UNIVERSDAD DIEGO PORTALES

Informe Tarea II: Kafka

Sistemas Distribuidos

- María José Erazo Gonzalez
- Catalina Gómez Zapkovic

1. Problema y Solución

Se tiene que el gremio de sopaipilleros de Chile, crece a un ritmo agigantado por lo que requiere de una actualización que de sus plataformas informáticas para poder ser capaces de gestionar procesos de manera eficiente y escalable.

Para la solución a su problema, se realiza un sistema distribuido con Kafka de manera de poder gestionar los procesos internos del gremio, esto es levantar un servidor que pueda recibir peticiones, a su vez un broker de Kafka con tópicos y particiones con los cuales el servidor se pueda comunicar y realizar los procesos requeridos por el gremio de sopaipilleros. Aparte se realiza una base de datos que permitirá registrar los datos que son necesarios para este sistema distribuido.

2. Módulos de código

2.1. Docker Compose

En esta tarea se realizó, al igual que la primera, un Docker Compose con todos los servicios que se utilizarán para poder montar el sistema distribuido implementado. Este compose montará contenedores de manera separada para Zookeper, Kafka, Api, Stock, Ubicación, Venta y DB (base de datos), por lo que todo lo realizado está "dockerizado".

2.2. API

Se realizó una API en donde se conecta el servidor con Kafka levantándolo en el puerto 3000. En este se crea el broker y el producer de Kafka, este último es quien recibirá la información que se le envíe a cada topic. También se crearán todos los tópicos a utilizar a través de tres rutas distintas, estas son las siguientes.

- **Ruta 1: Registro de miembros**

  En esta ruta se tiene una función asíncrona que estará recibiendo los datos que se envíen para registrar un nuevo miembro, la ruta es "**/new**". Aquí también se crea el tópico de los nuevos miembros llamado "*newMember*", como se pide que al ingresar un nuevo miembro que sea premium este vaya a una partición distinta, en esta función se realiza un if que verá cuando el miembro ingresado sea premium para poder enviar dicha información a esta partición.

- **Ruta 2: Registro de ventas**

  En esta ruta, al igual que en la anterior, se recibe la información que llegue a "**/nuevaVenta**" y se crean dos tópicos, estos son "*nuevaVenta*" y "*Ubicaciones*". La información de la venta recibida, esto es *Cliente*, *Cantidad de Sopaipillas*, *Stock restante*, *Ubicación* y *Patente*, será enviada a los dos tópicos creados y se procesará en otros módulos de código.

- **Ruta 3: Aviso de carrito prófugo**

  Al igual que en las anteriores rutas, aquí llegará la información que se envíe a la ruta "**/carritoPerdido**", esta también usará el tópico de "*Ubicaciones*" para recibir la información. Además este tópico se utilizará sólo cuando una persona quiera denunciar a un carrito prófugo y deberá enviar las coordenadas de dicho carrito. Cabe destacar que estas serán enviadas a una partición distinta a la que se utiliza en la ruta anterior.

- **Ruta 4: Ubicación**

  Como cuarta ruta se agregó "**/ubicacion**". Como en la problemática se especifica que los carritos poseen sistemas inteligentes con internet y GPS, se asume que estos son capaces de enviar su ubicación cada cierto tiempo para poder tener el registro de la posición que tiene dicho carrito, es decir, sus coordenadas. Para esta ruta también se utilizará el tópico de "*Ubicaciones*" para recibir las entradas pero con una partición distinta a la que se utiliza en la ruta de "**/carritoPerdido**".

2.3. Procesamiento de Venta

En este módulo de código primero se tiene la conexión de Kafka con el broker y el servidor, se definen las variables que se utilizarán y luego, para procesar los datos que llegan a la ruta implementada anteriormente se crea un Consumer Group, el cual se conecta y se suscribe a un tópico, que en este caso es el tópico "*nuevaVenta*". Todo esto se realiza a través de una función asíncrona. Luego se realiza un ciclo con *eachMessage* que tendrá lo que se hará por cada mensaje que lea el consumer.

Primero se parsea la data que llega para poder leerla como un String. Como se pide que se calcule las ventas totales, los clientes totales, y el promedio de ventas por cliente, se realiza un contador por cada uno de estos y después se manda a la base de datos al terminar el día. También cada vez que llegue un "message" se enviará a la base de datos.

