COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI ICADE CIHS

Sprint 6. Historias de Usuario 7 y 8

Elena Conderana Medem y Sergio Cuenca Núñez

Tecnologías de Datos Masivos
Big Data Technology

Índice

Introducción

1. Contexto

El objetivo final del proyecto busca diseñar e implementar una arquitectura Big Data completa, que permita procesar y analizar datos procedentes de `TradingView` para poder asesorar a clientes en el trading de criptomonedas. Este asesoramiento se compone de procesamiento en batch, cubierto en los sprints iniciales, y de procesamiento en tiempo real de las cotizaciones, motivo de los últimos sprints. Este último sprint se centra en almacenar la información en tiempo real de la herramienta ` TradingView` en un índice en ElasticSearch y en visualizar los mismos en vivo con Kibana.

2. Descripción del problema

La obtención de valores de cotización en tiempo real es muy elevado, lo cual complica observar cómo están evolucionando las cotizaciones y entender el contexto simplemente leyendo los números en una tabla. La visualización mediante un simple gráfico de línea permite apreciar el valor de las cotizaciones y su evolución de manera rápida y sencilla.

3. Objetivos

Los objetivos principales de esta práctica comprenden:

- Familiarización con ElasticSearch.
- Almacenamiento de las cotizaciones en tiempo real de Solana en ElasticSearch.
- Familiarización con Kibana.
- Visualización de las cotizaciones en tiempo real de Solana con Kibana.

4. Justificación

Para analizar en tiempo real las criptomonedas, visualizar la evolución de sus cotizaciones es esencial, pues permite ver sencilla y ordenadamente cómo evoluciona. Para poder realizar esta visualización con Kibana es necesario almacenar los datos obtenidos por el *producer* y consumidos por el *consumer* en ElasticSearch.

Metodología

En la siguiente sección se procede a describir en detalle cómo se han almacenado los datos obtenidos de la herramienta `TradingView` para Solana en ElasticSearch y cómo se ha empleado Kibana para realizar la visualización de los mismos. Se abordarán cuestiones técnicas sobre el proceso de desarrollo y sobre las tecnologías utilizadas.

5. Descripción del entorno de desarrollo

Este sprint se desarrolla en local utilizando la plataforma `Visual Studio Code` como entorno de desarrollo integrado principal. La solución se ha desarrollado en Python, haciendo uso de las siguientes 9 librerías:

- `kafka`: para la interacción con el clúster de Apache Kafka, incluyendo la producción y el consumo de mensajes.
- `json`: para la manipulación y serialización de datos en formato JSON.
- `random`: para la generación de cadenas aleatorias en la creación de sesiones.
- `re`: para el uso de expresiones regulares en el análisis de los mensajes recibidos del *websocket*.
- `string`: para la manipulación de cadenas para obtener caracteres ASCII en la generación de sesiones.
- `time`: para la gestión de pausas y esperas en la lógica de reconexión.
- `datetime`: para la obtención de la marca de tiempo actual y su formateo.
- `websocket`: para establecer y mantener la conexión *websocket* con el proveedor de datos en tiempo real.
- `requests`: para interactuar con el servidor de Elasticsearch.

Para realizar el acceso en tiempo real a los datos de cotización de Solana se va a emplear Apache Kafka, una plataforma distribuida para transmisión de datos que permite almacenar, procesar y suscribirse a flujos de datos de varias fuentes de manera inmediata. En este caso, actuará como el canal de comunicación principal para los datos de precios de Solana obtenidos en tiempo real. Dichos datos se almacenarán en un índice de Elasticsearch, un motor de búsqueda y análisis distribuido ampliamente utilizado para manejar grandes volúmenes de datos en tiempo real. Por último, los datos almacenados se visualizarán mediante un *dashboard* de Kibana, una plataforma de visualización de datos de código abierto desarrollada para explorar, visualizar y analizar datos almacenados en Elasticsearch.

