Deliverable 3

Creating a Kafka topic

First, we need to create a Kafka topic for the sensors, you can use the following command inside the Kafka container:

```
kafka-topics --create --topic transactions --partitions 1
↪  --replication-factor 1 --bootstrap-server localhost:9092
```

Then, we need to connect the topic to HDFS. To do that, we need to create a connector. The connector is a configuration file that tells Kafka how to connect to HDFS.

```
curl -X POST -H "Content-Type: application/json" --data '{
  "name": "hdfs-sink-connector",
  "config": {
      "connector.class":
↪  "io.confluent.connect.hdfs.HdfsSinkConnector",
      "tasks.max": "1",
      "topics": "sensores",
      "hdfs.url": "hdfs://namenode:9000",
      "flush.size": "10",
      "hdfs.authentication.kerberos": "false",
      "format.class":
↪  "io.confluent.connect.hdfs.json.JsonFormat",
      "partitioner.class":
↪  "io.confluent.connect.storage.partitioner.DefaultPartitioner",
      "rotate.interval.ms": "60000",
      "locale": "en",
      "timezone": "UTC",
      "value.converter.schemas.enable": "false"
  }
}' http://localhost:8083/connectors
```

Consuming data from CSV file and sending it to kafka

The code available in `src/dag_deliverable.py` starts by sending the data to the Kafka topic. It reads the CSV file and sends it to the Kafka topic using the ProduceToTopicOperator dag provider by returning json object representations of each of the rows in the CSV.

Sending it will automatically trigger the HDFS connector to save the data in HDFS.

Creating a table in Hive

To create a table in Hive, we first create a database and then an external table via the following SQL statements via the HiveServer2Hook dag provider, which provides us with a cursor we can use to execute things in Hive.

Note that the created table is external, pointing to a directory in HDFS, so that the data is not copied to Hive, but rather it is read from HDFS. It is also stated that the data must be specified to be in text files and in JSON format, so that Hive knows how to correctly handle the data.

```
> CREATE DATABASE IF NOT EXISTS weather
> CREATE EXTERNAL TABLE IF NOT EXISTS {HIVE_TABLE} (
    record_timestamp DATE,
    temperature_salon FLOAT,
    humidity_salon FLOAT,
    air_salon FLOAT,
    temperature_chambre FLOAT,
    humidity_chambre FLOAT,
    air_chambre FLOAT,
    temperature_bureau FLOAT,
    humidity_bureau FLOAT,
    air_bureau FLOAT,
    temperature_exterieur FLOAT,
    humidity_exterieur FLOAT,
    air_exterieur FLOAT
  )
  ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
  STORED AS TEXTFILE
  LOCATION 'hdfs://namenode:9000{HDFS_DIR}'
```

Querying the data

To query the data we connect to Hive the same way as before, then use the cursor to send the query.

For the first query, we must obtain the average temperatures of each room per day.

```
> SELECT
    DATE(record_timestamp) AS record_date,
    AVG(temperature_salon) AS avg_temp_salon,
    AVG(temperature_chambre) AS avg_temp_chambre,
    AVG(temperature_bureau) AS avg_temp_bureau,
    AVG(temperature_exterieur) AS avg_temp_exterieur
  FROM {HIVE_TABLE}
  GROUP BY DATE(record_timestamp)
  ORDER BY record_date
```

For the second we must obtain the moments with the worst air quaility. We assumed as no information was given, that the higher the `air` value was, the worse the air quality.

```
> SELECT
    record_timestamp,
    air_salon,
    air_chambre,
    air_bureau,
    air_exterieur,
  GREATEST(air_salon, air_chambre, air_bureau, air_exterieur) AS
  ↪  max_air_quality
  FROM {HIVE_TABLE}
  ORDER BY max_air_quality DESC
```

Finally, for the third, we must obtain the instances where humidity has changed more than 10% in an hour, for which we use tha `LAG` SQl function.

```
> SELECT
    record_timestamp,
    (1 - (humidity_salon/previous_humidity_salon)) AS
↪   change_salon,
    (1 - (humidity_chambre/previous_humidity_chambre)) AS
↪   change_chambre,
    (1 - (humidity_bureau/previous_humidity_bureau)) AS
↪   change_bureau,
    (1 - (humidity_exterieur/previous_humidity_exterieur)) AS
↪   change_exterieur
  FROM (
    SELECT record_timestamp, humidity_salon, humidity_chambre,
    ↪  humidity_bureau, humidity_exterieur,
      LAG(humidity_salon, 4, humidity_salon) OVER (ORDER BY
      ↪  record_timestamp) AS previous_humidity_salon,
      LAG(humidity_chambre, 4, humidity_chambre) OVER (ORDER BY
      ↪  record_timestamp) AS previous_humidity_chambre,
      LAG(humidity_bureau, 4, humidity_bureau) OVER (ORDER BY
      ↪  record_timestamp) AS previous_humidity_bureau,
      LAG(humidity_exterieur, 4, humidity_exterieur) OVER (ORDER
      ↪  BY record_timestamp) AS previous_humidity_exterieur
    FROM {HIVE_TABLE}
  ) AS t
  WHERE ABS(1 - (humidity_salon/previous_humidity_salon)) > 0.1
    OR ABS(1 - (humidity_chambre/previous_humidity_chambre)) >
    ↪  0.1
    OR ABS(1 - (humidity_bureau/previous_humidity_bureau)) > 0.1
    OR ABS(1 - (humidity_exterieur/previous_humidity_exterieur))
    ↪  > 0.1
```

