### **Report: Space Weather Data Pipeline (Airflow, Kafka, SQLite)**

**Date**: December 19, 2025

**Members**: Amirbekuly Abay 22B22B1520

Kalikyzy Daniya 22B22B1527

Manatkyzy Zhanel 22B22B1528

#### **1. API**

The **NOAA Space Weather API** is used for fetching solar flare data, specifically focusing on **solar X-ray flux**. The API provides real-time updates on solar flare intensities, which are crucial for understanding space weather phenomena.

**API Endpoint**:  
https://services.swpc.noaa.gov/json/goes/primary/xrays-1-day.json

**Data Structure**: The response from the API includes the following fields:

* **time\_tag**: Date and time of measurement (UTC)
* **flux**: Intensity of solar flare
* **satellite**: The satellite collecting the data (e.g., GOES-16)

**Why Chosen**:

* The data updates every minute, making it perfect for real-time and batch processing.
* It provides valuable insights into solar flare activity, which is essential for space weather studies.

#### **2. System Architecture**

The pipeline consists of three main parts:

* **Ingestion Layer** – Collects data from the NOAA API and streams it to Kafka.
* **Message Queue Layer** – Kafka stores raw data temporarily before processing.
* **ETL / Storage Layer** – Cleans the data and stores it in SQLite for further analytics.

| **Component** | **Technology** | **Role** |
| --- | --- | --- |
| **Orchestration** | Apache Airflow | Runs and monitors DAGs |
| **Data Source** | NOAA Space Weather API | Provides real-time solar flare data |
| **Data Buffer** | Apache Kafka | Stores raw data temporarily |
| **Database** | SQLite | Stores cleaned and aggregated data |

#### **3. Airflow DAGs**

The pipeline includes three primary DAGs to process the data:

1. **DAG 1: Continuous ingestion → Kafka**
   * **Goal**: Fetch data from the NOAA API every 1-5 minutes and send it to Kafka.
   * **Flow**: NOAA API → DAG1 → Kafka (solar\_flare\_raw)

**Code (job1\_producer.py)**:  
import requests

import json

from kafka import KafkaProducer

from datetime import datetime

producer = KafkaProducer(

bootstrap\_servers='localhost:9092',

value\_serializer=lambda v: json.dumps(v).encode('utf-8')

)

URL = "https://services.swpc.noaa.gov/json/goes/primary/xrays-1-day.json"

def fetch\_solar\_data():

r = requests.get(URL).json()

for item in r:

msg = {

"time\_tag": item["time\_tag"],

"flux": float(item["flux"]),

"satellite": item["satellite"]

}

producer.send("solar\_flare\_raw", msg)

producer.flush()

1. **DAG 2: Batch cleaning → SQLite (@hourly)**
   * **Goal**: Read new data from Kafka, clean it, and save it to SQLite.
   * **Flow**: Kafka (solar\_flare\_raw) → DAG2 → SQLite (solar\_events)

**Code (job2\_cleaner.py)**:  
import pandas as pd

import sqlite3

# Example Kafka message reading

df = pd.read\_json('kafka\_messages.json')

# Cleaning

df.dropna(inplace=True)

df['flux'] = df['flux'].astype(float)

# Save to SQLite

conn = sqlite3.connect("solar\_data.db")

df.to\_sql("solar\_events", conn, if\_exists="append", index=False)

conn.close()

1. **DAG 3: Daily Analytics (@daily)**
   * **Goal**: Calculate daily metrics like average, max, and min flux, and save to daily\_summary table.
   * **Flow**: SQLite (solar\_events) → DAG3 → SQLite (daily\_summary)

**Code (job3\_analytics.py)**:  
import pandas as pd

import sqlite3

conn = sqlite3.connect("solar\_data.db")

df = pd.read\_sql("SELECT \* FROM solar\_events", conn)

df['date'] = pd.to\_datetime(df['time\_tag']).dt.date

summary = df.groupby('date').agg(

avg\_flux=('flux', 'mean'),

max\_flux=('flux', 'max'),

min\_flux=('flux', 'min'),

measurements\_count=('flux', 'count')

).reset\_index()

summary.to\_sql("daily\_summary", conn, if\_exists="append", index=False)

conn.close()

#### **4. Kafka Topic Schema**

The solar\_flare\_raw Kafka topic stores raw solar flare data.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| **time\_tag** | datetime | Time of the measurement (UTC) |
| **flux** | float | Solar flare intensity |
| **satellite** | string | Satellite source of the data |

Each Kafka message contains the flux and satellite data for each minute.

#### **5. Data Cleaning Rules**

The data is cleaned by:

* Removing any missing values.
* Ensuring the flux field is a valid float.
* Storing cleaned data in the SQLite database.

