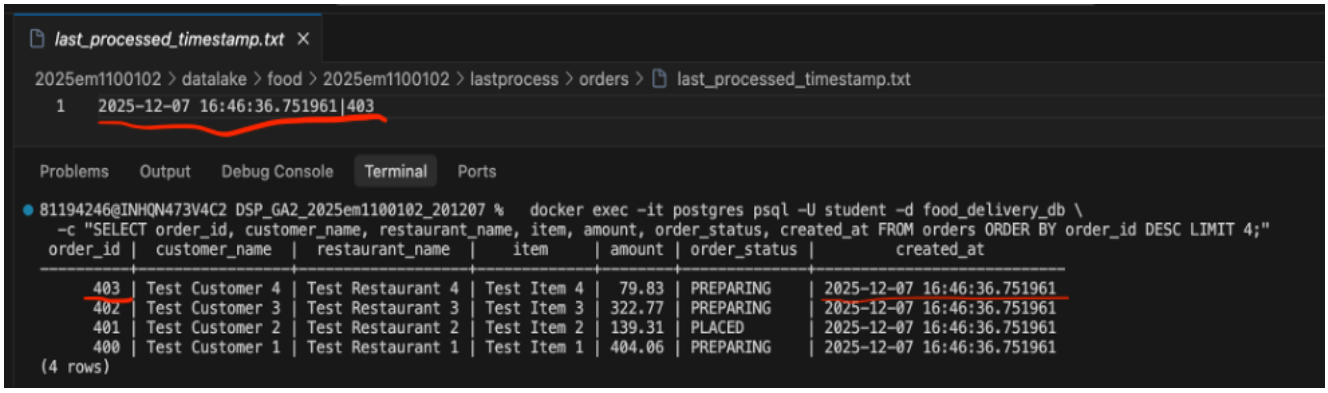
- Poll every N seconds (e.g., 5s) to find new rows:
- Query uses created_at > last_processed_timestamp.
- last_processed_timestamp is maintained in a separate file (specified in config) so state persists across runs.
- Transform records to JSON matching the required schema keys and types.
- Publish each JSON event to Kafka topic <rollnumber>_food_orders_raw.
- Update and store last_processed_timestamp after processing, preventing re-sending of old records.

How outcome is achieved

- The producer script acts as a CDC simulator, converting batch DB inserts into a near real-time event stream.
- File-based tracking of last_processed_timestamp ensures strict exactly-once-like semantics at the source side (no duplicates from re-reading the same created_at range).
- Publishing to a standardized topic name allows the consumer to reliably subscribe and process all new orders.



Last Processed Date-time



6. Part 3 — Kafka → Spark Structured Streaming Consumer (consumers/orders_stream_consumer.py)

Core responsibilities

- Subscribe to Kafka topic <rollnumber>_food_orders_raw using config values.
- Read streaming data as key-value pairs, parse JSON value into DataFrame with exact schema.
- Apply data quality rules.
    - Drop rows where order_id is null.
    - Drop rows where amount is negative.

Write to Data Lake

- Output format: Parquet.
- Path: datalake.path from config (local or S3).
- Partition by date derived from created_at (e.g., date=YYYY-MM-DD).
- Use append mode to keep adding new data.
- Configure checkpointing:
    - Use streaming.checkpoint_location to store offsets and state.
    - Ensures that, on restart, Spark resumes from the correct Kafka offsets without data loss or duplication.