6. Diseño de la solución

La solución consta de dos componentes principales, un productor de datos en tiempo real y un consumidor de dichos datos. El productor establece una conexión *websocket* con la herramienta `TradingView` y genera una sesión única suscribiéndose al símbolo de cotización deseado, que en este caso es la criptomoneda Solana. Tras establecer esta conexión el productor recibe continuamente los mensajes que se transmiten por el *websocket*. El productor extrae los valores de precio, cambio, porcentaje de cambio y volumen de cada mensaje y junto con una marca temporal forma un *payload* en formato JSON. Los *payloads* de JSON se envían como valor de los mensajes al *topic* de Kafka gittba_SOL, con clave "solana". La implementación descrita corresponde al Código 1.

```python
import json
import random
import re
import string
import time
from datetime import datetime
from websocket import create_connection, WebSocketConnectionClosedException
from kafka import KafkaProducer

# Configuración de Kafka
KAFKA_BROKER = '192.168.80.34:9092'
KAFKA_TOPIC = 'gittba_SOL'

# Crea el productor de Kafka
producer = KafkaProducer(
    bootstrap_servers=KAFKA_BROKER,
    key_serializer=lambda k: k.encode('utf-8'),
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

def generate_session():
    string_length = 12
    letters = string.ascii_lowercase
    return "qs_" + "".join(random.choice(letters) for _ in range(string_length))

def prepend_header(content):
    return f"~m~{len(content)}~m~{content}"

def construct_message(func, param_list):
    return json.dumps({"m": func, "p": param_list}, separators=(",", ":"))

def create_message(func, param_list):
    return prepend_header(construct_message(func, param_list))

def send_message(ws, func, args):
    try:
        ws.send(create_message(func, args))
    except WebSocketConnectionClosedException:
        print("Conexion cerrada mientras se intentaba enviar un mensaje.")
        reconnect(ws)

def send_ping(ws):
    try:
        ping_message = "~h~0"
        ws.send(ping_message)
    except Exception as e:
        print(f"Error enviando ping: {e}")

def process_data(data):
    price = data.get("lp", "No disponible")
    change = data.get("ch", "No disponible")
    change_percentage = data.get("chp", "No disponible")
    volume = data.get("volume", "No disponible")
    timestamp = datetime.now().strftime("[%Y-%m-%d %H:%M:%S]")

    payload = {
        "timestamp": timestamp,
        "price": price,
        "change": change,
```

```
        "change_percentage": change_percentage,
        "volume": volume
    }

    print(f"[{timestamp}] Enviando a Kafka: {payload}")
    producer.send(KAFKA_TOPIC, key="solana", value=payload)
    producer.flush()

def reconnect(symbol_id):
    print("Intentando reconectar...")
    time.sleep(5)
    start_socket(symbol_id)

def start_socket(symbol_id):
    session = generate_session()
    url = "wss://data.tradingview.com/socket.io/websocket"
    headers = json.dumps({"Origin": "https://data.tradingview.com"})

    try:
        ws = create_connection(url, headers=headers)
        print(f"Conectado a {url}")

        send_message(ws, "quote_create_session", [session])
        send_message(ws, "quote_set_fields", [session, "lp", "ch", "chp", "volume"])
        send_message(ws, "quote_add_symbols", [session, symbol_id])

        while True:
            try:
                result = ws.recv()
                if result.startswith("~m~"):
                    data_match = re.search(r"\{.*\}", result)
                    if data_match:
                        message = json.loads(data_match.group(0))
                        if message["m"] == "qsd":
                            process_data(message["p"][1]["v"])
                elif result.startswith("~h~"):
                    send_ping(ws)

            except WebSocketConnectionClosedException:
                print("Conexión cerrada inesperadamente.")
                reconnect(symbol_id)
                break
            except Exception as e:
                print(f"Error procesando mensaje: {e}")
                continue

    except WebSocketConnectionClosedException as e:
        print(f"Error al conectar: {e}. Reconectando en 5 segundos...")
        reconnect(symbol_id)
    except Exception as e:
        print(f"Error inesperado: {e}. Reconectando en 5 segundos...")
        reconnect(symbol_id)

if __name__ == "__main__":
    symbol_id = "BINANCE:SOLUSDT"
    start_socket(symbol_id)
```

Código 1. Productor de Datos en Tiempo Real.

