DOCKER COMPOSE BIG DATA

Étape 1

Créer un `docker-compose` qui gérera une architecture Big Data. Nous voulons, dans un premier temps, exécuter un **Jupyter Notebook**.

Étape 2

Ensuite, nous voulons effectuer une requête API simple pour obtenir des informations météorologiques et les **streamer avec KAFKA**, en utilisant ce fichier comme exemple :

```
import requests
import json
import time
from kafka import KafkaProducer

API_URL = "https://api.open-meteo.com/v1/forecast"
LAT, LON = 52.52, 13.41  # Berlin

KAFKA_TOPIC = "weather_transformed"
KAFKA_BROKER = "XXXX"

def fetch_weather():
    params = {
        "latitude": LAT,
        "longitude": LON,
        "current_weather": "true"
    }
    resp = requests.get(API_URL, params=params, timeout=10)
    resp.raise_for_status()
    return resp.json().get("current_weather", {})

def transform_weather(record: dict) -> dict:
    # Convertir Celsius en Fahrenheit
    if "temperature" in record:
        record["temp_f"] = record["temperature"] * 9/5 + 32
    # Alerte vent fort
```

```
    record["high_wind_alert"] = record.get("windspeed", 0) > 10
    return record

def main():
    producer = KafkaProducer(
        bootstrap_servers=KAFKA_BROKER,
        value_serializer=lambda v: json.dumps(v).encode("utf-8")
    )

    print("Weather streaming producer started...")
    while True:
        try:
            weather = fetch_weather()
            if weather:
                transformed = transform_weather(weather)
                producer.send(KAFKA_TOPIC, transformed)
                producer.flush()
                print("Sent to weather_transformed:", transformed)
        except Exception as e:
            print("Error fetching or sending weather:", e)
        time.sleep(30)  # fetch every 30 seconds

if __name__ == "__main__":
    main()
```

Étape 3

Ensuite, nous voulons avoir un fichier qui permet de calculer la **température moyenne** et le **nombre d'alertes vent fort** sur une fenêtre temporelle d'une minute, en utilisant **SPARK**. Exemple :

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col, window, avg, count
from pyspark.sql.types import StructType, StructField, DoubleType, Boolea
nType, StringType, TimestampType

def main():
    spark = SparkSession.builder \
        .appName("WeatherAggregation") \
```

```
        .master("XXXX (mais pas local[*])") \
        .config(
            "spark.jars.packages",
            "org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.1"
        ) \
        .getOrCreate()

    spark.sparkContext.setLogLevel("WARN")

    schema = StructType([
        StructField("temperature", DoubleType(), True),
        StructField("windspeed", DoubleType(), True),
        StructField("temp_f", DoubleType(), True),
        StructField("high_wind_alert", BooleanType(), True),
        StructField("time", StringType(), True)
    ])

    raw_df = spark.read \
        .format("kafka") \
        .option("kafka.bootstrap.servers", "XXXX") \
        .option("subscribe", "weather_transformed") \
        .option("startingOffsets", "earliest") \
        .load()

    json_df = raw_df.selectExpr("CAST(value AS STRING) as json")
    parsed = json_df.select(from_json(col("json"), schema).alias("data")).sel
ect("data.*")
    parsed = parsed.withColumn("event_time", col("time").cast(TimestampT
ype()))

    agg = parsed.groupBy(
        window(col("event_time"), "1 minute")
    ).agg(
        avg("temperature").alias("avg_temp_c"),
        count(col("high_wind_alert")).alias("alert_count")
    )

    agg.show(truncate=False)
```

```
    spark.stop()

if __name__ == "__main__":
    main()
```

Étape 4

Ajouter du code à la fonction précédente pour **sauvegarder les données dans un fichier CSV sur HADOOP HDFS**, en utilisant `InsecureClient` pour se connecter au NameNode, et en créant un dossier avant :

(Exemple de code à insérer dans la fonction Spark ci-dessus — formulation conservée identique)

Étape finale

Créer un **DAG AIRFLOW** qui permet de lire depuis le stream Kafka et de sauvegarder les alertes dans **HDFS**. Exemple de référence :

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
from kafka import KafkaConsumer
from hdfs import InsecureClient
import json
import traceback

KAFKA_TOPIC = "weather_transformed"
KAFKA_BROKER = "XXXX"
HDFS_DIR = "/user/jovyan/alerts"
HDFS_CLIENT = InsecureClient("XXXX", user="root")


def read_once_and_upload():
    consumer = KafkaConsumer(
        KAFKA_TOPIC,
        bootstrap_servers=KAFKA_BROKER,
        auto_offset_reset="earliest",
        enable_auto_commit=True,
        value_deserializer=lambda v: json.loads(v.decode("utf-8")),
```

```
        consumer_timeout_ms=5000
    )

    try:
        msg = next(consumer)
    except StopIteration:
        print("No messages in Kafka topic.")
        consumer.close()
        return

    record = msg.value
    consumer.close()

    # sérialiser
    json_data = json.dumps(record, indent=2)

    # créer un nom de fichier horodaté
    ts = datetime.utcnow().strftime("%Y%m%dT%H%M%SZ")
    filename = f"alerts_{ts}.json"
    hdfs_path = f"{HDFS_DIR}/{filename}"

    try:
        print("Creating directory and writing to:", hdfs_path)
        HDFS_CLIENT.makedirs(HDFS_DIR)
        # écrire directement, pas de fichier local
        with HDFS_CLIENT.write(hdfs_path, overwrite=False, encoding="utf-
8") as writer:
            writer.write(json_data)
        print("Upload successful:", hdfs_path)
    except Exception:
        print("❌ HDFS ERROR:")
        traceback.print_exc()
        raise


with DAG(
    dag_id="weather_alert_direct_to_hdfs_timestamped",
    start_date=datetime(2024, 1, 1),
```

```
    schedule=None,
    catchup=False,
) as dag:

    read_once = PythonOperator(
        task_id="consume_one_message",
        python_callable=read_once_and_upload,
    )

read_once
```