



# COMILLAS

UNIVERSIDAD PONTIFICIA

ICAI

ICADE

CIHS

# Sprint 5

## Grupo 2

Isabel V. Morell Maudes  
Teresa X. Garvía Gallego  
María Carreño Nin de Cardona  
Raquel Fernández Esquinas

Tecnologías de Procesamiento Big Data  
3º Grado en Ingeniería Matemática e Inteligencia Artificial

# Índice

INTRODUCCIÓN .....	3
METODOLOGÍA.....	4
RESULTADOS.....	5
CONCLUSIÓN .....	7

# Introducción

Este sprint supone la quinta práctica del proyecto final de la asignatura Tecnologías de Procesamiento Big Data sobre las criptomonedas. En esta fase, nos enfocaremos en la adquisición de datos en tiempo real provenientes de Trading View, lo que permitirá mejorar el acceso a la información utilizada en análisis posteriores.

El objetivo principal de este sprint es garantizar que podamos acceder, en tiempo real, a los datos de la criptomoneda que nos ha sido asignada (ETH). Para lograr esto, se utilizará una combinación de dos scripts. El primero, el script Python “crypto\_tradingview\_real\_time.py”, lo hemos ejecutado en una instancia de Amazon EC2. Posteriormente, la información capturada se publicará en el topic de Kafka correspondiente a cada criptomoneda en formato JSON, empleando el código del segundo script “producer.py” proporcionado en el archivo kafka\_2.13-3.6.0.zip.

Este proceso nos permite garantizar que tenemos una estructura de datos eficiente y actualizada, asegurando la disponibilidad de información en tiempo real para su posterior procesamiento y análisis. Al utilizar Apache Kafka como sistema de mensajería, se garantiza una transmisión segura y escalable de los datos, optimizando la recolección.

Con esta implementación, se busca fortalecer la capacidad de gestionar grandes volúmenes de datos de manera eficiente, asegurando que las fuentes de información sean precisas y actualizadas.

Toda la información generada en este sprint, incluyendo los archivos y datos relacionados, se encuentra disponible en la rama de desarrollo del repositorio de GitHub: [https://github.com/tgarviagallego/Proyecto\\_BigData.git](https://github.com/tgarviagallego/Proyecto_BigData.git)

# Metodología

Para la realización de este sprint, hemos desarrollado un proceso de adquisición de datos en tiempo real mediante el uso de Amazon EC2 y Apache Kafka.

Cada criptomoneda tiene asignado un topic específico en Kafka, lo que permite una mejor organización y distribución de los datos. Los mensajes enviados contienen información clave, como el precio actual y la fecha.

Inicialmente, hemos combinado el código presente tanto en “crypto\_tradingview\_real\_time.py” como en “producer.py”. De esta forma adquirimos los datos de Trading View y gracias a una conexión web y los publicamos en nuestro topic de ETH en formato JSON para que posteriormente pueda ser consumido por una herramienta de visualización de datos.

Para la conexión web hemos utilizado la librería de Python de websocket, que se ejecuta de manera continua para obtener los datos en tiempo real. Una vez recibimos los datos como cliente, los procesamos y obtenemos el precio de la criptomoneda en ese momento. Para publicar los datos en el topic de Kafka, primero cambiamos el formato del mensaje a un JSON y después publicamos el mensaje en el topic utilizando la librería subprocess, que se encontraba en el código “producer.py”, facilitado por los profesores.