Asimismo, el consumidor se conecta al broker de Apache Kafka y se suscribe al tópico gittba_SOL, en concreto a la partición 0. Periódicamente realiza un *polling* con un timeout de 5000ms para obtener nuevos mensajes. Para cada mensaje se verifica la tipología de los campos precio y volumen y se asigna un timestamp. Con estos datos se construye una URL para realizar una petición PUT a Elasticsearch con el contenido del mensaje y el offset como su id. El Código 2 muestra la implementación del consumidor de Apache Kafka descrita.

```
from kafka import KafkaConsumer
from kafka.structs import TopicPartition

# Crea el KafkaConsumer
consumer = KafkaConsumer(
    bootstrap_servers=['192.168.80.34:9092'],
    group_id='gittba_group_sol',
    auto_offset_reset='earliest'
)

# Asigna topic y partición
consumer.assign([TopicPartition('gittba_SOL', 0)])

# Lee los mensajes
records = consumer.poll(timeout_ms=5000)

# Procesa los mensajes
for topic_data, consumer_records in records.items():
    print("TopicPartition:", topic_data)
    for consumer_record in consumer_records:
        print("key:       " + str(consumer_record.key.decode('utf-8')))
        print("value:     " + str(consumer_record.value.decode('utf-8')))
        print("offset:    " + str(consumer_record.offset))
        print("timestamp: " + str(consumer_record.timestamp))

# Cierra el consumidor
consumer.close()
```

Código 2. Consumidor de Datos en Tiempo Real.

En paralelo a la ejecución del productor y del consumidor es necesario acceder a Kibana, crear el *index pattern* asociado a Solana y crear un *dashboard* con los precios de cotización en tiempo real. A modo de síntesis la Figura 1 muestra de manera visual y simplificada el funcionamiento de la solución implementada.

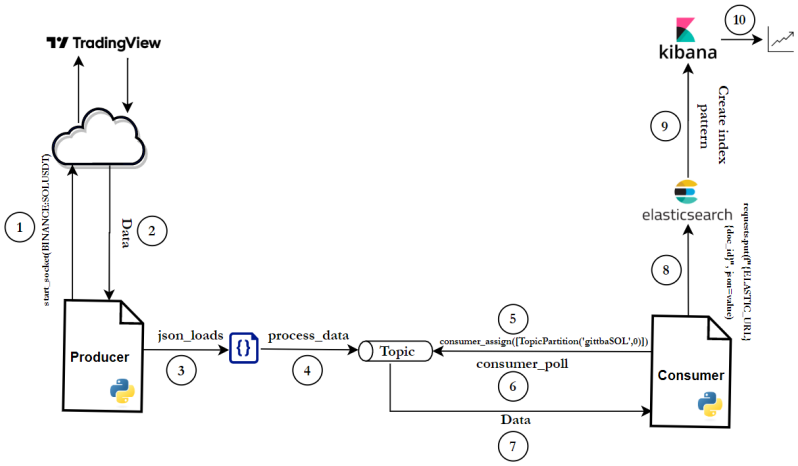

Figura 1. Diagrama de Flujo de Solución Implementada.

7. Pruebas realizadas

La comprobación de la bondad de la solución desarrollada se realiza comparando el gráfico con los precios de cierre de Solana obtenido en Kibana los valores correspondientes a los mismos instantes temporales ofrecidos por `TradingView`.

Resultados

A continuación, se procede a estudiar los resultados obtenidos durante la ejecución de la práctica.

