REAL TIME WEATHER INSIGHTS WITH KAFKA, SPARK AND AIRFLOW

Soutenu par:

- NOUALI Taha
- SAOUD Yassine
- ElOUAZZANI Ayman

Sous la direction de :

- Mr. Cherqi Othmane
- Mr. Ibrahim Rahhal

I. Introduction

Présentation du sujet


api
Apache
Airflow
python
kafka

II. Project Setup

Setup Using Docker-Compose


docker

```yaml
services:

  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    container_name: zookeeper-container
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181

  kafka:
    # image: wlsc/kafka:3.5.1
    image: confluentinc/cp-kafka:latest
    container_name: kafka-container
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092
      KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181


  spark_master:
    image: bitnami/spark:3.5.0
    container_name: spark_master
    ports:
      - 8085:8080
    volumes:
      - ./:/home
      - spark_data:/opt/bitnami/spark/data
```

III. Data Collection

OpenWeather

```python
import requests
import json
import os

from time import sleep, time, ctime
from time import sleep
from datetime import datetime
from kafka import KafkaProducer
from utils import create_table_from_res


api_key = os.getenv("API_Key")
dictionary = json.loads(cities_path)
producer= KafkaProducer(bootstrap_servers='kafka:9092')
TOPIC = "weather_topic"



for key , pos in dictionary.items():
lat = pos[0]
lon = pos[1]
uri = f'https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={api_key}'
r = requests.get(uri)
document = create_table_from_res(r.json())

message = f'{document["longitude"]},{document["latitude"]},{document["temperature"]},
{document["feels_like"]},{document["pressure"]},{document["rain_1h"]},{document["clouds"]},
{document["snow"]},{document["date"]}'

message = bytearray(message.encode("utf-8"))
producer.send(TOPIC, message)
```

IV. Orchestration With Airflow

```python
import requests
import json
import os

from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime
from kafka import KafkaProducer
from time import sleep
from datetime import datetime
from utils import create_table_from_res

def fetch_data():

    api_key = os.getenv("API_KEY")
    producer= KafkaProducer(bootstrap_servers='kafka:9092')
    TOPIC = "weather_topic"
    cities_dict = json.load("./cities_path.json")

    for key , pos in cities_dict.items():
        lat = pos[0]
        lon = pos[1]
        uri = f'https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid=
{api_key}'
        r = requests.get(uri)
        document = create_table_from_res(r.json())

        message = f'{document["longitude"]},{document["latitude"]},{document["temperature"]},
{document["feels_like"]},{document["pressure"]},{document["rain_1h"]},{document["clouds"]},
{document["snow"]},{document["date"]}'
        message = bytearray(message.encode("utf-8"))
        producer.send(TOPIC, message)


with DAG('fetch_data', start_date=datetime(2024, 1, 3),
            schedule_interval="@every_30_seconds",catchup=False) as dag:

        fetch_data_kafka = PythonOperator(
                task_id = 'get_rekrute_data',
                python_callable = fetch_data
        )

        fetch_data_kafka
```

V. Preprocessing With Pyspark Streaming

Connection with kafka

```
df = spark \
    .readStream \
    .format("kafka") \
    .option("checkpointLocation", "/opt/bitnami/spark/checkpoints") \
    .option("kafka.bootstrap.servers", "kafka:9092") \
    .option("subscribe", "weather_topic") \
    .load()
```