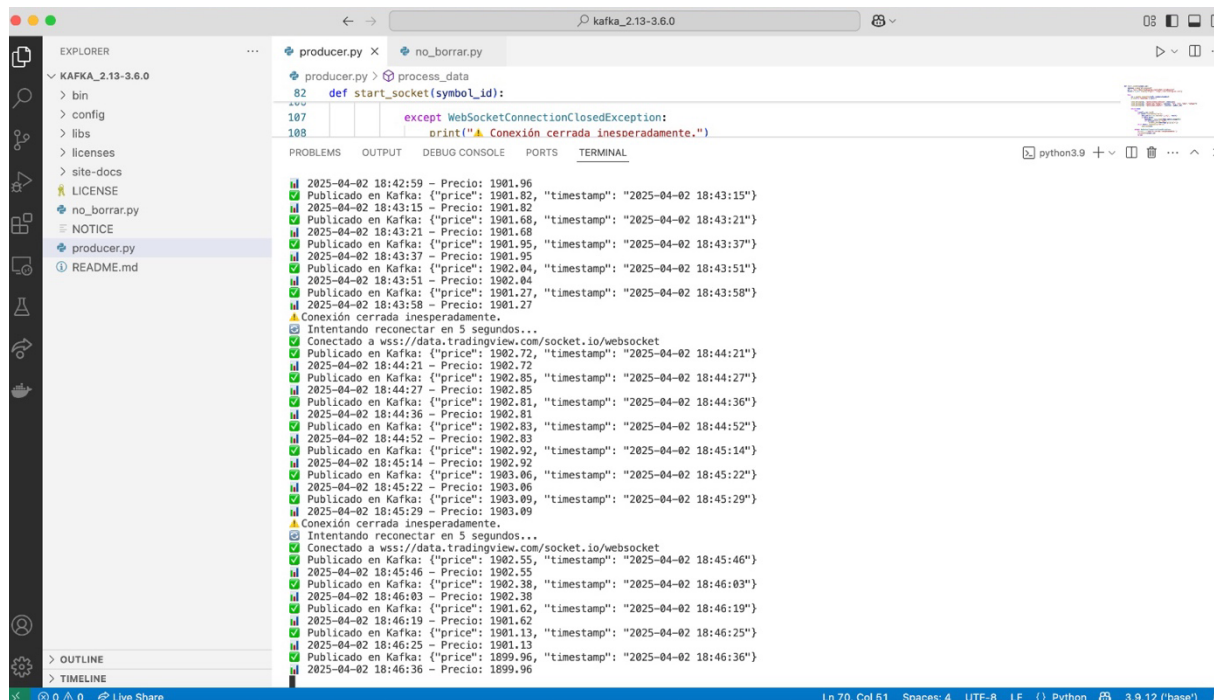
Para comprobar el correcto funcionamiento de la publicación de datos en tiempo real nos hemos conectado a través de una instancia de EC2 como consumidores del topic. Para ello hemos comenzado creando una instancia de EC2 y conectándonos a ella. Tras introducir las credenciales hemos ejecutado el siguiente comando para crear el consumidor de ese topic:

```
“bin/kafka-console-consumer.sh --bootstrap-server b-1-public.testkafka3.l7t9mi.c2.kafka.eu-south-2.amazonaws.com:9198,b-2-public.testkafka3.l7t9mi.c2.kafka.eu-south-2.amazonaws.com:9198 --consumer.config config/client.properties --topic imat3a_ETH --property print.key=true --property print.value=true --property print.partition=true --property print.offset=true --property print.timestamp=true --consumer-property group.id=imat3a_group02”
```

Esta implementación permite mejorar la eficiencia del procesamiento de datos, asegurando un flujo de información continuo y en tiempo real que servirá como base para la toma de decisiones en el ámbito del trading y la inversión en criptomonedas.