How the outcome is achieved

- Spark Structured Streaming runs as a long-lived streaming application that.
    - Continuously pulls messages from Kafka.
    - Keeps track of progress via checkpoints.
    - Persists a clean, partitioned Parquet dataset to the Data Lake that mirrors the evolving orders table in (near) real time.

order_id	customer_name	restaurant_name	item	amount	order_status	created_at	validation_status
348		Test Restaurant 1	Test Item 1	224.71	PLACED	44:37.8	FAILED
349	Test Customer 1		Test Item 1	174.86	CANCELLED	44:45.2	FAILED
350	Test Customer 1	Test Restaurant 1		476.16	PREPARING	44:51.6	FAILED
351	Test Customer 1	Test Restaurant 1	Test Item 1	0	DELIVERED	45:00.8	FAILED
352	Test Customer 1	Test Restaurant 1	Test Item 1	-4	PREPARING	45:08.3	FAILED

order_id	customer_name	restaurant_name	item	amount	order_status	created_at	validation_status	date
353	Test Customer 1	Test Restaurant 1	Test Item 1	246.92	PREPARING	45:20.2	SUCCESS	06/12/25

```
+--------+---------------+-----------------+-----------+------+------------+----------------------+-----------------+
|order_id|customer_name  |restaurant_name  |item       |amount|order_status|created_at            |validation_status|
+--------+---------------+-----------------+-----------+------+------------+----------------------+-----------------+
|393     |Test Customer 1|Test Restaurant 1|Test Item 1|-4.0  |DELIVERED   |2025-12-07 16:23:11.864|FAILED           |
|395     |               |Test Restaurant 1|Test Item 1|190.97|PLACED      |2025-12-07 16:24:24.392|FAILED           |
|396     |Test Customer 1|                 |Test Item 1|175.0 |DELIVERED   |2025-12-07 16:24:41.675|FAILED           |
|397     |Test Customer 1|Test Restaurant 1|           |123.77|PLACED      |2025-12-07 16:24:52.495|FAILED           |
|398     |Test Customer 1|Test Restaurant 1|Test Item 1|0.0   |PLACED      |2025-12-07 16:25:02.706|FAILED           |
+--------+---------------+-----------------+-----------+------+------------+----------------------+-----------------+

+--------+---------------+-----------------+-----------+------+------------+----------------------+-----------------+----------+
|order_id|customer_name  |restaurant_name  |item       |amount|order_status|created_at            |validation_status|date      |
+--------+---------------+-----------------+-----------+------+------------+----------------------+-----------------+----------+
|355     |Test Customer 1|Test Restaurant 1|Test Item 1|313.34|CANCELLED   |2025-12-07 01:06:43.706|SUCCESS          |2025-12-07|
|356     |Test Customer 1|Test Restaurant 1|Test Item 1|86.64 |DELIVERED   |2025-12-07 02:07:37.394|SUCCESS          |2025-12-07|
|357     |Test Customer 1|Test Restaurant 1|Test Item 1|328.47|CANCELLED   |2025-12-07 02:13:40.218|SUCCESS          |2025-12-07|
|358     |Test Customer 1|Test Restaurant 1|Test Item 1|62.71 |CANCELLED   |2025-12-07 02:14:37.176|SUCCESS          |2025-12-07|
|359     |Test Customer 2|Test Restaurant 2|Test Item 2|105.78|PLACED      |2025-12-07 02:14:37.176|SUCCESS          |2025-12-07|
|360     |Test Customer 1|Test Restaurant 1|Test Item 1|256.23|DELIVERED   |2025-12-07 04:47:47.454|SUCCESS          |2025-12-07|
|361     |Test Customer 2|Test Restaurant 2|Test Item 2|189.26|PREPARING   |2025-12-07 04:47:47.454|SUCCESS          |2025-12-07|
|362     |Test Customer 3|Test Restaurant 3|Test Item 3|428.44|PREPARING   |2025-12-07 04:47:47.454|SUCCESS          |2025-12-07|
|363     |Test Customer 1|Test Restaurant 1|Test Item 1|157.65|PLACED      |2025-12-07 05:16:57.861|SUCCESS          |2025-12-07|
|364     |Test Customer 2|Test Restaurant 2|Test Item 2|401.04|PREPARING   |2025-12-07 05:16:57.861|SUCCESS          |2025-12-07|
|365     |Test Customer 1|Test Restaurant 1|Test Item 1|155.52|CANCELLED   |2025-12-07 05:17:19.782|SUCCESS          |2025-12-07|
|366     |Test Customer 2|Test Restaurant 2|Test Item 2|400.11|CANCELLED   |2025-12-07 05:17:19.782|SUCCESS          |2025-12-07|
|367     |Test Customer 1|Test Restaurant 1|Test Item 1|497.85|PREPARING   |2025-12-07 06:40:08.952|SUCCESS          |2025-12-07|
|368     |Test Customer 1|Test Restaurant 1|Test Item 1|186.03|PLACED      |2025-12-07 06:58:03.511|SUCCESS          |2025-12-07|
|369     |Test Customer 1|Test Restaurant 1|Test Item 1|128.49|PREPARING   |2025-12-07 07:00:58.415|SUCCESS          |2025-12-07|
|370     |Test Customer 1|Test Restaurant 1|Test Item 1|225.85|CANCELLED   |2025-12-07 07:14:35.537|SUCCESS          |2025-12-07|
|371     |Test Customer 2|Test Restaurant 2|Test Item 2|325.16|PLACED      |2025-12-07 07:14:35.537|SUCCESS          |2025-12-07|
|372     |Test Customer 1|Test Restaurant 1|Test Item 1|272.2 |CANCELLED   |2025-12-07 07:28:42.559|SUCCESS          |2025-12-07|
|373     |Test Customer 2|Test Restaurant 2|Test Item 2|141.64|DELIVERED   |2025-12-07 07:28:42.559|SUCCESS          |2025-12-07|
|374     |Test Customer 1|Test Restaurant 1|Test Item 1|456.61|PLACED      |2025-12-07 07:31:47.448|SUCCESS          |2025-12-07|
+--------+---------------+-----------------+-----------+------+------------+----------------------+-----------------+----------+
```

