**Paos a paso para implementar un Sistema de Procesamiento en Tiempo Real con Kafka y Dask**

A continuación, se detalla, paso a paso y sin enredos, cómo montar un **sistema de procesamiento en tiempo real** usando **Kafka y Dask**, con datos traídos desde la **API de Alpha Vantage**. Todo está explicado de manera clara pa' que cualquier persona pueda seguirlo sin enredos.

**Receta**

**1. Instalación de los Herramientas Necesarias**

Antes de ensillar el caballo, hay que asegurarse de tener lo necesario pa' que todo marche bien.

**1.1 Instalar Docker y Docker Compose**

Kafka trabaja mejor dentro de **Docker**, así que hay que instalarlo primero.

**Descargar Docker desde aquí:**  
 <https://www.docker.com/get-started/>

Una vez instalado, hay que verificar que funcione con este comando en **PowerShell**:

docker --version

Si devuelve algo como Docker version X.X.X, significa que todo está en orden.

**1.2 Instalar Python y las Librerías Necesarias**

Se necesita **Python 3.8 o superior**. Para comprobar si ya está instalado, se usa este comando:

python --version

Si no aparece la versión, se debe descargar e instalar desde aquí:  
 <https://www.python.org/downloads/>

Luego, hay que instalar las herramientas necesarias con este comando:

pip install kafka-python pandas dask alpha\_vantage

**Esto instalará:**  
 **kafka-python** → Para interactuar con Kafka.  
 **pandas** → Para manejar datos en tablas.  
 **dask** → Para procesamiento en paralelo.  
 **alpha\_vantage** → Para obtener datos de la bolsa de valores.

**2. Configurar Kafka con Docker**

Kafka se va a correr dentro de Docker pa' que todo sea más fácil.

**2.1 Crear un Archivo docker-compose.yml**

En la carpeta de trabajo (Big\_Data3), se debe crear un archivo llamado **docker-compose.yml** con este contenido:

version: '3'

services:

zookeeper:

image: wurstmeister/zookeeper

ports:

- "2181:2181"

kafka:

image: wurstmeister/kafka

ports:

- "9092:9092"

environment:

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://localhost:9092

KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

**2.2 Iniciar Kafka en Docker**

En la terminal, dentro de la carpeta del proyecto, se ejecuta:

docker-compose up -d

Esto descargará y pondrá a trabajar **Kafka** y **Zookeeper**.

Para asegurarse de que todo está corriendo, se usa:

docker ps

Si aparecen los contenedores kafka y zookeeper, todo está **listo y funcionando**.

**3. Crear el Productor (producer2.py)**

Este script se encargará de traer datos desde **Alpha Vantage** y enviarlos a **Kafka**.

**Crear un archivo llamado produce2r.py** y copiar este código:

from kafka import KafkaProducer

import json

import time

from alpha\_vantage.timeseries import TimeSeries

# Configuración de Alpha Vantage

API\_KEY = "TU\_API\_KEY" # 🔴 Reemplazar con una clave real

SYMBOL = "AAPL"

INTERVAL = "1min"

# Configuración de Kafka

TOPIC = "stock\_data"

KAFKA\_SERVER = "localhost:9092"

# Inicializar el productor de Kafka

producer2 = KafkaProducer(

bootstrap\_servers=KAFKA\_SERVER,

value\_serializer=lambda v: json.dumps(v).encode("utf-8")

)

# Función para obtener datos

def fetch\_stock\_data():

ts = TimeSeries(key=API\_KEY, output\_format="json")

data, meta = ts.get\_intraday(symbol=SYMBOL, interval=INTERVAL, outputsize="compact")

latest\_time = list(data.keys())[0]

stock\_info = {"time": latest\_time, "price": data[latest\_time]["1. open"]}

return stock\_info

# Enviar datos a Kafka cada 10 segundos

while True:

stock\_data = fetch\_stock\_data()

producer.send(TOPIC, stock\_data)

print(f"Datos enviados: {stock\_data}")

time.sleep(10)

**Explicación:**

* Se conecta a **Alpha Vantage** para obtener datos de la bolsa.
* Envía los datos a Kafka cada **10 segundos** en el tópico **"stock\_data"**.

**4. Crear el Consumidor (consumer2.py)**

Este script se encargará de recibir los datos desde **Kafka** y guardarlos en un archivo CSV.

**Crear un archivo llamado consumer2.py** y copiar este código:

from kafka import KafkaConsumer

import json

import pandas as pd

# Configuración de Kafka

TOPIC = "stock\_data"

KAFKA\_SERVER = "localhost:9092"

# Inicializar el consumidor de Kafka

consumer = KafkaConsumer(

TOPIC,

bootstrap\_servers=KAFKA\_SERVER,

auto\_offset\_reset='earliest',

enable\_auto\_commit=True,

group\_id='stock-data-group',

value\_deserializer=lambda x: json.loads(x.decode('utf-8'))

)

print("🟢 Esperando datos...")

# Lista para almacenar los datos recibidos

data\_list = []

for message in consumer:

data = message.value

print(f"Datos recibidos: {data}")

# Agregar el mensaje a la lista

data\_list.append(data)

# Guardar los datos en un archivo CSV

df = pd.DataFrame(data\_list)

df.to\_csv("processed\_stock\_data.csv", index=False)

**Explicación:**

* Se conecta a **Kafka** y espera datos en el tópico **"stock\_data"**.
* Guarda los datos en el archivo **processed\_stock\_data.csv**.

**5. Procesar los Datos con Dask**

Ahora se usará **Dask** para procesar los datos en paralelo.

**Crear un archivo llamado process\_dask.py** y copiar este código:

import dask.dataframe as dd

# Cargar los datos con Dask

df = dd.read\_csv("processed\_stock\_data.csv")

# Mostrar estadísticas básicas

print(df.describe().compute())

**Explicación:**

* Carga el archivo CSV usando **Dask** para procesamiento eficiente.
* Calcula estadísticas como **media, mínimo y máximo de los precios**.

**6. Ejecutar Todo el Sistema**

**Abrir tres terminales** y ejecutar los siguientes comandos en orden:

1. **Iniciar Kafka**

docker-compose up -d

1. **Ejecutar el productor (envía datos a Kafka)**

python producer2.py

1. **Ejecutar el consumidor (recibe y guarda datos)**

python consumer.py

1. **Procesar los datos con Dask**

python process\_dask.py

**Conclusión**

Finalmente, se tiene un **sistema de procesamiento en tiempo real** que:  
 Obtiene datos de la bolsa con **Alpha Vantage**.  
 Los envía a **Kafka**.  
 Recibe y guarda los datos en un **CSV**.  
 Procesa la información con **Dask**.

**Posibles Problemas y Soluciones**

**Kafka no arranca** → Revisar con docker ps si está corriendo.  
**No se reciben datos** → Asegurarse de que producer2.py y consumer.py están en ejecución.  
**Error en Alpha Vantage** → Verificar la API Key o probar con otra.

La versiones utilizadas dentro de los script consumer2.py y producer2.py