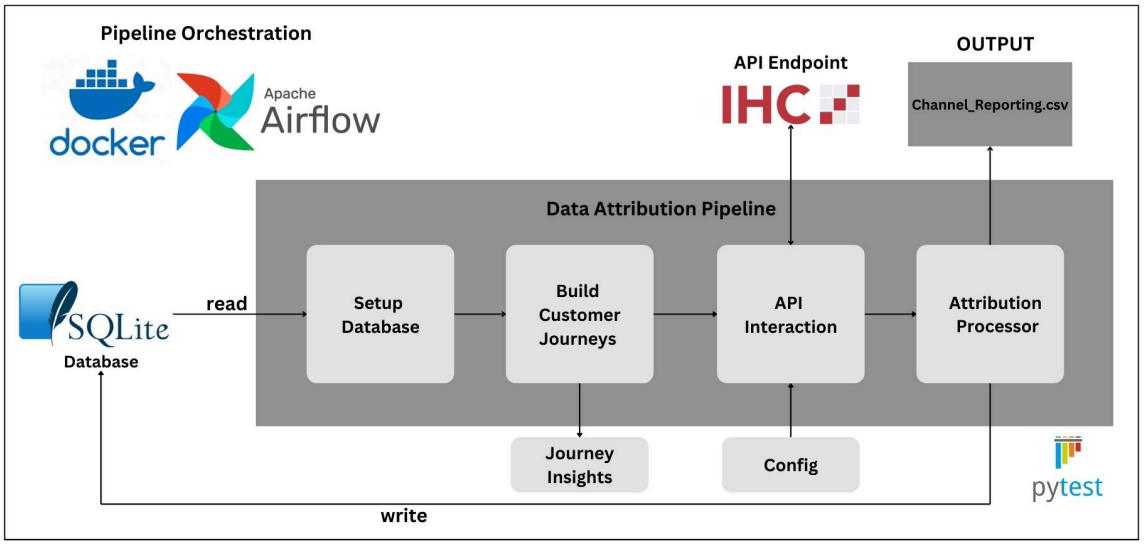
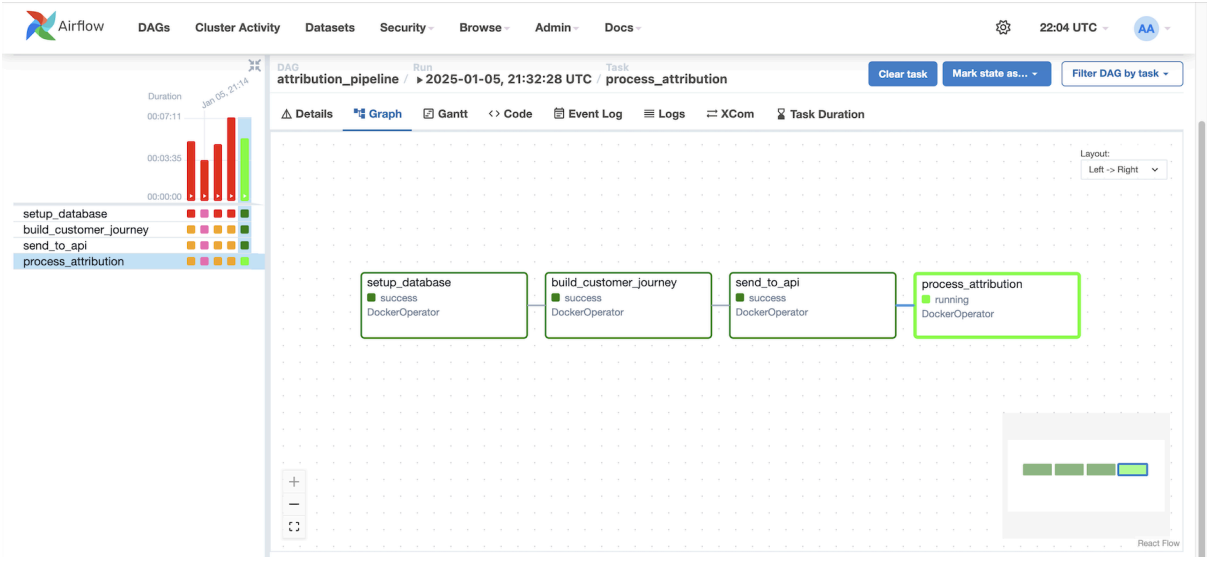
Attribution Pipeline Design Report

Pipeline Design

The pipeline is structured to process customer journey data, attribute conversions to various marketing channels, generate actionable reporting, **and accept a time range as input.** The diagram below shows the overall architecture of the system.



1. **Database Setup:**
    - **Script:** `setup_db.py`
    - **Purpose:**
        i. Initializes the SQLite database.
        ii. Creates required tables.
        iii. Validates database schema.
2. **Data Preparation:**
    - **Script:** `build_customer_journey.py`
    - **Purpose:**
        i. Constructs customer journeys by joining session_sources and conversions.
        ii. Provides insights on the Customer Journeys
3. **API Interaction:**
    - **Script:** `send_to_api.py`
    - **Purpose:**
        i. Formats journey data for API.
        ii. Handles API communication.
        iii. Implements retry logic.
        iv. Processes data in batches.
        v.

4. **Attribution Processing and Reporting:**
    - **Script:** `attribution_processor.py`
    - **Purpose:**
        i. Processes API responses.
        ii. Updates attribution results by populating attribution_customer_journey and channel_reporting tables.
        iii. Calculates key metrics (CPO, ROAS) and dumps it into a csv file.
5. **Config and Test Cases:**
    - API configurations like batch_size, API Key, etc. can be configured in the config.yaml file inside the config dir.
    - The test cases can be found in the tests dir. See Readme on how to run them.
6. **Orchestration:**
    - **Dockerization:** Ensures consistent deployment across environments.
    - **Makefile**: No need to remember or enter long docker commands. Also processes start date and end date as input to the pipeline. See Readme for instructions on how to use
    - **Airflow DAGs:** Automates and schedules the execution of pipeline steps, enabling modularity and traceability. As seen in the screenshot below
7. 

Assumptions

1. The example dataset provided for parameter training to create a new conversion type for the IHC Attribution model is good enough for the training.
2. Assumes the redistribution parameters are static and predetermined.
3. All of the time stamps are in the same time zone.
4. The time range provided as input to the pipeline accepts the date and not the time stamp.

Potential Improvements

1. Add robust data validation layers to identify and handle missing, duplicate, or malformed data at each step of the pipeline. Right now it's been done only while building the customer journeys
2. Use Pyspark for faster data processing.
3. Parallelize API requests to improve processing speed for larger datasets.
4. Expand test cases to cover edge cases like empty datasets, invalid API responses, and partial database updates.