Evidences

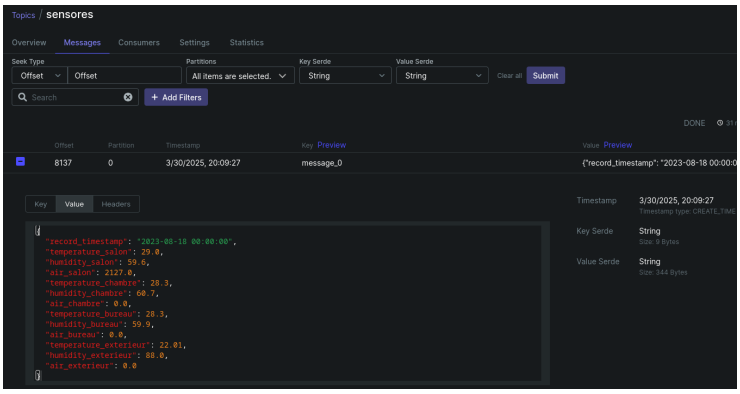

Figure 1: Data sent to kafka topic

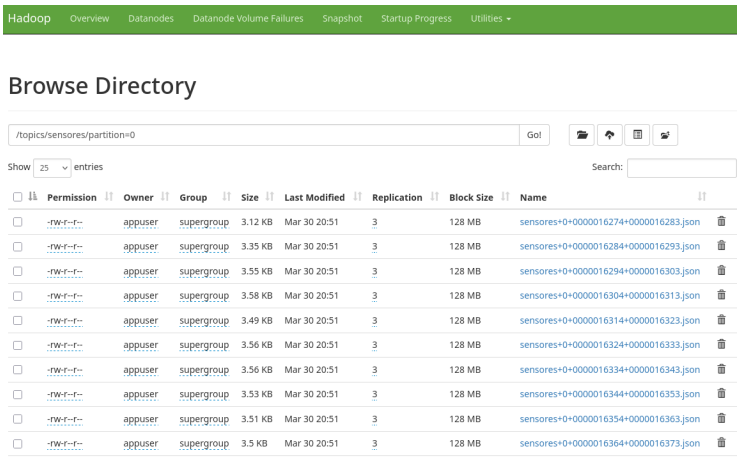

Figure 2: Data in HDFS

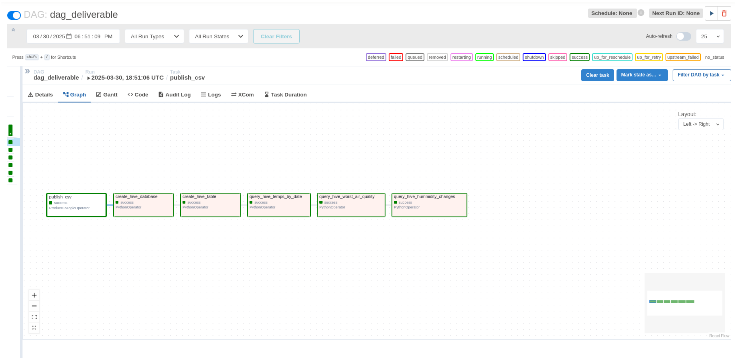

Figure 3: Executed DAG

```
airflow
*** Found local files:
***   * /opt/airflow/logs/dag_id=dag_deliverable/run_id=manual__2025-03-30T18:51:06.652527+00:00/task_id=query_hive_temps_by_date/attempt=1.log
[2025-03-30, 18:51:21 UTC] {local_task_job_runner.py:120} ▶ Pre task execution logs
[2025-03-30, 18:51:21 UTC] {base.py:84} INFO - Using connection ID 'hive_default' for task execution.
[2025-03-30, 18:51:21 UTC] {hive.py:475} INFO - USE `default`
[2025-03-30, 18:51:21 UTC] {hive.py:475} INFO - SELECT
            DATE(record_timestamp) AS record_date,
            AVG(temperature_salon) AS avg_temp_salon,
            AVG(temperature_chambre) AS avg_temp_chambre,
            AVG(temperature_bureau) AS avg_temp_bureau,
            AVG(temperature_exterieur) AS avg_temp_exterieur
            FROM weather.sensor_data
            GROUP BY DATE(record_timestamp)
            ORDER BY record_date

[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:158} INFO - Resultados de Hive:
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:159} INFO - Fecha | Temp. Salón | Temp. Chambre | Temp. Bureau | Temp. Exterieur
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-18', 26.229101558526356, 28.598307371139526, 26.73072910308838, 24.42216157913208)
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-19', 28.050390700499218, 29.84069013595581, 28.42786451180776, 25.36934902270635)
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-20', 28.243880212306976, 30.23046875, 28.78352866570155, 26.553378919760387)
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-21', 28.857747435569763, 31.425130267937977, 29.564453144868214, 26.901757776737213)
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-22', 29.128385444482166, 31.279622495174408, 29.874804655710857, 26.359472533067066)
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-23', 28.581835905710857, 30.892057240009308, 29.53346347808838, 25.215944012006123)
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-24', 28.492903689543407, 30.34453123807907, 29.536718785762787, 24.043489535649616)
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-25', 27.586328109105427, 29.72272139787674, 28.93925780057907, 22.69482409954071)
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-26', 27.202278693517048, 29.283984382947285, 28.522656222184498, 20.18541673819224)
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-27', 24.549544374148052, 27.476041615009308, 26.213346501191456, 17.748059888680775)
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-28', 24.95449225107829, 25.51738288005193, 25.62330734729767, 15.337604145208994)
[2025-03-30, 18:51:24 UTC] {dag_deliverable.py:163} INFO - ('2023-08-29', 24.112304627895355, 24.361328125, 24.803906281789143, 16.013854146003723)
