



COMILLAS

UNIVERSIDAD PONTIFICIA

ICAI

ICADE

CIHS

Sprint 6

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 | 8 |

Introducción

En el ámbito del estudio del mercado de criptomonedas, en este sprint nos vamos a enfocar en la recepción y el envío del precio de la criptomoneda ETH en cada instante de tiempo.

Por ello, debemos abordar una serie de problemas que abarcan desde la lectura de los datos de TradingView hasta el procesamiento de los mismos y su envío a través de un topic de Kafka para visualizarlos en Grafana.

Los objetivos principales de este sprint son, por tanto, establecer una comunicación entre nodos, donde el nodo intermedio actuará tanto consumidor como productor de los datos.

De esta forma podremos visualizar los datos y ver su evolución en tiempo real para poder realizar un estudio detallado sobre, por ejemplo, la tendencia del precio y la fortaleza del mercado.

Toda la documentación generada en este sprint se encuentra disponible en la rama de desarrollo del repositorio de GitHub: https://github.com/tgarviagallego/Proyecto_BigData.git

Metodología

Para el desarrollo de este sprint hemos utilizado tecnologías como Grafana e instancias de Amazon EC2 para la ejecución del código tanto del consumidor como del productor. El productor consume datos de la página web de TradingView y lo filtra para quedarse únicamente con el precio y el timestamp. El código se ha desarrollado en Python, usando librerías como boto3 para establecer la conexión con AWS y botocore, que permite configurar aspectos del cliente de AWS.

Para poder ejecutar nuestro productor de datos de TradingView hemos utilizado el ordenador local, mientras que para ejecutar el consumidor de esos datos y productor de los datos filtrados por aquellos que nos interesan hemos utilizado una instancia de EC2 basada en una imagen de Linux de Amazon.

Para la visualización de los datos (precio) en Grafana, hemos creado un dashboard, hemos creado una visualización y nos hemos conectado a la base de datos “CryptoIcaiDatabase”, a la tabla “CryptoMonedas” y hemos seleccionado la criptomoneda que correspondía a nuestro grupo, ETH.

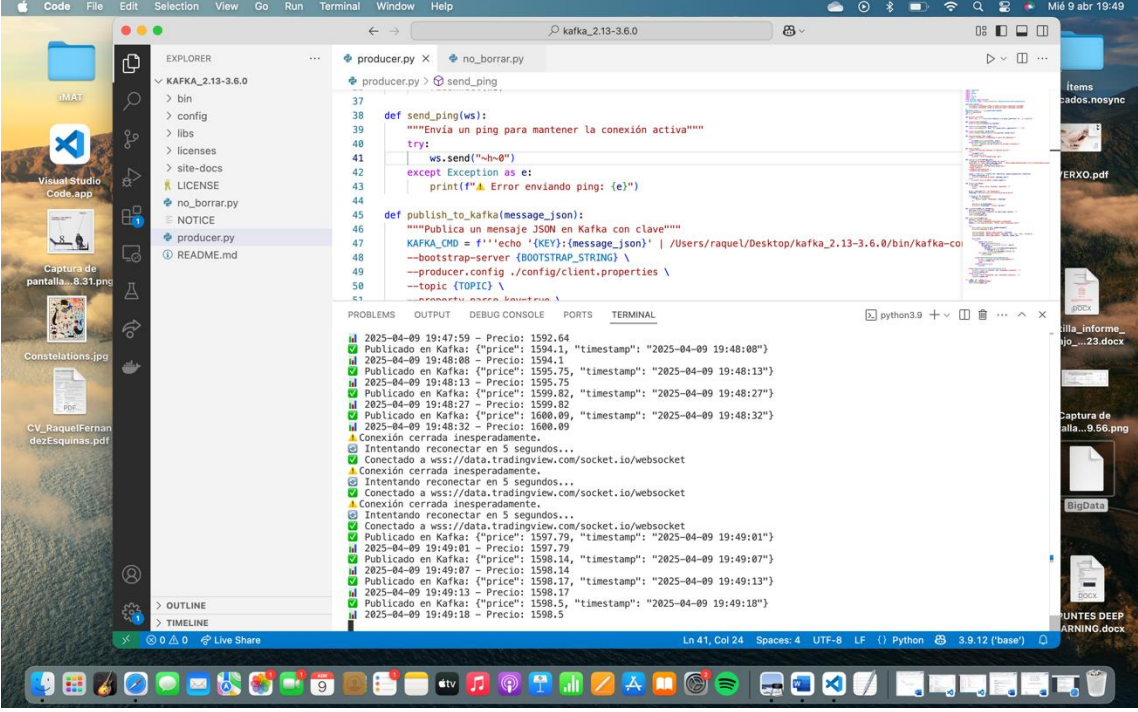
Nuestro script de consumidor y productor se basa en el código proporcionado por el profesor “timestream.py”, pero añadiendo la conexión a la web de TradingView.

