A close-up of a logo

Description automatically generated

**Sprint 5. Historia de Usuario 6**

**Elena Conderana Medem y Sergio Cuenca Núñez**

**Tecnologías de Datos Masivos**

***Big Data Technology***

# **Índice**

[Índice 2](#_Toc194417526)

[Introducción 3](#_Toc194417527)

[1. Contexto 3](#_Toc194417528)

[2. Descripción del problema 3](#_Toc194417529)

[3. Objetivos 3](#_Toc194417530)

[4. Justificación 3](#_Toc194417531)

[Metodología 4](#_Toc194417532)

[5. Descripción del entorno de desarrollo 4](#_Toc194417533)

[6. Diseño de la solución 4](#_Toc194417534)

[7. Pruebas realizadas 7](#_Toc194417535)

[Resultados 8](#_Toc194417536)

[8. Descripción de los resultados 8](#_Toc194417537)

[9. Pantallazos de la Ejecución 8](#_Toc194417538)

[10. Discusión de los resultados 10](#_Toc194417539)

[Conclusión 10](#_Toc194417540)

[11. Resumen del Proceso 11](#_Toc194417541)

[12. Principales Logros 11](#_Toc194417542)

# **Introducción**

# **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 anteriores, y de procesamiento en tiempo real de las cotizaciones. El sprint actual se centra en adquirir los datos en tiempo real de la herramienta ` TradingView` mediante *topics* de Kafka, que permitan consumir dichos datos y posteriormente almacenarlos para su análisis y tratamiento

# **Descripción del problema**

El propósito del análisis técnico de criptomonedas consiste en anticipar con mayor probabilidad cambios y tendencias en las mismas. Este propósito requiere almacenar el precio en tiempo real de las cotizaciones para poder analizarlo e inferir tendencias futuras. Para ello es necesario desarrollar una solución que acceda y recopile en tiempo real el volumen y el precio de cada moneda.

# **Objetivos**

Los objetivos principales de esta práctica comprenden:

* Acceder en tiempo real a los volúmenes y precios de cotización de las criptomonedas en tiempo real.
* Familiarización con Kafka.
* Implementación de un productor y un consumidor de Kafka.

# **Justificación**

Para analizar en tiempo real las criptomonedas el primer paso consiste en conseguir adquirir dichos datos en ese mismo instante. La implementación de un consumidor y un productor dentro de un *topic* de Kafka permite realizar dicho acceso para posteriormente poder almacenar y trabajar con los datos.

# **Metodología**

En la siguiente sección se procede a describir en detalle cómo se ha realizado el acceso a los datos en tiempo real de la herramienta `TradingView` para Solana con la herramienta Kafka. Se abordarán cuestiones técnicas sobre el proceso de desarrollo y sobre las tecnologías utilizadas.

# **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 8 librerías:

* `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.
* `kafka`: para la interacción con el clúster de Apache Kafka, incluyendo la producción y el consumo de mensajes.

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.

# **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 escucha 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. Simultáneamente, para mantener la conexión activa el servidor y productor se envían mensajes *ping* mutuamente. 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.

**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 clúster de Apache Kafka y se suscribe al tópico gittba\_SOL. El consumidor tiene asociado el identificador de grupo gittba\_group\_sol y se asigna específicamente a la partición 0 del tópico, realizando un *polling* periódico cada 5000ms para recibir todos los mensajes disponibles en el tópico. Dichos mensajes son impresos por consola para verificar que se está obteniendo la información objetivo. El Código 2 muestra la implementación del consumidor de Apache Kafka descrita. A modo de síntesis la Figura 1 muestra de manera visual y simplificada el funcionamiento de la solución implementada.

**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.

Diagrama

El contenido generado por IA puede ser incorrecto.

Figura 1. Diagrama de Flujo de Solución Implementada.

# **Pruebas realizadas**

El correcto acceso a los datos se realiza comparando los valores de precio y de volumen obtenidos para Solana en distintos instantes temporales con los valores en esos mismos instantes ofrecidos por `TradingView`.

# **Resultados**

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

# **Descripción de los resultados**

Con la metodología anterior se consigue obtener los datos de precio, cambio, porcentaje de cambio y volumen en tiempo real para la criptomoneda Solana. El productor imprime por pantalla cada vez que envía un mensaje al tópico el contenido de este, para comprobar que se está enviando el contenido deseado. El consumidor a su vez imprime por consola la clave y el valor de los mensajes adquiridos para corroborar que se está accediendo a los mensajes deseados.