Processing code

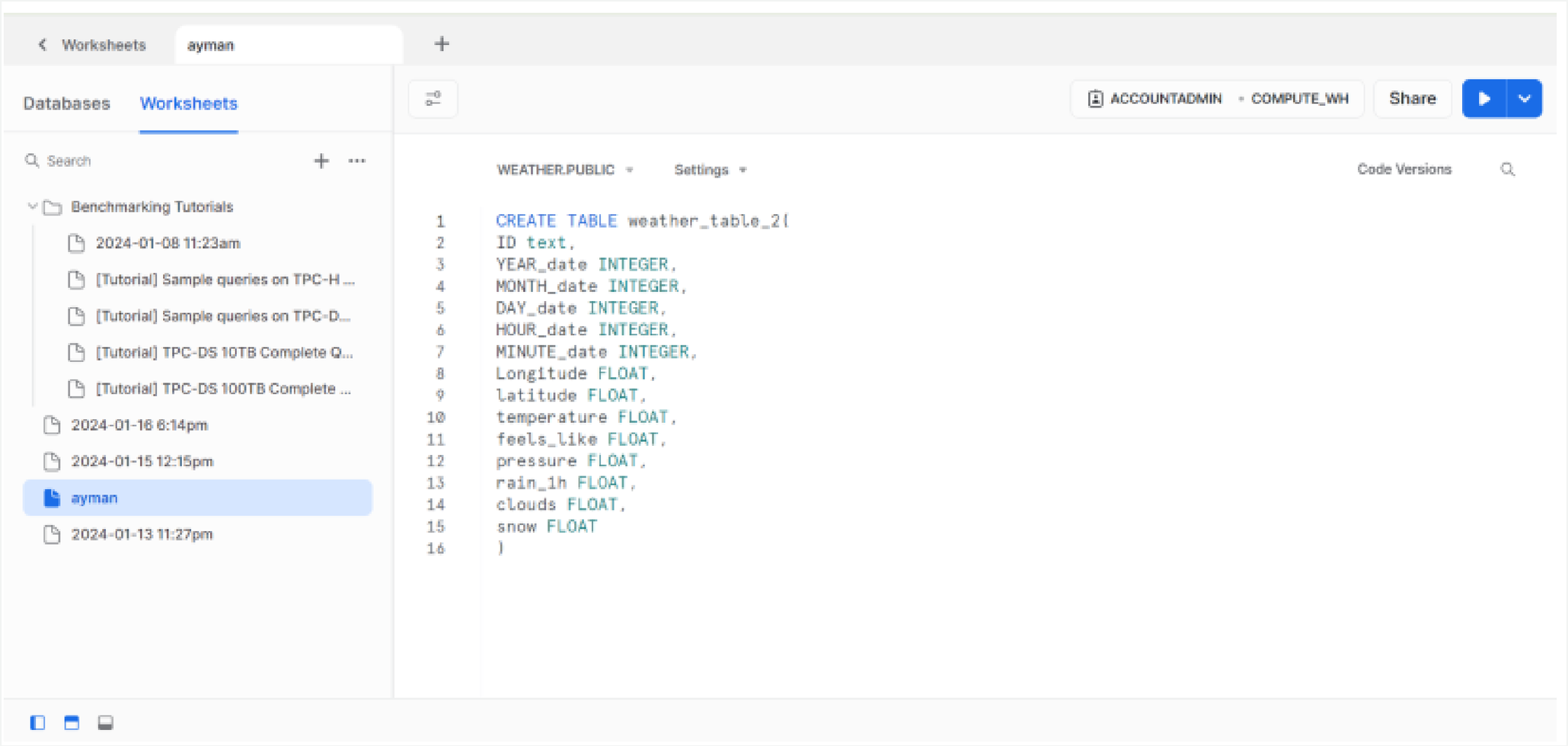
```
parsed_df = df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)").selectExpr(
    "split(value, ',') as data"
).selectExpr(
    "cast(data[0] as Double) as longitude",
    "cast(data[1] as Double) as latitude",
    "cast(data[2] as Double) as temperature",
    "cast(data[3] as Double) as feels_like",
    "cast(data[4] as Double) as pressure",
    "cast(data[5] as Double) as rain_1h",
    "cast(data[6] as Double) as clouds",
    "cast(data[7] as Double) as snow",
    "cast(data[8] as String) as date"
)


reordered_df = parsed_df.select("longitude", "latitude", "temperature", "feels_like",
"pressure", "rain_1h", "clouds", "snow", "date")

df_extracted = reordered_df.withColumn("year", date_format(to_timestamp("date", "yyyy-MM-dd
HH:mm:ss"), "yyyy")) \
                        .withColumn("month", date_format(to_timestamp("date", "yyyy-MM-dd
HH:mm:ss"), "MM")) \
                        .withColumn("day", date_format(to_timestamp("date", "yyyy-MM-dd HH:mm:ss"),
"dd")) \
                        .withColumn("hour", date_format(to_timestamp("date", "yyyy-MM-dd
HH:mm:ss"), "HH")) \
                        .withColumn("minute", date_format(to_timestamp("date", "yyyy-MM-dd
HH:mm:ss"), "mm"))

df_extracted = df_extracted.withColumnRenamed("date", "id")
snowflake_column_order = ["id", "year", "month", "day", "hour", "minute", "longitude",
"latitude", "temperature", "feels_like", "pressure","rain_1h", "clouds", "snow"]
df_ordered = df_extracted.select(snowflake_column_order)
```