```

Figure 4: Hive query 1

```
airflow
*** Found local files:
***   * /opt/airflow/logs/dag_id=dag_deliverable/run_id=manual__2025-03-30T18:51:06.652527+00:00/task_id=query_hive_worst_air_quality/attempt=1.log
[2025-03-30, 18:51:26 UTC] {local_task_job_runner.py:120} ▶ Pre task execution logs
[2025-03-30, 18:51:27 UTC] {base.py:84} INFO - Using connection ID 'hive_default' for task execution.
[2025-03-30, 18:51:27 UTC] {hive.py:475} INFO - USE `default`
[2025-03-30, 18:51:27 UTC] {hive.py:475} INFO - SELECT
            record_timestamp,
            air_salon,
            air_chambre,
            air_bureau,
            air_exterieur,
            GREATEST(air_salon, air_chambre, air_bureau, air_exterieur) AS max_air_quality
            FROM weather.sensor_data
            ORDER BY max_air_quality DESC

[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:188} INFO - Días con peor calidad de aire en cualquier parte de la casa:
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:189} INFO - Fecha | Calidad Salón | Calidad Chambre | Calidad Bureau | Calidad Exterieur
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-08-18', 2439.0, 0.0, 0.0, 0.0)
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-08-18', 2389.0, 0.0, 0.0, 0.0)
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-08-18', 2382.0, 0.0, 0.0, 0.0)
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-08-18', 2375.0, 0.0, 0.0, 0.0)
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-08-18', 2368.0, 0.0, 0.0, 0.0)
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-09-01', 2363.0, 0.0, 0.0, 0.0)
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-09-01', 2352.0, 0.0, 0.0, 0.0)
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-09-01', 2342.0, 0.0, 0.0, 0.0)
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-08-18', 2342.0, 0.0, 0.0, 0.0)
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-08-18', 2337.0, 0.0, 0.0, 0.0)
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-09-01', 2330.0, 0.0, 0.0, 0.0)
[2025-03-30, 18:51:28 UTC] {dag_deliverable.py:193} INFO - ('2023-09-01', 2320.0, 0.0, 0.0, 0.0)
```

Figure 5: Hive query 2

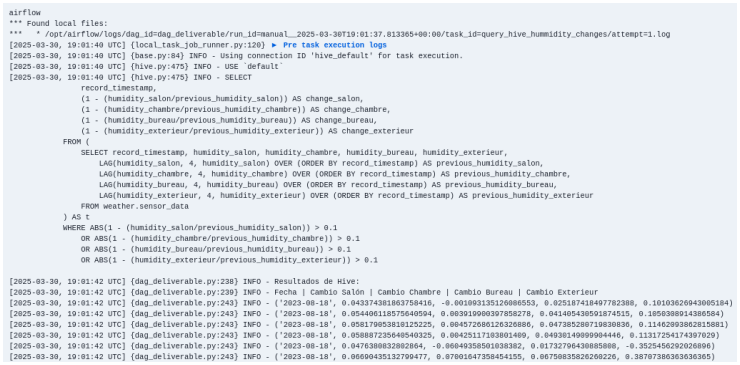

Figure 6: Hive query 3

Challenges encountered

Overall no challenges were found from the technical side, apart from creating the Hive table so that it points to the correct directory in HDFS. Apart from that, there was only one frustrating thing, and it was the need to reinitialize airflow when DAGs changed, as for me, it did not automatically refreshed. Also, debugging the DAG was not trivial, one needed isolate the execution of part of the DAG to reduce time to execute and also wait for the overhead of the DAG initialization. However the logging mechanism was confortable so at least we had that. Other than that, I did not find any further troubles.