order_id	customer_name	restaurant_name	item	amount	order_status	created_at	validation_status
355	Test Customer 1	Test Restaurant 1	Test Item 1	313.34	CANCELLED	2025-12-07T01:06:43.706000	SUCCESS
356	Test Customer 1	Test Restaurant 1	Test Item 1	86.64	DELIVERED	2025-12-07T02:07:37.394000	SUCCESS
357	Test Customer 1	Test Restaurant 1	Test Item 1	328.47	CANCELLED	2025-12-07T02:13:40.218000	SUCCESS
358	Test Customer 1	Test Restaurant 1	Test Item 1	62.71	CANCELLED	2025-12-07T02:14:37.176000	SUCCESS
359	Test Customer 2	Test Restaurant 2	Test Item 2	105.78	PLACED	2025-12-07T02:14:37.176000	SUCCESS
360	Test Customer 1	Test Restaurant 1	Test Item 1	256.23	DELIVERED	2025-12-07T04:47:47.454000	SUCCESS
361	Test Customer 2	Test Restaurant 2	Test Item 2	189.26	PREPARING	2025-12-07T04:47:47.454000	SUCCESS
362	Test Customer 3	Test Restaurant 3	Test Item 3	428.44	PREPARING	2025-12-07T04:47:47.454000	SUCCESS
363	Test Customer 1	Test Restaurant 1	Test Item 1	157.65	PLACED	2025-12-07T05:16:57.861000	SUCCESS
364	Test Customer 2	Test Restaurant 2	Test Item 2	401.04	PREPARING	2025-12-07T05:16:57.861000	SUCCESS
365	Test Customer 1	Test Restaurant 1	Test Item 1	155.52	CANCELLED	2025-12-07T05:17:19.782000	SUCCESS
366	Test Customer 2	Test Restaurant 2	Test Item 2	400.11	CANCELLED	2025-12-07T05:17:19.782000	SUCCESS
367	Test Customer 1	Test Restaurant 1	Test Item 1	497.85	PREPARING	2025-12-07T06:40:08.952000	SUCCESS
368	Test Customer 1	Test Restaurant 1	Test Item 1	186.03	PLACED	2025-12-07T06:58:03.511000	SUCCESS
369	Test Customer 1	Test Restaurant 1	Test Item 1	128.49	PREPARING	2025-12-07T07:00:58.415000	SUCCESS
370	Test Customer 1	Test Restaurant 1	Test Item 1	225.85	CANCELLED	2025-12-07T07:14:35.537000	SUCCESS
371	Test Customer 2	Test Restaurant 2	Test Item 2	325.16	PLACED	2025-12-07T07:14:35.537000	SUCCESS
372	Test Customer 1	Test Restaurant 1	Test Item 1	272.2	CANCELLED	2025-12-07T07:28:42.559000	SUCCESS
373	Test Customer 2	Test Restaurant 2	Test Item 2	141.64	DELIVERED	2025-12-07T07:28:42.559000	SUCCESS
374	Test Customer 1	Test Restaurant 1	Test Item 1	456.61	PLACED	2025-12-07T07:31:47.448000	SUCCESS