8. Descripción de los resultados

Con la metodología anterior se consigue obtener los datos de instante temporal, precio, cambio, porcentaje de cambio y volumen en tiempo real para la criptomoneda Solana. De dichos valores se emplea el instante temporal y el precio de cierre para realizar una visualización de la evolución en tiempo real del precio de cotización de Solana.

9. Pantallazos de la Ejecución

La Figura 2 muestra la cotización de Solana en tiempo real, mientras que en la Figura 3 se aprecia la cotización extraída directamente de `TradingView`.

Figura 2. Cotización de Solana en Tiempo Real en Kibana.

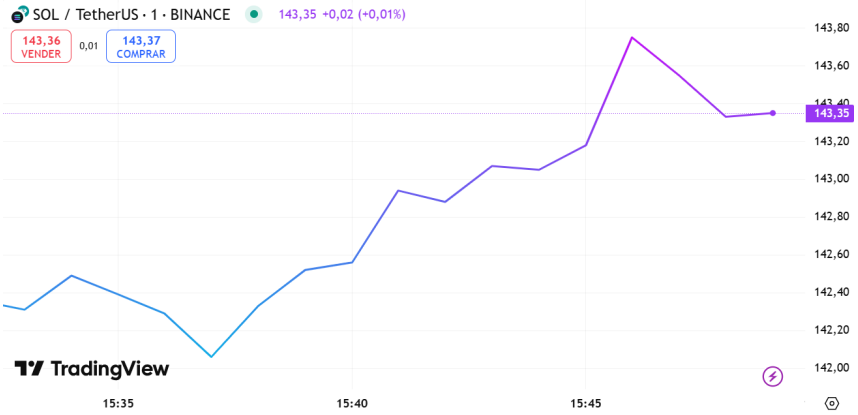

Figura 3. Cotización de Solana en Tiempo Real en `TradingView`.

10. Discusión de los resultados

La correcta implementación de la solución propuesta se corrobora comparando los valores de ambos gráficos, donde se reconoce que ambas cotizaciones muestran el mismo patrón y los precios son idénticos en su valor numérico.

Conclusión

Por último, se van a resumir los principales hallazgos y aprendizajes obtenidos durante la práctica, y la relevancia de la solución implementada.

11. Resumen del Proceso

Para el almacenamiento y visualización de los precios y volúmenes de cotización de Solana en tiempo real en la herramienta Kibana se han seguido los siguientes pasos.

1. Identificación del problema: Almacenar para su visualización el precio de cotización en tiempo real de Solana.
2. Diseño de la arquitectura e implementación: Establecimiento de un *websocket* entre el productor y la herramienta `TradingView` para obtener los datos en tiempo real. El productor transforma estos datos a formato JSON y los envía a un tópico de Apache Kafka. El consumidor se suscribe al tópico en cuestión y realiza un *poll* de mensajes que indexa en ElasticSearch. Los datos almacenados son visualizados en tiempo real mediante un *dashboard* en Kibana.
3. Pruebas y análisis: En Kibana se visualiza mediante un gráfico de líneas la evolución del precio de cierre de Solana en cada instante. El gráfico resultante se compara con la cotización mostrada en `TradingView`. La similitud en apariencia y coincidencia de los valores numéricos permiten corroborar el correcto funcionamiento de la solución.

12. Principales Logros

La solución propuesta implementa de manera exitosa un *pipeline* de datos para monitorizar y analizar precios de criptomonedas en tiempo real. El productor accede a los datos en tiempo real de la herramienta `TradingView`, los transforma a formato JSON y los envía a un tópico de Apache Kafka. Paralelamente se ha logrado desarrollar un consumidor capaz de conectarse a este tópico e indexar los mensajes en Elasticsearch para su posterior representación en Kibana. Para concluir se van a recorrer los logros más importantes alcanzados en base a los resultados que se han analizado en la sección anterior.

- Familiarización con ElasticSearch.
- Almacenamiento de las cotizaciones en tiempo real de Solana en ElasticSearch.
- Familiarización con Kibana.
- Visualización de las cotizaciones en tiempo real de Solana con Kibana.