FACULTAD DE
INGENIERÍA Y CIENCIAS

Tarea : Kafka Eventos

Sistemas Distribuidos

Bastián Figueroa y Sebastián Riquelme
Nicolás Hidalgo

Índice

1. Problema y Solución

- Problema: El gremio de sopaipilleros de Chile ha crecido a un ritmo agigantado , lo cual los métodos anticuados de trabajo se sienten obsoletos , por ende se debe realizar una actualización que implica utilizar plataformas informáticas capaces de gestionar dichos procesos de manera eficiente y estable. Siendo así se busca aprovechar estos sistemas , las cuales permitan:

  1. Calcular estadísticas sobre ventas (cantidad de ventas por día y clientes frecuentes).
  2. Tener la posición geográfica en tiempo real de cada sopaipillero. Además se busca alertar eventos extraños.
  3. Preparar la reposición del stock de manera automática, al momento de tener menos de 20 de masas de sopaipillas.

- Solución:

  Se lleva a cambo crear un sistema de procesamiento orientado a eventos, ya que en este caso ocurren muchos eventos en el gremio de sopaipilleros. Para ello se contara con un sistema llamado Apache Kafka el cual distribuye en base a tópicos y particiones los distintos .eventos"que se deben llevar a cabo en el gremio de sopaipilleros.

2. Metodología

Para empezar a solucionar la problemática , se planifica una arquitectura para el sistema, las cuales contendrá los puntos claves a fin de obtener una solución óptima. La arquitectura a seguir es la siguiente:

1. Servidor : Se levanta un servidor en node js llamado express , el cual recibe todas las peticiones que se requieren. El servidor es el corazón del gremio de sopaipilleros.

2. Kafka : Se levante un broker de Kafka con los siguientes tópicos el cual cada tópico debe tener a lo menos 2 particiones:

   a) topic_coordenadas

   b) topic_nuevos-usuarios

   c) topic_stock

   d) topic_ventas

3. Servicio Base de Datos : Se ocupa la base de datos mariadb la cual permite almacenar información.

4. Peticiones : El servidor/es debe ser capaz de recibir 3 tipos de peticiones.

   a) Registro de un nuevo miembro para el gremio.

   b) Registro de una venta.

   c) Aviso de agente extraño.

5. Procesamiento : Cada una de las peticiones deberá ser procesada a posterior por distintos servidores.

   a) Procesamiento de ventas diarias.

   b) Procesamiento de stock para reposición.

   c) Procesamiento de ubicación.

Para que esta metodología se contemplan varios clientes (miembros del gremio) , el cual funcionara mediante una petición POST (En postman) , el cual permite que el servidor con el sistema y el cliente se comuniquen de manera sencilla.

3. Explicación de módulos de código

- Directorios:

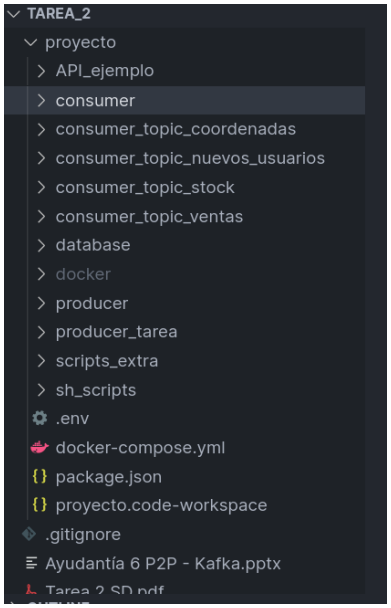

Figura 1: Directorio de códigos realizados

La Figura 1 contiene todos los códigos que se necesitan para realizar la solución a la problemática, los cuales a continuación se describen (los códigos completos estan en la sección enlaces, donde se encontrará el repositorio):

1. Producer: Este bloque de código es un ejemplo de un productor usando la librería kafkajs. Es un ejemplo básico que no usamos en el proyecto, pero el script real producer que está en el directorio producer_tarea sigue la misma estructura, sólo que con diferentes rutas de api para escribir en diversos topics.

```
const express = require('express');
const { Kafka } = require('kafkajs');
//const cors = require('cors');
const app = express();

//app.use(cors());
app.use(express.json());

const kafka = new Kafka({
    brokers: ["kafka:9092"]
});

const producer = kafka.producer();

app.get('/', async (req, res) => {
    console.log("\n\n\n-------Mensaje producer--------\n\n")

    //chantar timeout
    console.log("Producer conectando...\n")
    await producer.connect()
    console.log("Producer conectado!\n")

    //AQUI CAGA
    await producer.send({
        topic: "test-topic",
        //value: JSON.stringify(user)
        messages: [{ value: "Hola desde JS" }]
    })

    console.log("Producer send terminado!")

    res.send("Producer send terminado!")
});


app.listen(3003, () => {
    console.log("\nServer PRODUCER corriendo en puerto: 3003\n");
});
```