7. Part 4 — Incremental Testing & Evaluation Pattern

Evaluation flow:

- Instructor inserts 5 new orders into PostgreSQL.

- Runs both producer and consumer scripts.

Verifies that:

- All 5 new records appear in the Data Lake.
- No missing or duplicate orders.
- Inserts another 5 incremental records and repeats.
- Continues multiple cycles to confirm robust incremental ingestion.

How correctness is validated:

- Source vs Lake counts are compared to ensure all new orders propagate.
- Timestamps and partitions are checked to ensure records land in the correct date folders.

Repeated add → process → verify cycles test:

- The polling logic (created_at > last_processed_timestamp).
- The state file for last processed timestamp.
- The Spark checkpoints and Kafka offset handling.
- Successful tests show the pipeline reliably processes only new data across multiple runs without duplicating older records.

8. Assignment Deliverables & Config-Driven Design

8.1 Submission Structure

Standardized folder layout:

- db/orders.sql
- producers/orders_cdc_producer.py
- consumers/orders_stream_consumer.py
- scripts/producer_spark_submit.sh, scripts/consumer_spark_submit.sh
- configs/orders_stream.yml
- README.md

Outcome:

- Ensures that instructors can run and evaluate your project consistently with minimal setup.

8.2 Config File:

- configs/orders_stream.yml
- Centralizes all critical parameters:
- Postgres: jdbc_url, host, port, db (or database), user, password, table.
- Kafka: brokers/bootstrap_servers, topic.

- Datalake: path, format.

Streaming:

- checkpoint_location
- last_processed_timestamp_location
- batch_interval (poll frequency in seconds)

How it helps:

- Decouples code from environment-specific settings.
- Enables easy switching between local vs S3 or different DB/Kafka endpoints.
- Guarantees all students follow a uniform configuration interface, simplifying evaluation.

8.3 Spark Submit Scripts

Sample commands:

- producer_spark_submit.sh runs the CDC producer with required Kafka package, using the config file.
- consumer_spark_submit.sh runs the streaming consumer similarly.

Outcome:

- Provides a one-command entry point for running each component.
- Ensures the correct Spark packages and config paths are used every time, reducing runtime errors.

9. How Everything Together Achieves Goal

- PostgreSQL is the authoritative source of orders.
- CDC Producer (PySpark) converts new rows into event streams using created_at + last_processed_timestamp.
- Kafka buffers and distributes those events reliably in real time.
- Spark Structured Streaming Consumer transforms and cleans them continuously.
- Data Lake (Parquet, partitioned) accumulates all clean orders in an analysis-friendly, scalable format.

10. Artifacts

2025em1100102/food_delivery_streaming/local/

├── db/

│ └── orders.sql

├── producers/

```
│   └── orders_cdc_producer.py
├── consumers/
│   └── orders_stream_consumer.py
├── scripts/
│   ├── producer_spark_submit.sh
│   └── consumer_spark_submit.sh
├── configs/
│   └── orders_stream.yml
└── README.md
```