

Interview Challenge

Description

Galp is going to open a new store in New York City and our team needs to create a new pipeline to ingest all the generated transactions to our data lake. As the local currency (dollars) is different, we need to use a Rest API to do the exchange rate to euros. Airflow will be used for the store ingestion and NiFi will be used to get the dollar / euro exchange rate.

Airflow

We need to ingest all transactions from the new store in New York. The data is locally stored on a database (MySQL). Once a day, Airflow pipeline will read the data for the previous day and save it on Galp’s data lake.

All transactions table data format:

|  |  |
| --- | --- |
| **Field Name** | **Type** |
| timestamp | Timestamp |
| product\_id | Varchar |
| product\_name | Varchar |
| product\_price | Double |
| cliente\_id | Long |

Think how you can build this pipeline in Airflow.

A: I have no knowledge in Airflow, but in other similar tools. Based on what I quickly searched for, I would create a DAG using the proper connections between the environments for the following tasks in series:

1st Get Largest Timestamp: the objective of this task would be to obtain the max(timestamp) present in the destination and would pass it through XCOM to the next task;

2nd Obtain data: The objective of this task would be to obtain data from the origin where the timestamp was greater than the one verified in the previous step; and carry out the cargo at the destination.

3º Write the result in a log table.

SQL

The table shown below store all the readings of a set of sensors in a refinery

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Type** | **Comments** |
| name | string | Sensor name |
| ts | timestamp | Sensor reading timesatmp |
| value | double | Sensor reading value |
| status | Int | Sensor status |
| year | Int | Sensor reading year |
| month | Int | Sensor reading month |
| day | int | Sensor reading day |

Write the SQL statements that allow you to answer the following questions

A:

• Total number of rows;

select count(\*) qt\_rows from schema.table;

• Number of distinct sensors present on the database;

select count(distinct name) qt\_distinct\_names from schema.table;

• Number of rows for the sensor PPL340;

select count(\*) qt\_ppl340 from schema.table where name=”PPL340”;

• The number of rows by year for the sensor PPL340;

select count(\*) qt\_rows, year

from schema.table

where name=”PPL340”

group by year;

• Average number of readings by year for the sensor PPL340;

with ppl340\_readings as (

select count(\*) qt, year

from schema.table

where name=”PPL340”

group by year

)

select avg(qt) avg\_qt\_ppl340 from ppl340\_readings ;

• For PPL340, Identify the years in which the number of readings is less than the

average;

with yearly\_readings as (

select count(\*) qt, year

from schema.table

group by year

)

select year

from schema.table

where name=”PPL340”

group by year

having count(\*) < ( select avg(qt) from yearly\_readings );

Spark

Answer the questions from the previous exercise (SQL) using any Apache Spark API

A:

# Using PySpark

from pyspark.sql.functions import count, col, avg, countDistinct

df = spark.sql('select \* from schema.table');

df.cache()

# Total number of rows;

df.agg(count("\*").alias("qt\_rows")).display()

# Number of distinct sensors present on the database;

df.agg(countDistinct(col("name")).alias("qt\_distinct\_names")).display()

# Number of rows for the sensor PPL340;

df.where(col("name")=="PPL340").agg(count("\*").alias("qt\_ppl340")).display()

# The number of rows by year for the sensor PPL340;

df.where(col("name")=="PPL340").groupBy(col("year")).agg(count("\*").alias("qt\_rows")).display()

# Average number of readings by year for the sensor PPL340

df.where(col("name")=="PPL340").groupBy(col("year")).agg(count("\*").alias("qt")).agg(avg(col("qt")).alias("avg\_qt\_ppl340")).display()

# For PPL340, Identify the years in which the number of readings is less than the average;

df.where(col("name")=="PPL340").groupBy(col("year")).agg(count("\*").alias("qt")).agg(avg(col("qt")).alias("avg\_qt\_ppl340")).collect()[0]["avg\_qt\_ppl340"]

df.where(col("name")=="PPL340").groupBy(col("year")).agg(count("\*").alias("qt\_rows")).where(col("qt\_rows") < average\_ppl340).display()

Bash

Every six months, we need to run a script bash to delete all the files older than six months. Think how you can create a new script bash to do the housekeeping.

# Using find ... exec to apply rm to all files into filepath older than 180 days (6 months)

find /filepath -mtime +180 -exec rm {} \+

Data Building Tool - dbt

What is dbt? how it works?

DBT it’s a data transformation tool, with connection capability for extract and load in different environments (both DBMS and Big Data environments). Its transformations are primarily based on SQL. It is possible to create and automate data quality tests (such as uniques, constraints, etc) and document data models, serving them in facts and dimensions with star-schema or snowflake to deliver this data.

It works mainly through the creation of facts and dimensions inside the /models folder with the generator codes in “.sql” format, general configuration of the project in the dbt\_project.yml file; and general configuration with tests and documentation of the model in the “.yml” file referring to the table provided as “.sql”.

Python (Database extraction)

Imagine you need to extract all the data from a proprietary database through an ODBC driver on an hourly basis. The primary key is a field called “sensor\_name” and there are more than 46000 sensors. The output of the extraction will be a Parquet file for all the sensors and to a one-hour window (example: 2020-01-01 00:00:00 to 2020-01-01 00:59:59).

Think how you would build your code to do this hourly extraction so that it will be ready to be consumed by other applications as soon as possible.

Fields and data format:

|  |  |
| --- | --- |
| **Field Name** | **Type** |
| timestamp | Timestamp |
| sensor\_name | Varchar |
| value | Double |

Notes:

• It takes more than one hour to complete the query: “SELECT \* FROM table WHERE timestamp >= '2020-01-01 00:00:00' AND timestamp < '2020-01-01

01:00:00'“.

• You cannot modify anything on the database side.

Data lake structure:

• Raw – where source data lands.

• Standardized – transform raw data to Parquet.

• Aggregated – aggregated data based in standardized data.

Note: For all exercises keep in mind the data lake structure.

A: I believe that the best way would be for each successful execution to write in a load control log table the specific interval that that execution would understand and then, in the next load, filter Where timestamp is greater than or equal to the maximum window time of that table in load control. Otherwise, the alternative would require materializing all the parquet files as a table in the raw layer and always before starting the ETL process, checking the destination for the maximum timestamp and based on this data, creating the time window.

Appendix

To create an Airflow cluster use Docker with the image “apache/airflow:1.10.15”.