2.4. Procesamiento de Stock

En este módulo de código se tiene el procesamiento del stock de cada carrito, este funciona igual que el módulo descrito anteriormente (ventas) pero al momento de procesar la data se tendrá un arreglo de 5 que al llenarse lo procesará y lo guardará o actualizará en la base de datos dependiendo del carrito. Además, si alguno de los carritos tiene un stock de menos de 20 sopaipillas, entregará un aviso por pantalla.

2.5. Procesamiento de Ubicación

Al igual que los módulos anteriores, este funciona de la misma manera, se conecta a Kafka, se conecta a la base de datos y crea un Consumer Group para procesar la data que le llegue al tópico. La información que le llega cambia según la partición ya que, a la partición 0 llegarán las ubicaciones de los carritos no perdidos y a las partición 1 aquellos que sí se perdieron. Por cada ubicación que llegue, se muestra por pantalla junto con la patente del carrito. Si hay un carrito prófugo se muestra igualmente por pantalla.

3. Configuración de Kafka

Para configurar Kafka, lo primero es dockerizarlo, en el docker se encuentra Zookeeper, una dependencia de Kafka que permite que los servicios de Kafka puedan funcionar simultáneamete entre otras cosas, luego se configura Kafka para que este pueda correr en su propio contenedor al construir el Docker Compose. Aquí es donde se define la cantidad de particiones que se tendrá por topic, tal como se muestra en la Figura 1.

```yaml
  zookeeper:
    image: 'bitnami/zookeeper:3.7.0'
    restart: always
    environment:
      ALLOW_ANONYMOUS_LOGIN: "yes"
    ports:
      - 2181:2181
      - 2888:2888
      - 3888:3888

  kafka:
    image: 'bitnami/kafka:2.8.1'
    restart: always
    depends_on:
      - zookeeper
    environment:
      ALLOW_PLAINTEXT_LISTENER: "yes"
      KAFKA_CFG_ZOOKEEPER_CONNECT: "zookeeper:2181"
      KAFKA_CFG_NUM_PARTITIONS: 2
    ports:
      - 9092:9092
```

Figura 1: Configuración de contenedor de Kafka en Docker Compose.

Para poder utilizar Kafka, en cada una de los módulos de código se realiza la conexión de Kafka en el cual se crean los "Brokers", instancias de Kafka que permitirán conectarse desde el servidor a cada uno de los topic, tal como se muestra en la Figura 2. Estos tópicos son colecciones de mensajes los cuales tienen particiones que se pueden manejar dependiendo de lo que se requiera. Zookeeper es quién se encargará de administrar las instancias.

```javascript
const express = require("express");
const { Kafka } = require('kafkajs')
//const client = require("./connect")

const port = process.env.PORT;
const app = express();

app.use(express.json());

const kafka = new Kafka({
    brokers: [process.env.kafkaHost]
});
```

Figura 2: Conexión de Kafka y creación del broker

4. Respuestas a las preguntas

1. ¿Cómo Kafka puede escalar tanto vertical como horizontalmente? Relacione su respuesta con el problema asociado, dando un ejemplo para cada cada uno de los tipos de escalamiento.

   Kafka puede escalar horizontalmente debido a que se puede instanciar varios Brokers a la vez, lo que permite el poder tener varia máquinas en ejecución y facilitar el trabajo estando todo junto en un clúster, un ejemplo de esto es el tener varios brokers con distintos procesamientos como el caso de Venta, Stock y Ubicación. De manera vertical Kafka puede escalar si se aumenta el espacio de almacenamiento como por ejemplo reemplazar nodos existentes por nodos con mayor capacidad, esto para poder tener mayor cantidad de particiones y tópicos, un ejemplo sería si se reemplazaran los nodos que ya se tienen por unos con mayor capacidad de almacenamiento.

2. ¿Qué características puede observar de Kafka como sistema distribuido? ¿Cómo se reflejan esas propiedades en la arquitectura de Kafka?

Como sistema distribuido, Kafka puede distribuir sus recursos en más de un broker o partición, por ejemplo que en el tópico de "*Ubicaciones*" se utilicen distintas particiones para distintos procesos, lo que a su vez permiten que en paralelo un grupo de clientes pueda recibir varios mensajes, además es posible replicar las particiones para poder tener una alta disponibilidad. Esto va a garantizar en el sistema un alto rendimiento, baja latencia y alta disponibilidad, además de no tener pérdida de datos por los mismos brokers y particiones que este tiene.

5. Anexos

5.1. Enlace a Video

`https://drive.google.com/drive/folders/1MjSuWQE1b1nd7w5L9C94pB_cDvNJwdno?usp=sharing`

5.2. Enlace a Github

`https://github.com/majo-erazo/Tarea2-SD`