Procesamiento de Flujo de Datos Masivos

Estrategia de aplicación de Big Data

Integrantes

Mario Benitez

Marcelo Roig

Edgar Mendoza

Rolando Vega

Gustavo Weingartshofer

Big Data & Business Analytics

Especialización en TIC aplicadas a la Educación Superior

Departamento de Elearning
Facultad Politécnica
Universidad Nacional de Asunción

Identificación

Módulo: Procesamiento de Flujo de Datos Masivos.
Curso: Big Data y Business Analytics.

Objetivos

Capturar un Flujo de datos para su posterior análisis.

Actividad

El objetivo de esta tarea es fijar los contenidos desarrollados durante el módulo de Data Streaming, incluyendo la aplicación de herramientas como Redpanda y KSQLDB para la solución de un problema.

La tarea consiste en construir un servicio que consuma la API en tiempo real de Finnhub: https://finnhub.io/docs/api/websocket-trades y consuma actualizaciones para los siguientes símbolos:

- AAPL
- AMZN
- BINANCE:BTCUSDT

De modo a procesar las actualizaciones, deben seguirse los siguientes pasos:

1. Instalar un cluster Redpanda de manera local utilizando un archivo docker-compose.yml.

2. Implementar un producer utilizando kafka-python, de acuerdo a la documentación de Redpanda y similar al ejemplo desarrollado en clase, para suscribirse a los eventos de la API.

3. Instalar KSQLDB modificando el archivo docker-compose.yml, de acuerdo a la documentación de Redpanda.

De forma a contestar los primeros 3 items, se tiene el repositorio ya en GitHub: https://github.com/Grupo1FPUNA0BIGDATA/BigData_KSQL.git

Donde se observa el Código en secciones:
En primer lugar, se corren la primera y segunda parte del Código "docker-compose.yml" con la finalidad de instalar en Docker un contenedor con Redpanda y un extra Redpanda-console, la consola sirve para la observación de la correcta ejecución de los streams provenientes de FinnHub.

Tras la instalación de ello, se crea un ambiente virtual ya que se trabajará con Python3, se activa y se procede a instalar y actualizar pip.

Paso seguido, se instala Kafka-Python y Websocket-client. Tras ello, se crea En el Redpanda un Topic para recibir los streams proveniente de FinnHub.

Luego de ello, se crea un archivo "Producer.py", el mismo se extrae del github: https://github.com/rparrapy/rp-stock-monitor, similar al desarrollado en clases con el profesor de la materia, este archivo es el encargado de recibir el flujo y almacenarlo en Redpanda, para ello se realiza una transformación ETL de forma a ir adecuando la salida del productor (o lo que se va a almacenar en Redpanda) en formato JSON.

Una vez que el producer.py va recibiendo los streams de FinnHub, se van almacenando en Redpanda en formato JSON, se vuelve nuevamente al código de "docker-compose.yml" para correr la tercera parte, la cual es el KSQL.

En la instalación del KSQL, se van configurando los puertos, se instala tanto un Servidor como un cliente; tras haber finalizado la instalación, si los puertos coinciden, entonces se podrá ingresar a la base de datos y se crea el stream que va ligado al topic configurado en el "Producer.py" y en Redpanda.

A continuación los comandos usados en SQL:

Creación del Stream:

```
CREATE STREAM Trading (Senhal STRING, precio DOUBLE, volumen DOUBLE, fecha STRING)
  WITH (kafka_topic='stock-trading', value_format='json', partitions=1);
```

Luego las tablas:

```
CREATE TABLE Promedio_Senhal AS
        SELECT  Senhal, avg(Precio) AS Precio_Prom
        FROM Trading
        GROUP BY Senhal
        EMIT CHANGES;

CREATE TABLE Cantidad_Senhal AS
        SELECT Senhal, count(Senhal) AS Cantidad_Senhales
        FROM Trading
        GROUP BY Senhal
        EMIT CHANGES;

CREATE TABLE Precio_Senhal_Max AS
        SELECT Senhal, max(Precio) AS Precio_Maximo
        FROM Trading
        GROUP BY Senhal
        EMIT CHANGES;

CREATE TABLE Precio_Senhal_Min AS
        SELECT Senhal, min(Precio) AS Precio_Minimo
        FROM Trading
        GROUP BY Senhal
        EMIT CHANGES;
```

Para finalmente poder contestar las 4 consultas:

4. Ejecutar ksqldb-cli, definir los streams y las tablas necesarias para responder a las siguientes preguntas:

4.1) ¿Cuál fue el promedio ponderado de precio de una unidad por cada uno de los símbolos procesados? (e.j. AAPL)

select * from promedio_senhal emit changes;

```
+----------------------------------------+----------------------------------------+
|SENHAL                                  |PRECIO_PROM                             |
+----------------------------------------+----------------------------------------+
|BINANCE:BTCUSDT                         |26072.29251797424                       |
```

4.2) ¿Cuántas transacciones se procesaron por símbolo?

select * from cantidad_senhal emit changes;

```
+----------------------------------------+----------------------------------------+
|SENHAL                                  |CANTIDAD_SENHALES                       |
+----------------------------------------+----------------------------------------+
|BINANCE:BTCUSDT                         |6908                                    |
```

4.3) ¿Cuál fue el máximo precio registrado por símbolo?

select * from precio_senhal_max emit changes;

```
+----------------------------------------+----------------------------------------+
|SENHAL                                  |PRECIO_MAXIMO                           |
+----------------------------------------+----------------------------------------+
|BINANCE:BTCUSDT                         |26111.36                                |
```

4.4) ¿Cuál fue el mínimo precio registrado por símbolo?

select * from precio_senhal_min emit changes;

```
+----------------------------------------+----------------------------------------+
|SENHAL                                  |PRECIO_MINIMO                           |
+----------------------------------------+----------------------------------------+
|BINANCE:BTCUSDT                         |26052.5                                 |
```

Material Audio Visual:
https://drive.google.com/drive/folders/1GpGcF_kSLx14DYcqtyblYJ6elrKymdTk?usp=sharing