# **Pantallazos de la Ejecución**

La Figura 2 muestra el contenido de los mensajes enviados por el productor al tópico de Kafka. El consumidor obtiene dichos mensajes y los imprime por pantalla mostrando la clave y el valor de los mismos, como se ejemplifica en la Figura 3.

➜ sprint5 git:(main) python3 crypto\_real\_time\_producer.py

Conectado a wss://data.tradingview.com/socket.io/websocket

Error procesando mensaje: 'm'

Error procesando mensaje: Extra data: line 1 column 128 (char 127)

[[2025-04-01 15:38:56]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:38:56]', 'price': 127.09, 'change': 2.55, 'change\_percentage': 2.05, 'volume': 1672953.384}

[[2025-04-01 15:38:58]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:38:58]', 'price': 127.1, 'change': 2.56, 'change\_percentage': 2.06, 'volume': 1672992.425}

[[2025-04-01 15:39:01]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:01]', 'price': 'No disponible', 'change': 'No disponible', 'change\_percentage': 'No disponible', 'volume': 1673014.129}

[[2025-04-01 15:39:03]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:03]', 'price': 127.14, 'change': 2.6, 'change\_percentage': 2.09, 'volume': 1673064.823}

[[2025-04-01 15:39:05]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:05]', 'price': 127.15, 'change': 2.61, 'change\_percentage': 2.1, 'volume': 1673105.127}

[[2025-04-01 15:39:08]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:08]', 'price': 127.26, 'change': 2.72, 'change\_percentage': 2.18, 'volume': 1673247.115}

[[2025-04-01 15:39:10]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:10]', 'price': 127.27, 'change': 2.73, 'change\_percentage': 2.19, 'volume': 1673277.802}

[[2025-04-01 15:39:12]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:12]', 'price': 127.21, 'change': 2.67, 'change\_percentage': 2.14, 'volume': 1673313.719}

[[2025-04-01 15:39:14]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:14]', 'price': 'No disponible', 'change': 'No disponible', 'change\_percentage': 'No disponible', 'volume': 1673344.036}

[[2025-04-01 15:39:16]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:16]', 'price': 127.26, 'change': 2.72, 'change\_percentage': 2.18, 'volume': 1673386.176}

[[2025-04-01 15:39:19]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:19]', 'price': 127.22, 'change': 2.68, 'change\_percentage': 2.15, 'volume': 1673426.589}

[[2025-04-01 15:39:21]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:21]', 'price': 127.2, 'change': 2.66, 'change\_percentage': 2.14, 'volume': 1673478.531}

[[2025-04-01 15:39:23]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:23]', 'price': 127.17, 'change': 2.63, 'change\_percentage': 2.11, 'volume': 1673517.234}

[[2025-04-01 15:39:26]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:26]', 'price': 127.18, 'change': 2.64, 'change\_percentage': 2.12, 'volume': 1673518.631}

[[2025-04-01 15:39:29]] Enviando a Kafka: {'timestamp': '[2025-04-01 15:39:29]', 'price': 127.14, 'change': 2.6, 'change\_percentage': 2.09, 'volume': 1673534.922}

Figura 2. Envío a Kafka de Datos en Tiempo Real por Productor

TopicPartition: TopicPartition(topic='gittba\_SOL', partition=0)

key: solana

value: {"timestamp": "[2025-03-25 16:42:51]", "price": 145.07, "change": 4.06, "change\_percentage": 2.88, "volume": 2502685.271}

offset: 7

timestamp: 1742917371648

key: solana

value: {"timestamp": "[2025-03-25 16:42:54]", "price": "No disponible", "change": "No disponible", "change\_percentage": "No disponible", "volume": 2502795.097}

offset: 8

timestamp: 1742917374578

key: solana

value: {"timestamp": "[2025-03-25 16:42:56]", "price": "No disponible", "change": "No disponible", "change\_percentage": "No disponible", "volume": 2502800.978}

offset: 9

timestamp: 1742917376600

key: solana

value: {"timestamp": "[2025-04-01 15:38:56]", "price": 127.09, "change": 2.55, "change\_percentage": 2.05, "volume": 1672953.384}