#### **6. Database Schema**

**solar\_events Table**:

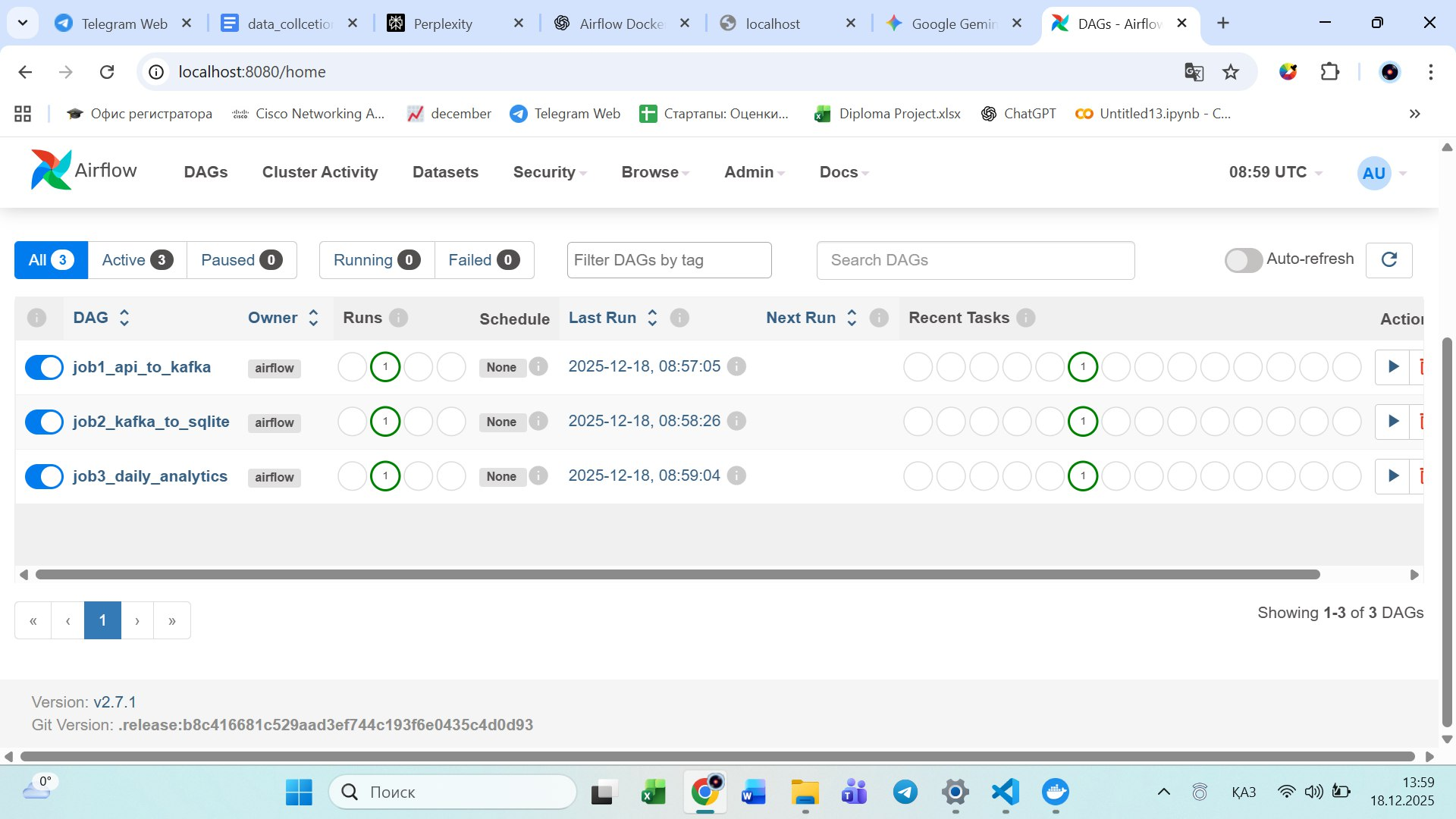
| **Column** | **Type** | **Description** |
| --- | --- | --- |
| **time\_tag** | TEXT | Time of data collection |
| **flux** | REAL | Solar flare intensity |
| **satellite** | TEXT | Satellite source |

**daily\_summary Table**:

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| **date** | TEXT | Date of the summary |
| **avg\_flux** | REAL | Average flux for the day |
| **max\_flux** | REAL | Maximum flux for the day |
| **min\_flux** | REAL | Minimum flux for the day |
| **measurements\_count** | INTEGER | Number of measurements |

#### **7. System Status**

The system components, including **Airflow**, **Kafka**, and **SQLite**, are working correctly, and the Airflow web interface is available at: http://localhost:8082.



#### **8. How to Run the System**

To start the pipeline:

1. Open the Airflow web interface.
2. Enable the following DAGs:
   * job1\_ingestion
   * job2\_clean\_store
   * job3\_daily\_summary
3. The system will start collecting, cleaning, and analyzing solar flare data.

### **Conclusion**

This pipeline effectively demonstrates the combination of **streaming** and **batch** data processing for real-time solar flare analysis. It uses **Apache Kafka** for streaming data, **Apache Airflow** for orchestration, and **SQLite** for storage, making it a comprehensive and functional solution.