Este archivo utiliza las librerías de boto3 y botocore.config para crear un cliente de timestream que escriba los datos en esta base de datos. También utilizamos la librería de python subprocess, en concreto, la clase Popen para consumir los datos publicados en kafka.

Para verificar el funcionamiento del sistema hemos ejecutado el comando del consumidor del tema de visualización en streaming para comprobar la correcta producción de datos desde el productor conectado a TradingView desde una instancia EC2. Por último, y tras comprobar que este intercambio de datos se realizaba correctamente, utilizamos Grafana para comprobar la correcta producción del script del consumer + producer. Una vez que los datos aparecen en el dashboard, damos por concluido el sprint.

Resultados

Para probar el correcto funcionamiento de la práctica hemos ejecutado en un ordenador en local el productor de datos de la moneda ETH en su topic correspondiente como se muestra en la siguiente imagen:



The screenshot shows a VS Code editor with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The code editor shows a Python script named `producer.py` with the following content:

```

37
38 def send_ping(ws):
39     """Envía un ping para mantener la conexión activa"""
40     try:
41         ws.send("h-0")
42     except Exception as e:
43         print(f"Error enviando ping: {e}")
44
45 def publish_to_kafka(message_json):
46     """Publica un mensaje JSON en Kafka con clave"""
47     KAFKA_CMD = f'''echo '{(KEY):(message_json)}' | /Users/raquel/Desktop/kafka_2.13-3.6.0/bin/kafka-co
48     --bootstrap-server {BOOTSTRAP_STRING} \
49     --producer.config ./config/client.properties \
50     --topic {TOPIC} \
51     --property acks=least_1

```

The terminal output shows the following logs:

```

2025-04-09 19:47:59 - Precio: 1592.64
✓ Publicado en Kafka: {"price": 1594.1, "timestamp": "2025-04-09 19:48:08"}
✓ Publicado en Kafka: {"price": 1594.1, "timestamp": "2025-04-09 19:48:08"}
✓ Publicado en Kafka: {"price": 1595.75, "timestamp": "2025-04-09 19:48:13"}
✓ Publicado en Kafka: {"price": 1595.75, "timestamp": "2025-04-09 19:48:13"}
✓ Publicado en Kafka: {"price": 1599.82, "timestamp": "2025-04-09 19:48:27"}
✓ Publicado en Kafka: {"price": 1599.82, "timestamp": "2025-04-09 19:48:27"}
✓ Publicado en Kafka: {"price": 1600.09, "timestamp": "2025-04-09 19:48:32"}
✓ Publicado en Kafka: {"price": 1600.09, "timestamp": "2025-04-09 19:48:32"}
✓ Publicado en Kafka: {"price": 1597.79, "timestamp": "2025-04-09 19:49:01"}
✓ Publicado en Kafka: {"price": 1597.79, "timestamp": "2025-04-09 19:49:01"}
✓ Publicado en Kafka: {"price": 1598.14, "timestamp": "2025-04-09 19:49:07"}
✓ Publicado en Kafka: {"price": 1598.14, "timestamp": "2025-04-09 19:49:07"}
✓ Publicado en Kafka: {"price": 1598.17, "timestamp": "2025-04-09 19:49:13"}
✓ Publicado en Kafka: {"price": 1598.17, "timestamp": "2025-04-09 19:49:13"}
✓ Publicado en Kafka: {"price": 1598.5, "timestamp": "2025-04-09 19:49:18"}
✓ Publicado en Kafka: {"price": 1598.5, "timestamp": "2025-04-09 19:49:18"}