offset: 10

timestamp: 1743514736583

key: solana

value: {"timestamp": "[2025-04-01 15:38:58]", "price": 127.1, "change": 2.56, "change\_percentage": 2.06, "volume": 1672992.425}

offset: 11

timestamp: 1743514738642

key: solana

value: {"timestamp": "[2025-04-01 15:39:01]", "price": "No disponible", "change": "No disponible", "change\_percentage": "No disponible", "volume": 1673014.129}

offset: 12

timestamp: 1743514741495

key: solana

value: {"timestamp": "[2025-04-01 15:39:03]", "price": 127.14, "change": 2.6, "change\_percentage": 2.09, "volume": 1673064.823}

offset: 13

timestamp: 1743514743339

key: solana

value: {"timestamp": "[2025-04-01 15:39:05]", "price": 127.15, "change": 2.61, "change\_percentage": 2.1, "volume": 1673105.127}

offset: 14

timestamp: 1743514745597

key: solana

value: {"timestamp": "[2025-04-01 15:39:08]", "price": 127.26, "change": 2.72, "change\_percentage": 2.18, "volume": 1673247.115}

offset: 15

timestamp: 1743514748309

key: solana

value: {"timestamp": "[2025-04-01 15:39:10]", "price": 127.27, "change": 2.73, "change\_percentage": 2.19, "volume": 1673277.802}

offset: 16

timestamp: 1743514750340

key: solana

value: {"timestamp": "[2025-04-01 15:39:12]", "price": 127.21, "change": 2.67, "change\_percentage": 2.14, "volume": 1673313.719}

offset: 17

timestamp: 1743514752345

key: solana

value: {"timestamp": "[2025-04-01 15:39:14]", "price": "No disponible", "change": "No disponible", "change\_percentage": "No disponible", "volume": 1673344.036}

offset: 18

timestamp: 1743514754392

key: solana

value: {"timestamp": "[2025-04-01 15:39:16]", "price": 127.26, "change": 2.72, "change\_percentage": 2.18, "volume": 1673386.176}

offset: 19

timestamp: 1743514756342

key: solana

value: {"timestamp": "[2025-04-01 15:39:19]", "price": 127.22, "change": 2.68, "change\_percentage": 2.15, "volume": 1673426.589}

offset: 20

timestamp: 1743514759410

key: solana

value: {"timestamp": "[2025-04-01 15:39:21]", "price": 127.2, "change": 2.66, "change\_percentage": 2.14, "volume": 1673478.531}

offset: 21

timestamp: 1743514761760

key: solana

value: {"timestamp": "[2025-04-01 15:39:23]", "price": 127.17, "change": 2.63, "change\_percentage": 2.11, "volume": 1673517.234}

offset: 22

timestamp: 1743514763919

key: solana

value: {"timestamp": "[2025-04-01 15:39:26]", "price": 127.18, "change": 2.64, "change\_percentage": 2.12, "volume": 1673518.631}

offset: 23

timestamp: 1743514766374

Figura 3. Obtención de Mensajes del Tópico por Parte del Consumidor

# **Discusión de los resultados**

La correcta implementación de la solución propuesta se puede llevar a cabo comparando los valores obtenidos con los que se aprecian en la herramienta `TradingView`. Si bien no se aprecia el desglose por segundos de las cotizaciones se puede verificar que los valores se encuentran dentro de los límites de la vela japonesa.

# **Conclusión**

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

# **Resumen del Proceso**

Para el acceso a los precios y volúmenes de cotización de Solana en tiempo real se han seguido los siguientes pasos.

1. Identificación del problema: Acceder en tiempo real a la herramienta `TradingView` para obtener el precio y volumen de cotización 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 cada 5 segundos para obtener los mensajes con los datos en tiempo real obtenidos y procesados por el productor.
3. Pruebas y análisis: El consumidor imprime por pantalla la clave y el valor de cada mensaje obtenido del tópico. Los valores de precio y volumen se pueden comparar con los equivalentes para esos instantes temporales de la herramienta `TradingView` que permiten corroborar el correcto funcionamiento de la solución.

# **Principales Logros**

La solución propuesta implementa de manera exitosa un productor que 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. También se ha logrado desarrollar un consumidor capaz de conectarse a este tópico y acceder a los mensajes publicados por el productor. 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 Apache Kafka.
* Obtención en tiempo real de los datos de cotización de `TradingView`.
* Implementación de un productor y de un consumidor que se conectan a través de un tópico exitosamente.