Arquitectura de Streaming en Tiempo Real

Detección de Trending Topics con Apache Kafka y Spark Structured Streaming.

AUTORES:
Yahya El Baroudi
Samuel Corrionero
Ismael González
Jairo Farfán

Objetivo del proyecto

Problema:

Batch Processing

Las redes sociales generan millones de datos por segundo. Procesarlos “por lotes” (batch) es demasiado lento.

Solución:

Streaming Architecture

Implementar una arquitectura Streaming que analice la información en el momento exacto en que se genera.

Caso de Uso:

Trending Topics

Un simulador de Twitter que detecta los hashtags más populares cada minuto (Trending Topics).

Diseño e Infraestructura

Diseño basado en **Productor - Intermediario - Consumidor.**

Infraestructura: Virtualizada con Docker para garantizar la portabilidad.

Tecnologías utilizadas

Docker: Orquestación de contenedores (Zookeeper + Kafka).

Apache Kafka: Ingesta de datos masiva y desacoplada.

Apache Spark (PySpark): Motor de procesamiento distribuido.

API: Uso de Spark Dataframes y Structured Streaming.

Fragmento de docker-compose.yml

```yaml
services:
  # ------------------------------------------------------------------------------
  # SERVICIO 1: ZOOKEEPER (El Coordinador)
  # Kafka no puede funcionar solo; necesita a Zookeeper para gestionar el clúster.
  # Se encarga de elegir el líder, guardar la configuración y saber qué nodos están vivos.
  # ------------------------------------------------------------------------------
  zookeeper:
    image: confluentinc/cp-zookeeper:7.4.4  # Imagen oficial de Confluent (muy estable)
    hostname: zookeeper
    container_name: zookeeper  # Nombre fijo para que Kafka lo encuentre por DNS interno
    ports:
      - "2181:2181"  # Puerto estándar de administración (PC:Contenedor)
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181  # Puerto donde Kafka buscará a Zookeeper
      ZOOKEEPER_TICK_TIME: 2000    # "Latido" del corazón del sistema (en ms)

  # ------------------------------------------------------------------------------
  # SERVICIO 2: KAFKA (El Buzón / Broker)
  # Es el sistema de mensajería. Recibe datos (Productor) y los sirve (Spark).
  # ------------------------------------------------------------------------------
  kafka:
    image: confluentinc/cp-kafka:7.4.4
    hostname: kafka
    container_name: kafka
    depends_on:
      - zookeeper  # Orden a Docker: "¡No arranques Kafka hasta que Zookeeper esté listo!"
    ports:
      - "9092:9092"  # El puerto CRÍTICO. Aquí se conectarán vuestros scripts de Python.

    environment:
      # Identificador único de este nodo de Kafka dentro del clúster
      KAFKA_BROKER_ID: 1

      # Conexión interna con el jefe (Zookeeper).
```

El Productor (Simulador de datos)

Script en Python: Simulación de usuarios reales.

Generación aleatoria de tweets: Con hashtags (#BigData, #Spark, #Python).

Envío de datos: Formato JSON serializado.

Ejecución del simulador en tiempo real.

El Consumidor (Spark Structured Streaming)

El **núcleo** del proyecto.

Tratamos el flujo de datos infinito como una **Tabla Infinita (Unbounded Table)**.

Uso de **Dataframes**: Definición de esquema (Schema Enforcement) para estructurar el JSON entrante.

```
schema = StructType([
    StructField("usuario", StringType(), True),
    StructField("texto", StringType(), True),
    StructField("hashtag_principal", StringType(), True),
    StructField("timestamp", DoubleType(), True),
])
```

Definición de Esquema

Definición de Esquema

```
spark_df = read_stream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", KAFKA_BOOTSTRAP_SERVERS) \
    .option("subscribe", KAFKA_TOPIC) \
    .option("startingOffsets", "latest") \
    .load()
)
```

Lectura del Stream

Lectura del Stream

```
Top hashtags en este minuto:
+------------------------------------------+-----------------+---------------+
|window                                    |hashtag_principal|num_ocurrencias|
+------------------------------------------+-----------------+---------------+
|{2025-12-15 16:07:00, 2025-12-15 16:08:00}|#IA              |11             |
|{2025-12-15 16:07:00, 2025-12-15 16:08:00}|#Streaming       |11             |
|{2025-12-15 16:07:00, 2025-12-15 16:08:00}|#Spark           |10             |
|{2025-12-15 16:07:00, 2025-12-15 16:08:00}|#RealTime        |9              |
|{2025-12-15 16:07:00, 2025-12-15 16:08:00}|#Kafka           |8              |
|{2025-12-15 16:07:00, 2025-12-15 16:08:00}|#Examen          |8              |
|{2025-12-15 16:07:00, 2025-12-15 16:08:00}|#BigData         |7              |
|{2025-12-15 16:07:00, 2025-12-15 16:08:00}|#Python          |3              |
+------------------------------------------+-----------------+---------------+

=========================================
```

Salida en Consola

Lógica de Procesamiento (Transformaciones)

Limpieza: Parseo de JSON a columnas.

Explosión: Función explode() para separar múltiples hashtags.

Windowing (Ventanas de Tiempo): Agrupación de datos en ventanas de 60 segundos.

Resultado: Conteo en vivo de ocurrencias.

```
# ============================
# 4. AGREGACIÓN EN VENTANAS DE 60 s (minuto real)
# ============================

windowed_counts = (
    df_with_ts
    .withWatermark("ts", "2 minutes")
    .groupBy(
        window(col("ts"), "60 seconds"),   # ventana FIJA de 1 minuto
        col("hashtag_principal")
    )
    .agg(count("*").alias("num_ocurrencias"))
)

MAX_HASHTAGS = 10  # mostramos hasta 10 si hay
```

```
# 3. PARSEAR EL JSON
# ============================

schema = StructType([
    StructField("usuario", StringType(), True),
    StructField("texto", StringType(), True),
    StructField("hashtag_principal", StringType(), True),
    StructField("timestamp", DoubleType(), True),
])

parsed_df = (
    value_df
    .select(from_json(col("json_str"), schema).alias("data"))
    .select("data.*")
)

df_with_ts = (
    parsed_df
    .where(col("hashtag_principal").isNotNull())
    .withColumn("ts", current_timestamp())
)
```

Conclusiones y Retos

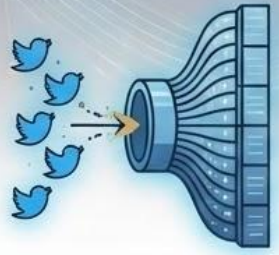
Retos Superados

Configuración de red en Docker (comunicación entre contenedores y host).

Compatibilidad de versiones (Librerías JAR de Spark-Kafka).

Conclusiones

Spark Structured Streaming simplifica el Big Data usando la misma lógica que SQL.



La arquitectura es escalable: podríamos procesar millones de tweets reales con el mismo código.

MUCHAS GRACIAS

POR VUESTRA ATENCIÓN