# Resultados

Tras crear el productor y el consumidor en Kafka, dimos comienzo a la comunicación entre ambos. En primer lugar, ejecutamos el fichero “producer.py”, el cual recoge los datos de Trading View en tiempo real. Podemos ver en las imágenes adjuntas el resultado de esta ejecución, donde aparece para cada timestamp el precio que ha recogido de la página Trading View. Tuvimos que añadir las credenciales para poder ejecutar el script correctamente.



```

def start_socket(symbol_id):
    except WebSocketConnectionClosedException:
        print("⚠️ Conexión cerrada inesperadamente.")

2025-04-02 18:42:59 - Precio: 1901.96
✓ Publicado en Kafka: {"price": 1901.82, "timestamp": "2025-04-02 18:43:15"}
2025-04-02 18:43:15 - Precio: 1901.82
✓ Publicado en Kafka: {"price": 1901.68, "timestamp": "2025-04-02 18:43:21"}
2025-04-02 18:43:21 - Precio: 1901.68
✓ Publicado en Kafka: {"price": 1901.95, "timestamp": "2025-04-02 18:43:37"}
2025-04-02 18:43:37 - Precio: 1901.95
✓ Publicado en Kafka: {"price": 1902.84, "timestamp": "2025-04-02 18:43:51"}
2025-04-02 18:43:51 - Precio: 1902.04
✓ Publicado en Kafka: {"price": 1901.27, "timestamp": "2025-04-02 18:43:58"}
2025-04-02 18:43:58 - Precio: 1901.27
⚠️ Conexión cerrada inesperadamente.
⚠️ Intentando reconectar en 5 segundos...
✓ Conectado a wss://data.tradingview.com/socket.io/websocket
✓ Publicado en Kafka: {"price": 1902.72, "timestamp": "2025-04-02 18:44:21"}
2025-04-02 18:44:21 - Precio: 1902.72
✓ Publicado en Kafka: {"price": 1902.85, "timestamp": "2025-04-02 18:44:27"}
2025-04-02 18:44:27 - Precio: 1902.85
✓ Publicado en Kafka: {"price": 1902.81, "timestamp": "2025-04-02 18:44:36"}
2025-04-02 18:44:36 - Precio: 1902.81
✓ Publicado en Kafka: {"price": 1902.83, "timestamp": "2025-04-02 18:44:52"}
2025-04-02 18:44:52 - Precio: 1902.83
✓ Publicado en Kafka: {"price": 1902.92, "timestamp": "2025-04-02 18:45:14"}
2025-04-02 18:45:14 - Precio: 1902.92
✓ Publicado en Kafka: {"price": 1903.06, "timestamp": "2025-04-02 18:45:22"}
2025-04-02 18:45:22 - Precio: 1903.06
✓ Publicado en Kafka: {"price": 1903.09, "timestamp": "2025-04-02 18:45:29"}
2025-04-02 18:45:29 - Precio: 1903.09
⚠️ Conexión cerrada inesperadamente.
⚠️ Intentando reconectar en 5 segundos...
✓ Conectado a wss://data.tradingview.com/socket.io/websocket
✓ Publicado en Kafka: {"price": 1902.55, "timestamp": "2025-04-02 18:45:46"}
2025-04-02 18:45:46 - Precio: 1902.55
✓ Publicado en Kafka: {"price": 1902.38, "timestamp": "2025-04-02 18:46:03"}
2025-04-02 18:46:03 - Precio: 1902.38
✓ Publicado en Kafka: {"price": 1901.62, "timestamp": "2025-04-02 18:46:19"}
2025-04-02 18:46:19 - Precio: 1901.62
✓ Publicado en Kafka: {"price": 1901.13, "timestamp": "2025-04-02 18:46:25"}
2025-04-02 18:46:25 - Precio: 1901.13
✓ Publicado en Kafka: {"price": 1899.96, "timestamp": "2025-04-02 18:46:36"}
2025-04-02 18:46:36 - Precio: 1899.96
  
```

```

✓ Conectado a wss://data.tradingview.com/socket.io/websocket
✓ Publicado en Kafka: {"price": 1902.72, "timestamp": "2025-04-02 18:44:21"}
2025-04-02 18:44:21 - Precio: 1902.72
✓ Publicado en Kafka: {"price": 1902.85, "timestamp": "2025-04-02 18:44:27"}
2025-04-02 18:44:27 - Precio: 1902.85
✓ Publicado en Kafka: {"price": 1902.81, "timestamp": "2025-04-02 18:44:36"}
2025-04-02 18:44:36 - Precio: 1902.81
✓ Publicado en Kafka: {"price": 1902.83, "timestamp": "2025-04-02 18:44:52"}
2025-04-02 18:44:52 - Precio: 1902.83
✓ Publicado en Kafka: {"price": 1902.92, "timestamp": "2025-04-02 18:45:14"}
2025-04-02 18:45:14 - Precio: 1902.92
✓ Publicado en Kafka: {"price": 1903.06, "timestamp": "2025-04-02 18:45:22"}
2025-04-02 18:45:22 - Precio: 1903.06
✓ Publicado en Kafka: {"price": 1903.09, "timestamp": "2025-04-02 18:45:29"}
2025-04-02 18:45:29 - Precio: 1903.09
  
```

En nuestro caso hemos ejecutado el script desde nuestro ordenador de forma local en vez de desde AWS.

Posteriormente creamos una instancia de AWS donde vamos a alojar el script “consumer.py”. Siguiendo las instrucciones dadas en clase, conseguimos obtener un consumer que quedaba a la escucha en nuestro topic: “imat3a\_ETH”.

