



Departamento de Elearning
Facultad Politécnica
Universidad Nacional de Asunción

Módulo: **Data Streaming**

Diplomado: **Big Data y Business Analytics**

Unidad V - Actividad 1

Integrantes:

- Rodney Fabian Carreras Galeano
- Elizabeth Caballero Marecos
- Adela Adriana Davalos Jara
- Silvestre Bernal

Agosto - 2023

TAREA MÓDULO DATA STREAMING

Objetivo

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.

Instrucciones

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



```
stock-monitor > docker-compose.yml
1  ---
2  version: '3.9'
3  name: redpanda-ksqlfp
4  networks:
5  redpanda_network:
6    driver: bridge
7  volumes:
8    redpanda: null
9  services:
10 redpanda:
11   command:
12     - redpanda
13     - start
14     - --smp
15     - '1'
16     - --reserve-memory
17     - 0M
18     - --overprovisioned
19     - --set
20     - redpanda.cluster_id=turning-red
21     - --set
22     - redpanda.enable_idempotence=true
23     - --set
24     - redpanda.enable_transactions=true
25     - --set
26     - redpanda.auto_create_topics_enabled=true
27     - --node-id
28     - '0'
29     - --kafka-addr
30     - PLAINTEXT://0.0.0.0:29092,OUTSIDE://0.0.0.0:9092
31     - --advertise-kafka-addr
32     - PLAINTEXT://redpanda:29092,OUTSIDE://localhost:9092
33   image: docker.vectorized.io/vectorized/redpanda:v21.11.11
34   container_name: redpanda
35   ports:
36     - 9092:9092
37     - 29092:29092
```

Levantar los contenedores

`docker compose up -d`

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.

```
producer.py X
producer.py > ...
1
2 import websocket
3 import json
4 from datetime import datetime
5 from kafka import KafkaProducer
6 from kafka.errors import KafkaError
7
8
9 producer = KafkaProducer(
10     bootstrap_servers = "localhost:9092",
11     value_serializer=lambda m: json.dumps(m).encode("ascii"),
12 )
13
14 topic = "stock-updates"
15 def on_ws_message(ws, message):
16     data = json.loads(message)["data"]
17     records = [
18         {"simbolo":d["s"],
19          "precio":d["p"],
20          "volumen":d["v"],
21          "fecha_hora": datetime.utcfromtimestamp(d["t"] / 1000).strftime(
22              "%Y-%m-%d %H:%M:%S"
23          ),
24         }
25         for d in data
26     ]
27
28     for r in records:
29         future = producer.send(topic, value=r)
30         future.add_callback(on_success)
31         future.add_errback(on_error)
32     producer.flush()
33
34 def on_ws_error(ws, error):
35     print(error)
36
37 def on_ws_close(ws, close_status_code, close_msg):
38     producer.flush()
39     producer.close()
40     print("### closed ###")
41
42 def on_ws_open(ws):
43     ws.send({'type':"subscribe","symbol":"AAPL"})
44     ws.send({'type':"subscribe","symbol":"AMZN"})
45     ws.send({'type':"subscribe","symbol":"BINANCE:BTCUSDT"})
46
47 def on_success(metadata):
48     print(f"Mensaje producido para Topic '{metadata.topic}' at offset {metadata.offset}")
49
50 def on_error(e):
51     print(f"Error enviando el mensaje: {e}")
52
53 if __name__ == "__main__":
54     websocket.enableTrace(True)
55     ws = websocket.WebSocketApp("wss://ws.finnhub.io?token=cjggo9hr01qh977engqgcjggo9hr01qh977",
56                                on_message = on_ws_message,
57                                on_error = on_ws_error,
58                                on_close = on_ws_close)
59     ws.on_open = on_ws_open
60     ws.run_forever()
```

3. Instalar [KSQLDB](#) modificando el archivo `docker-compose.yml`, de acuerdo a la [documentación](#) de Redpanda.

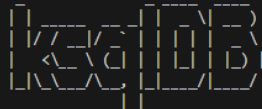
```

38
39 ksqldb-server:
40   image: confluentinc/ksqldb-server:0.25.1
41   hostname: ksqldb-server
42   container_name: ksqldb-server
43   depends_on:
44     - redpanda
45   ports:
46     - "8088:8088"
47   environment:
48     KSQL_LISTENERS: "http://0.0.0.0:8088"
49     KSQL_BOOTSTRAP_SERVERS: "redpanda:29092"
50     KSQL_KSQL_SCHEMA_REGISTRY_URL: "http://schema-registry:8081"
51     KSQL_KSQL_LOGGING_PROCESSING_STREAM_AUTO_CREATE: "true"
52     KSQL_KSQL_LOGGING_PROCESSING_TOPIC_AUTO_CREATE: "true"
53
54 ksqldb-cli:
55   image: confluentinc/ksqldb-cli:0.25.1
56   container_name: ksqldb-cli
57   depends_on:
58     - redpanda
59     - ksqldb-server
60   entrypoint: /bin/sh
61   tty: true

