Data Pipeline Project Report*(Airflow + Kafka + SQLite)*

1.Team member: Dosmaganbetkyzy Ayaulym, Podshai Dinara, Rashid Dana

2. API Justification

For this project, the **Kraken Public API** was selected as the data source.

API used: Endpoint: https://api.kraken.com/0/public/Ticker

Justification:

The Kraken API provides **real-time cryptocurrency market data**, which is well-suited for streaming and analytics tasks.

The data is **frequently updated**, satisfying the requirement for pseudo-streaming ingestion.

The API is **public and free**, requiring no authentication.

The response is returned in **JSON format**, which is ideal for Kafka-based pipelines.

The API contains both **raw values and aggregated metrics** (prices, volumes, highs/lows), making it suitable for downstream analytics.

3. Architecture Overview

The system is built as a **three-stage data pipeline** orchestrated using Apache Airflow and Apache Kafka.

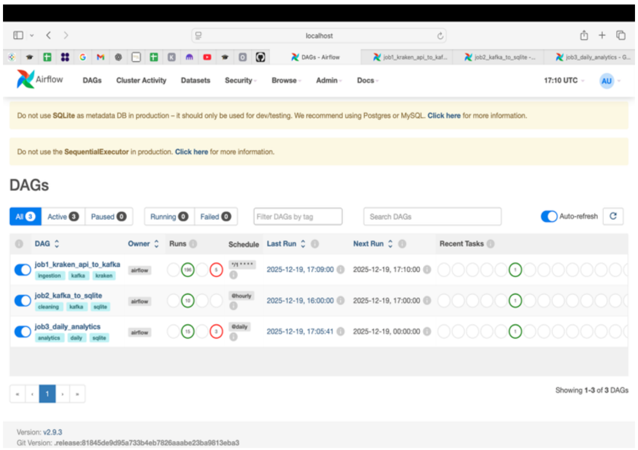

Figure 1. Airflow DAGs overview showing ingestion, cleaning, and analytics pipelines.

3.1 DAG 1 – Continuous Ingestion Job (Pseudo Streaming)

data from the Kraken API every 1 minute (fits the “30 seconds–few minutes” requirement)

Sends raw JSON responses to a Kafka topic called raw_events

Simulates continuous data ingestion (pseudo-streaming)

Figure 2. DAG1 responsible for continuous ingestion from Kraken API to Kafka.

Flow:
Kraken API → DAG 1 → Kafka (raw_events)

3.2 DAG 2 – Hourly Cleaning + Storage Job (Batch)

Scheduled to run **hourly**

Reads new messages from the Kafka topic

Applies data cleaning and normalization

Writes cleaned data into an SQLite database table events

Flow:
Kafka → DAG 2 → Cleaning → SQLite (events)

3.3 DAG 3 – Daily Analytics Job (Batch)

Scheduled to run **daily**

Reads cleaned data from SQLite

Computes aggregated analytics

Writes results into SQLite table daily_summary

Flow:
SQLite (events) → DAG 3 → Analytics → SQLite

4. Kafka Topic Schema (Topic name: raw_events)

The Kafka topic raw_events stores raw JSON messages fetched from the Kraken API.
Each message contains the trading pair identifier (pair), ingestion timestamp, and the full unprocessed API response (raw_payload).
This topic serves as the input for downstream batch processing and data cleaning.

Figure 3. Kafka topic `raw_events` containing raw JSON messages from the Kraken API.

Schema (JSON):

{

"ingested_at": "ISO-8601 timestamp",

"pair": "XBTUSD",

"raw_payload": {

"error": [],

"result": {

"XXBTZUSD": {

"a": ["ask_price", "..."],

```
        "b": ["bid_price", "..."],
        "c": ["last_price", "..."],
        "v": ["volume_today", "volume_24h"],
        "p": ["vwap_today", "vwap_24h"],
        "t": ["trades_today", "trades_24h"],
        "l": ["low_today", "low_24h"],
        "h": ["high_today", "high_24h"],
        "o": "open_price"
      }
    }
  }
}
```


5. Data Cleaning Rules (DAG 2)

The following cleaning and validation rules are applied:

Conversion of numeric fields from strings to float / int

Handling of missing or invalid values (safe casting)

Filtering out records with missing critical fields (e.g. last price)

Normalization of timestamps to ISO-8601 format

Preservation of the original raw JSON for traceability

6. SQLite Schema (Table: events (Cleaned Data) Stores cleaned and normalized event-level data.)

This screenshot shows the SQLite events table after the cleaning stage. The data contains normalized numeric values and verified records produced by DAG 2

Column	Description
Id	Auto increment primary key
Ingested at	Timestamp of ingestion
Pair	Trading pair
Kraken_symbol	Kraken internal symbol
Ask_price	Ask price
Bid_price	Bid price
Last_price	Last traded price
Volume_today	Volume(today)
Volume_24h	Volume (24h)
Vwap_today	VWAP (today)
Vwap_24h	VWAP(24h)
Trades_today	Trades count (today)
Trades_24h	Trades count(24h)
Low_today	Lowest price (today)
Low_24h	Lowest price(24h)
High_today	Highest price (today)
High_24h	Highest price (24h)
Open_price	Opening price
Raw_json	Original raw JSON

6.2 Table: daily_summary (Aggregated Analytics)

Stores daily aggregated metrics.

Column	Description
day	Aggregation date
Pair	Trading pair
Count_events	Number of records
Avg_last_price	Average last price
Min_last_price	Min last price
Max_last_price	Max last price
Avg_spread	Average bid ask spread

```
airflow@b82509167962:/opt/airflow$ python - <<'PY'
import sqlite3

conn = sqlite3.connect('/opt/airflow/data/events.db')
cur = conn.cursor()

cur.execute('select count(*) from events')
print('events_count =', cur.fetchone()[0])

cur.execute('select ingested_at, pair, last_price from events order by id desc limit 5')
for row in cur.fetchall():
    print(row)

conn.close()
PY
events_count = 242
('2025-12-19T17:59:05.008623+00:00', 'XBTUSD', 87069.0)
('2025-12-19T17:58:03.447994+00:00', 'XBTUSD', 87304.8)
('2025-12-19T17:57:04.017285+00:00', 'XBTUSD', 87302.9)
('2025-12-19T17:56:03.286710+00:00', 'XBTUSD', 87200.2)
('2025-12-19T17:55:03.693174+00:00', 'XBTUSD', 87189.5)
airflow@b82509167962:/opt/airflow$
```

7. Conclusion

This project successfully implements a full data pipeline using:

Apache Airflow for orchestration

Apache Kafka for streaming ingestion

SQLite for storage and analytics

All project requirements are met, including continuous ingestion, batch cleaning, analytics, and persistent storage.