```

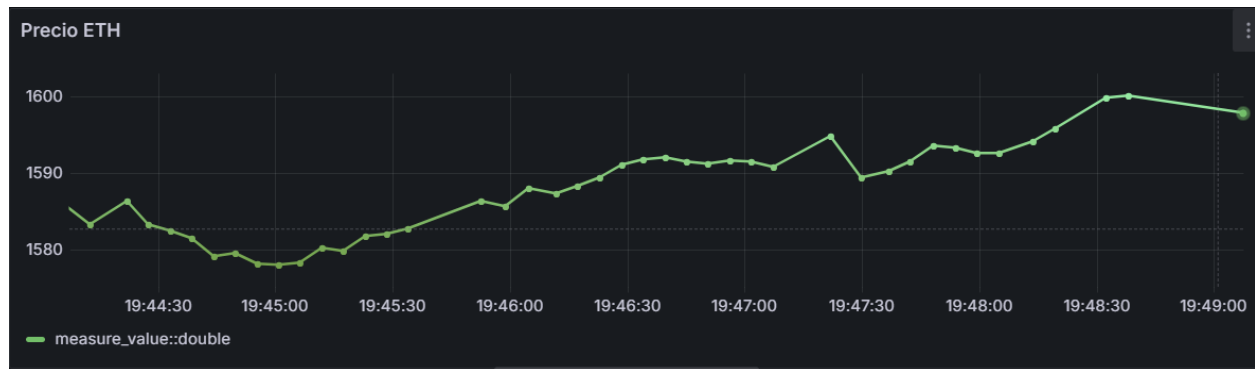
Como se puede observar, se están publicando datos entorno a los 1580-1600. Estos datos están siendo consumidos gracias a un fichero llamado `timestream.py` que ejecutamos en una instancia de `ec2`. Como se muestra en la imagen siguiente, el consumidor está recibiendo sin problema los datos y los está guardando en `timestream`:

```
Writing records
WriteRecords Status: [200]
1592.6
Writing records
WriteRecords Status: [200]
1592.64
Writing records
WriteRecords Status: [200]
1594.1
Writing records
WriteRecords Status: [200]
1595.75
Writing records
WriteRecords Status: [200]
1599.82
Writing records
WriteRecords Status: [200]
1600.09
Writing records
WriteRecords Status: [200]
1597.79
Writing records
WriteRecords Status: [200]
1598.14
Writing records
WriteRecords Status: [200]
```

Por último, utilizamos la consulta siguiente para ver en el dashboard creado los datos de ETH extraídos de trading view:

```
SELECT * FROM $__database.$__table
WHERE mesasure_name='ETH'
```

La imagen final de Grafana es la siguiente:



Conclusión

En este Sprint hemos conseguido establecer un flujo de datos completo en tiempo real, desde la obtención del precio de la criptomoneda ETH en TradingView hasta su visualización en Grafana mediante el uso de Kafka como el intermediario de mensajes.

En este Sprint hemos tenido que incorporar múltiples herramientas e integrar varias tecnologías como Python, Amazon EC2, AWS Timestream y Grafana; esto nos ha permitido afianzar más los conocimientos y adquirir práctica en el uso de las herramientas de Big Data, además de ser capaces de comprender el ciclo de vida de los datos en un sistema distribuido.

Al implementar un sistema que actúa tanto como consumidor como productor de los datos, hemos podido filtrar la información relevante obtenida de la página de trading view data y enfocarnos en su análisis visual a través de un dashboard personalizado de Grafana. Posteriormente esto se podría usar para estudios más avanzados sobre la evolución del mercado de criptomonedas.

En conclusión, este sprint ha sido clave para consolidar conocimientos en la manipulación de datos en tiempo real, fortalecer habilidades técnicas en entornos cloud, y sentar las bases para futuras fases del proyecto orientadas a un análisis más profundo del comportamiento del mercado.