Universidad Diego Portales

Tarea 2 Sistemas Distribuidos

Gastón Donoso Oviedo
20.470.363-9
27 de Octubre 2022

Problema y solución

El problema entregado se basa en sopaipilleros que desean procesar sus ventas diarias, las ubicaciones de sus carros, el stock que se maneja y el ingreso de nuevos miembros a su gremio.

Para poder realizar todo lo pedido se generó un broker de kafka el cual procesa todas las solicitudes que se van creando, en este broker se encuentran cuatro distintos tópicos, los que tienen dos particiones cada uno, estos tópicos son miembros, ventas, ubicación y stock, y las particiones son para separar los miembros premium con los normales, para diferenciar la ubicación a tiempo real de un carrito con la ubicación al momento de crear una alarma, para separar la información de un carrito con la de un cliente y para separar el stock entre uno normal con uno que hay que reponer. Para poder recibir toda esta información se crearon cuatro distintos módulos, en los cuales cada uno recibe la información de un tópico para así no tener un solo punto de proceso y en donde se crean consumer groups que permiten guardar el último punto procesado por cada módulo.

Explicación de módulos de código

- **Producer**

  Para los producer se utilizó la librería aiokafka para enviar la información hasta el broker y también se utilizó flask para generar un interfaz que recibiera los datos.

  Todos los productores se encuentran en el mismo código y se basan en el mismo formato, que es recibir un POST desde flask, procesar la información y llamar a la función que envía los datos hasta Kafka.

```
@app.route('/Aviso', methods=['POST'])
def Aviso():
    if request.method == 'POST':
        carrito = request.form['carrito']
        latitud = request.form['LatitudAviso']
        longitud = request.form['LongitudAviso']
```

```
        hora = request.form['horaAviso']
        ubicacion = {'carrito' : carrito, 'latitud' : latitud, 'longitud': longitud,
                     'hora': hora}
        ubicacion = json.dumps(ubicacion).encode('utf-8')
        asyncio.run(IngresoAviso(ubicacion))
    return render_template('index.html')
```

Este ejemplo es de el aviso que se envía cuando hay una alerta por parte de un carrito y muestra como se procesa la información y como se llama a la función IngresoAviso(), la cual crea un producer que genera la conexión con Kafka y se le entregan los datos procesados y en formato json, ya que este formato es el que se maneja en Kafka. cabe mencionar que este llamado se hace con una función asincrónica.

```
async def IngresoAviso(coordenadas):
    producer = AIOKafkaProducer(
        bootstrap_servers='kafka:9092')
    await producer.start()
    try:
        await producer.send_and_wait("Ubicacion", coordenadas, partition=1)
    finally:
        await producer.stop()
```

La función como se muestra crea un producer con AIOKafkaProducer y se le entrega la dirección del servidor en donde se encuentra el broker al que se conectará, con esto se llama a la función start(), la cual ya genera la conexión directa entre el cliente y el broker. Luego en la función send_and_wait() se inserta el tópico al cual se envía la información, la información que se recopiló y la partición que guardará los datos y por último se para la conexión con el servidor mediante la función stop().

- ## Consumer

En el desarrollo del sistema se crearon dos tipos de consumer, el primero es el que ejecuta una función en bucle y otro que se ejecuta una vez ante el llamado de un tercero. Una función en bucle se generó en la recepción de miembros.

```
nest_asyncio.apply()
loop = asyncio.get_event_loop()

async def consume():

    topic = 'NuevoMiembro'
    bootstrap_servers = 'kafka:9092'
```

```
    normal = KafkaConsumer(bootstrap_servers=bootstrap_servers, auto_offset_reset='latest'
                           , group_id='normal')
    normal.assign([TopicPartition(topic, 0)])

    for message in normal:
        print ("NORMAL ->%s:%d:%d: key=%s value=%s" % (message.topic, message.partition,
                message.offset, message.key,message.value))

loop.run_until_complete(consume())
```

Esta función se ejecuta en bucle y es consumidor del registro de nuevos miembros al gremio. Para este código se crea un evento cíclico con asyncio.get_event_loop() y luego llama a run_until_complete() para ejecutar la función que hace de consumer. Ya ejecutandose el consumidor se definen dos variables, la primera es el nombre del topic al cual se le pedirá la información y el servidor en donde se encuentra este tópico, luego de esto se crea un objeto KafkaConsumer y se le entregan los valores mencionados previamente además de el id del grupo al cual pertenece el consumidor. Tras crear el objeto se determina la partición que se utilizará con assign([TopicPartition()]), entregándose a esta función el tópico y el numero de partición al que se desea acceder. Con todo lo anterior listo se crea un ciclo for para leer los mensajes obtenidos uno a uno y finalmente imprimir en pantalla la información rescatada.