Connection with Snowflake

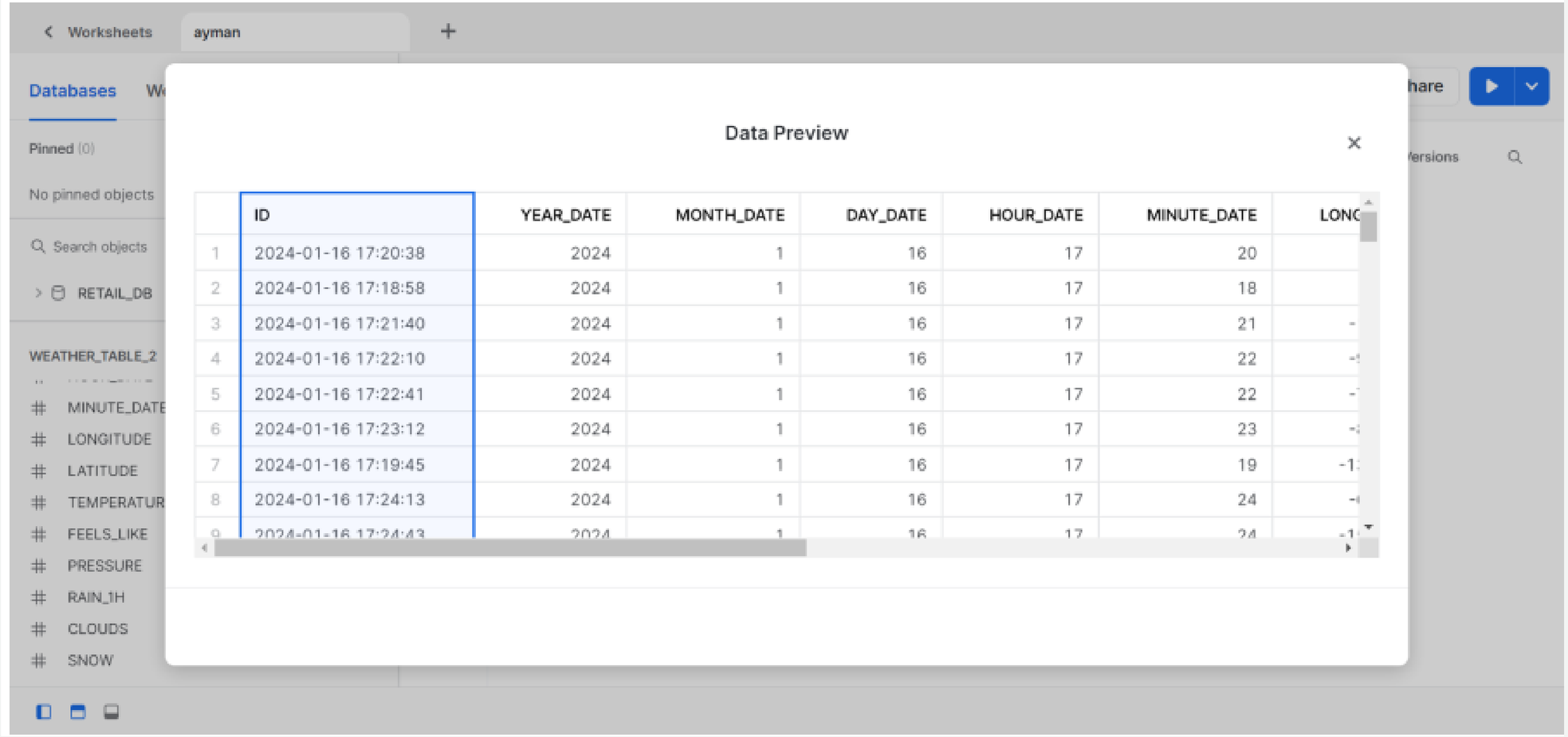
```
df_ordered.writeStream \
    .foreachBatch(
        lambda ds , dt: ds.write
                    .format("net.snowflake.spark.snowflake")
                    .options(**sf_options)
                    .option("dbtable", "weather_table_2")
                    .mode("append")
                    .save()) \
    .start() \
    .awaitTermination()
```

VI. Loading Data In Snowflake Data Platform

Weather_Table



Results :



	ID	YEAR_DATE	MONTH_DATE	DAY_DATE	HOUR_DATE	MINUTE_DATE	LONG
1	2024-01-16 17:20:38	2024	1	16	17	20	
2	2024-01-16 17:18:58	2024	1	16	17	18	
3	2024-01-16 17:21:40	2024	1	16	17	21	-
4	2024-01-16 17:22:10	2024	1	16	17	22	-
5	2024-01-16 17:22:41	2024	1	16	17	22	-
6	2024-01-16 17:23:12	2024	1	16	17	23	-
7	2024-01-16 17:19:45	2024	1	16	17	19	-1
8	2024-01-16 17:24:13	2024	1	16	17	24	-
9	2024-01-16 17:24:43	2024	1	16	17	24	-1

Merci pour votre attention