```

4. Ejecutar `ksqldb-cli`, definir los streams y tablas necesarios para responder a las siguientes preguntas:
`docker exec -it ksqldb-cli ksql http://ksqldb-server:8088`

```
(env) PS C:\Trabajos\stock-monitor> docker exec -it ksqldb-cli ksql http://ksqldb-server:8088
OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in version 9.0 and will likely be removed in a future
release.
```



```
=====
=                                     =
=  ksql                             =
=                                     =
=                                     =
=                                     =
=                                     =
=                                     =
=                                     =
=                                     =
=                                     =
=                                     =
=====
The Database purpose-built
for stream processing apps
=====
```

Copyright 2017-2022 Confluent Inc.

CLI v0.25.1 Server v0.25.1 located at http://ksqldb-server:8088

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

```
ksql> CREATE TABLE PROMEDIO_X_SIMBOLO AS
>SELECT simbolo,
>(sum(PRECIO)/COUNT(*)) PRECIO_PROMEDIO
>from stock
>GROUP BY simbolo
>EMIT CHANGES;

ksql> Select * from promedio_x_simbolo;
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|SIMBOLO|                                     |PRECIO_PROMEDIO|
|-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|BINANCE:BTCUSD|                               |26166.884738559747|
Query terminated
ksql>
```

b. ¿Cuántas transacciones se procesaron por símbolo?

```
ksql> CREATE TABLE CANTIDAD_X_SIMBOLO AS
>SELECT simbolo,
>count(*) cantidad_procesada
>from stock
>GROUP BY simbolo
>EMIT CHANGES;

Message
-----
Created query with ID CTAS_CANTIDAD_X_SIMBOLO_30
-----

ksql> Select * from CANTIDAD_X_SIMBOLO;
+-----+-----+
|SIMBOLO|CANTIDAD_PROCESADA|
+-----+-----+
|BINANCE:BTCUSD|5697|
+-----+-----+
Query terminated
ksql>
```

c. ¿Cuál fue el máximo precio registrado por símbolo?

```
ksql> CREATE TABLE MAXIMO_X_SIMBOLO AS
>SELECT simbolo,
>MAX(PRECIO) PRECIO_MAXIMO
>from stock
>GROUP BY simbolo
>EMIT CHANGES;

Message
-----
Created query with ID CTAS_MAXIMO_X_SIMBOLO_34
-----

ksql> SELECT * FROM MAXIMO_X_SIMBOLO;
+-----+-----+
|SIMBOLO|PRECIO_MAXIMO|
+-----+-----+
|BINANCE:BTCUSD|26197.48|
+-----+-----+
Query terminated
ksql>
```

d. ¿Cuál fue el mínimo precio registrado por símbolo?

```
ksql> CREATE TABLE MINIMO_X_SIMBOLO AS
>SELECT simbolo,
>MIN(PRECIO) PRECIO_MINIMO
>from stock
>GROUP BY simbolo
>EMIT CHANGES;

Message
-----
Created query with ID CTAS_MINIMO_X_SIMBOLO_38
-----

ksql> SELECT * FROM MINIMO_X_SIMBOLO;
+-----+-----+
|SIMBOLO|PRECIO_MINIMO|
+-----+-----+
|BINANCE:BTCUSD|26149.5|
+-----+-----+
Query terminated
ksql>
```

Presentación

El proyecto debe desarrollarse en los grupos previamente formados. El entregable debe incluir:

Enlace a repositorio público de Github incluyendo:

- docker-compose.yml
- código fuente del consumer
- **Link:** [silverespacio/stock-monitor: Obtiene Actualizaciones de API \(github.com\)](https://github.com/silverespacio/stock-monitor)
- archivos .sql con las 4 consultas utilizadas para responder las preguntas de la sección anterior
- <https://github.com/silverespacio/stock-monitor/blob/main/Varios/ksql-ApiStream.sql>

Una grabación de pantalla mostrando el sistema en funcionamiento y explicando brevemente los pasos que se siguieron para su implementación.

Link: <https://github.com/silverespacio/stock-monitor/blob/main/Varios/Redpanda-ksql.mp4>