Ejecutando el “producer.py” y quedando el consumidor a la espera, vimos cómo iba recibiendo los datos que el primero publicaba al topic.

```

CreateTime:1743596843069    Partition:0    Offset:37    A    {"price": 1862.77, "timestamp": "2025-04-02 14:27:17"}
CreateTime:1743596853734    Partition:0    Offset:38    A    {"price": 1863.62, "timestamp": "2025-04-02 14:27:27"}
CreateTime:1743596861212    Partition:0    Offset:39    A    {"price": 1862.07, "timestamp": "2025-04-02 14:27:35"}
CreateTime:1743612173017    Partition:0    Offset:40    A    {"price": 1901.78, "timestamp": "2025-04-02 18:42:46"}
CreateTime:1743612179011    Partition:0    Offset:41    A    {"price": 1902.77, "timestamp": "2025-04-02 18:42:53"}
CreateTime:1743612185437    Partition:0    Offset:42    A    {"price": 1901.96, "timestamp": "2025-04-02 18:42:59"}
CreateTime:1743612201315    Partition:0    Offset:43    A    {"price": 1901.82, "timestamp": "2025-04-02 18:43:15"}
CreateTime:1743612208985    Partition:0    Offset:44    A    {"price": 1901.68, "timestamp": "2025-04-02 18:43:21"}
CreateTime:1743612223138    Partition:0    Offset:45    A    {"price": 1901.95, "timestamp": "2025-04-02 18:43:37"}
CreateTime:1743612237691    Partition:0    Offset:46    A    {"price": 1902.04, "timestamp": "2025-04-02 18:43:51"}
CreateTime:1743612244277    Partition:0    Offset:47    A    {"price": 1901.27, "timestamp": "2025-04-02 18:43:58"}
CreateTime:1743612267112    Partition:0    Offset:48    A    {"price": 1902.72, "timestamp": "2025-04-02 18:44:21"}
CreateTime:1743612273378    Partition:0    Offset:49    A    {"price": 1902.85, "timestamp": "2025-04-02 18:44:27"}
CreateTime:1743612282230    Partition:0    Offset:50    A    {"price": 1902.81, "timestamp": "2025-04-02 18:44:36"}
CreateTime:1743612299293    Partition:0    Offset:51    A    {"price": 1902.83, "timestamp": "2025-04-02 18:44:52"}
CreateTime:1743612320607    Partition:0    Offset:52    A    {"price": 1902.92, "timestamp": "2025-04-02 18:45:14"}
CreateTime:1743612328622    Partition:0    Offset:53    A    {"price": 1903.06, "timestamp": "2025-04-02 18:45:22"}
CreateTime:1743612334538    Partition:0    Offset:54    A    {"price": 1903.09, "timestamp": "2025-04-02 18:45:29"}
CreateTime:1743612352363    Partition:0    Offset:55    A    {"price": 1902.55, "timestamp": "2025-04-02 18:45:46"}
CreateTime:1743612373493    Partition:0    Offset:56    A    {"price": 1902.38, "timestamp": "2025-04-02 18:46:03"}
CreateTime:1743612384836    Partition:0    Offset:57    A    {"price": 1901.62, "timestamp": "2025-04-02 18:46:19"}
CreateTime:1743612390317    Partition:0    Offset:58    A    {"price": 1901.13, "timestamp": "2025-04-02 18:46:25"}
CreateTime:1743612402879    Partition:0    Offset:59    A    {"price": 1899.96, "timestamp": "2025-04-02 18:46:36"}

```

Como se puede observar en la imagen superior, conseguimos establecer una conexión entre productor y consumidor.

## Conclusión

La adquisición de datos en tiempo real mediante Amazon EC2 y Apache Kafka ha permitido mejorar la disponibilidad y precisión de la información utilizada en el análisis de las criptomonedas. La creación e implementación de este flujo de datos, nos ha permitido optimizar la captura y transmisión de la información para garantizar que los datos sean accesibles constantemente para realizar análisis posteriores.

Gracias a este sprint, hemos sido capaces de realizar una estructura que nos permite obtener los datos de nuestra criptomoneda en tiempo real, de manera eficiente y segura. Esto nos permite tener un acceso muy rápido a los datos disponibles, lo que nos puede facilitar el posterior análisis, como por ejemplo identificar patrones en el mercado.

A pesar de los avances logrados, es importante continuar optimizando los procesos de adquisición de datos, garantizando la integridad de los datos en todo momento, lo cual es un punto clave para asegurar que nuestros datos son fiables y que no los estamos modificando.

En conclusión, la implementación de este flujo de trabajo ha mejorado nuestra capacidad para gestionar grandes volúmenes de datos y ha mejorado nuestra velocidad de acceso a los datos.