El otro tipo de función se ve en la api de ventas, que recibe toda la información de los clientes y las ventas realizadas.

```
@app.route('/Ventas', methods = ['POST'])
def Ventas():
    if request.method == 'POST':
        delete_data()
        consumeVenta()
        consumeCliente()
        if get_ventas() != []:
            calcular_estadisticas()
        estadistica = view_estadisticas()
        promedio = view_promedios()
    return render_template('estadisticas.html', estadisticas=estadistica, promedios=promedio)
```

En este extracto de código se recibe un POST, el cual señala que se deben ejecutar las acciones diarias para calcular las estadísticas de los carritos, detrás de todo esto hay una algoritmia que permite calcular todos los datos de ventas totales, cantidad de ventas y promedio de compra por cliente en cada carrito, pero el análisis se enfocará en lo que es Kafka y el consumer, en este estracto el consumer actúa en consumeVenta()

y consumeCliente(), ambas funciones son muy similares, y solo se diferencian en la partición en la cual trabajan.

```
def consumeVenta():

    topic = 'NuevaVenta'
    bootstrap_servers = 'kafka:9092'

    consumer = KafkaConsumer(bootstrap_servers=bootstrap_servers,
                             group_id='estadisticasVentas', consumer_timeout_ms = 1000)
    consumer.assign([TopicPartition(topic, 0)])

    for msg in consumer:
        agregar_venta(json.loads(msg.value))

    consumer.close()
```

Esta función a diferencia de la cíclica no se ejecuta de manera asincrónica, sin embargo se utiliza el mismo procedimiento de inicializar los datos de tópico y servidor para crear el objeto KafkaConsumer, a este objeto, además del identificador de consumer group se le entrega la variable consnumer_timeout_ms, el cual cierra la conexión del consumidor en caso de que se cumpla un plazo en el cual no se transfiere información. Luego se genera el enlace con la partición especifica en la cual se trabajará, se leen los mensajes recibidos para enviarlos a una función que agrega la venta al sistema y luego se cierra la conexión cuando se procesan todos los mensajes.

Configuración de Kafka

En Docker, al crear el servidor de Kafka se entregan algunas variables de entorno que se utilizan como configuración del entorno.

```
environment:
      - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
      - KAFKA_BROKER_ID=1
      - KAFKA_CFG_LISTENERS=PLAINTEXT://:9092
      - KAFKA_CFG_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9092
      - ALLOW_PLAINTEXT_LISTENER=yes
```

KAFKA_CFG_ZOOKEEPER_CONNECT entrega los valores de conexión a Zookeper, entregando dirección y puerto de enlace, con KAFKA_BROKER_ID se determina el valor de identificación del broker, y con KAFKA_CFG_LISTENERS, KAFKA_CFG_ADVERTISED_LISTENERSKAFKA_CFG_ y ALLOW_PLAINTEXT_LISTENE se determina que la comunicación del sistema será sin autenticación y desencriptada.

También se deben crear los tópicos con los que se trabajará, para esto se utiliza el comando

```
kafka-topics.sh --create --bootstrap-server kafka:9092 --replication-factor 1
--partitions 2 --topic Ubicacion
```

En el cual se entrega la dirección y puerto del broker, el factor de replicación, que es el valor de replicas que se tendrá del tópico en distintos brokers, las cantidad de particiones que se desean y el nombre del tópico. Lo anterior se debe hacer para cada tópico que se desea crear.

Respuestas a las preguntas

- ¿Cómo Kafka puede escalar tanto vertical como horizontalmente?
  En cuanto a escalamiento horizontal, se pueden instalar múltiples máquinas que se encuentren en el mismo consumer group, para poder procesar de una forma más rápida información que tenga un flujo muy alto, en el caso del problema dado se puede relacionar con las ventas, ya que si se genera constantemente harta cantidad de información, una sola máquina no podrá procesar todo de forma eficaz, por lo que estas múltiples máquinas podrán trabajar la información en conjunto sin la necesidad de que estas se encuentren en comunicación. Y en cuanto a escalamiento vertical se puede relacionar a que si no hay suficiente presupuesto para un escalamiento horizontal, se puede mejorar el hardware de la máquina y aunque esta no sea lo suficientemente potente se podrá asegurar que la información de las ventas llegará a destino si o si por el formato de offsets que maneja y de esta forma no se pierde procesamiento de datos.
- ¿Qué características puede observar de Kafka como sistema distribuido? ¿Cómo se reflejan esas propiedades en la arquitectura de Kafka?
  De Kafka se puede observar que se pueden generar distintos brokers que procesen la información que van generando los clientes, balanceando así las cargas y descentralizando el almacenamiento de datos, para esto ayuda mucho la creación de particiones en los tópicos, ya que estas se pueden enviar a diferentes servidores a la conveniencia del sistema, algo que también se puede realizar es que estos brokers le pidan la información a otros brokers y de esta manera tener respaldos en caso de que alguno falle. Y también como se mencionó en la pregunta anterior se puede ver que distintos servidores puedan recibir información de un mismo tópico o partición y realizar una misma tarea sin la necesidad de que estos se encuentren en comunicación.

Enlaces

- Repositorio de GitHub del sistema, https://github.com/gndonoso/TareaKafkaSD