Figura 2: Producer

2. Consumer: Tal como en el directorio producer, este es un código básico para mostrar la estructura que siguen los consumer usando la librería kafkajs. Este bloque de código no es usado en el proyecto, pero sirve como guía para agregar más consumers.

```
const express = require("express");
const { Kafka } = require("kafkajs");
//const cors = require('cors');
const app = express();

//app.use(cors());
app.use(express.json());

//Conexion a kafka
const kafka = new Kafka({
    brokers: ["kafka:9092"],
});

app.get("/", async (req, res) => {
    console.log("\n\n\n-------Mensaje consumer-------\n\n\n");

    console.log("Iniciando objeto consumer...\n");
    const consumer = kafka.consumer({
        groupId: "test-topic-consumer" /* , fromBeginning: true */,
    });
    console.log("Consumer iniciado!\n");
    console.log("Conectando a consumer...\n");

    await consumer.connect();
    console.log("Consumer conectado!\n");
    console.log("Suscribiendose al topic...\n");

    await consumer.subscribe({ topic: "test-topic", fromBeginning: true });
    console.log("Suscrito al topic!\n");
    console.log("Ejecutando consumer run...\n");
    await consumer.run({
        eachMessage: async ({ topic, partition, message }) => {
            console.log("\n\nTOPIC: ", topic, "\n\n");
            console.log("\n\nMESSAGE: ", message, "\n\n");
            console.log(
                "\nMESSAGE:VALUE: ",
                JSON.parse(message.value.toString()),
                "\n"
            );
            //let data = JSON.parse(message.value)    ;
            //console.log(data)
        },
    });

    console.log("Consumer terminado!");

    res.send("Consumer terminado!");
});

app.listen(3002, () => {
    console.log("\nServer CONSUMER corriendo en puerto: 3002\n");
});
```

Figura 3: Consumer

3. Consumer_topic_coordenadas: Corresponde al tópico donde el producer se conecta para realizar dichas actividades correspondientes en este caso , esta relacionado a observar cada coordenada que contiene cada carrito, para tener registro de posición de cada carrito y asi evitar que se pierda.
4. Consumer_topic_nuevos_usuarios: Corresponde al tópico donde el producer se conecta para realizar dichas actividades correspondientes en este caso , esta relacionado a registrar nuevos trabajadores que usaran carritos , en este caso como se tienen dos tipos de registro uno premium y uno básico, por ende se trabaja con una condición para observar a que partición ira cada nuevo usuario.
5. Consumer_topic_stock: Corresponde al tópico donde el producer se conecta para realizar dichas actividades correspondientes en

este caso , esta relacionado con observar el stock de los carritos , lo cual tambien observa los datos que se ingresan al registrar una venta, asi finalmente cuando el stock es menor a 20 unidades se repondrá al carrito lo que falte para volver a su stock inicial.

6. Consumer_topic_ventas: Corresponde al tópico donde el producer se conecta para realizar dichas actividades correspondientes en este caso , esta relacionado a registrar las ventas.
7. database: La base de datos corresponde a una versión de mysql, llamada mariadb la cual es de código abierto y contiene todo lo que se necesita guardar.
8. docker_compose.yml: Contiene todos los contenedores que se requieren para la solución a la problemática , por ejemplo estan llamados todos los códigos antes descritos junto con su configuración correspondiente, además de la base de datos y el servicio de kafka.

4. Configuración de Kafka

```yaml
version: "3"
services:
    zookeeper:
        image: "bitnami/zookeeper:latest"
        ports:
            - "2181:2181"
        environment:
            - ALLOW_ANONYMOUS_LOGIN=yes
    kafka:
        image: "bitnami/kafka:latest"
        container_name: kafka
        #networks:
        #  - my_network
        ports:
            - "9092:9092"
        environment:
            - KAFKA_BROKER_ID=1
            - KAFKA_CFG_LISTENERS=PLAINTEXT://:9092
            - KAFKA_CFG_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9092
            - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
            - ALLOW_PLAINTEXT_LISTENER=yes
            - KAFKA_CFG_NUM_PARTITIONS=2
        depends_on:
            - zookeeper
```

Figura 4: Configuración de Kafka

Para esta actividad se realizo la siguiente configuración de kafka:

1. KAFKA_BROKER_ID=1 : Corresponde al id del servidor de kafka , en este caso solo se usa un servidor de kafka, pero eventualmente se puede crear mas servicios de kafka.

2. KAFKA_CFG_LISTENERS=PLAINTEXT://:9092 : Es lo que utilizara el intermediario para crear sockets de servidor y que el oyente interno escuchará en el puerto 9092.

3. KAFKA_CFG_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9092 : Los clientes internos se conectarán al intermediario en el host de kafka y los clientes externos se conectarán al host local.

4. KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181 : El contenedor kafka lee esta variable entorno para saber donde llegar a nuestro contenedor Zookeeper en la red Docker kafka-net.

5. ALLOW_PLAINTEXT_LISTENER=yes : Que todos estén activos.

6. KAFKA_CFG_NUM_PARTITIONS=2 : Corresponde a cuantas particiones por defecto tendrán los topic creados.

5. Respuestas Preguntas

- ¿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.

  R: Kafka puede escalar tanto vertical como horizontalmente ya que los mensajes que se le entregan no se eliminan una vez son consumidos por un tópico, lo cual pueden residir de forma indefinida si asi se requiere sin impactar el rendimiento del sistema. Esto significa que un consumidor puede preprocesar todos los mensajes de un tópico y a partir de ellos regenerar el estado puede ser en memoria o incluso en una base de datos. También proporcionando resistencia a fallos. Llevándolo a la problemática de la tarea 2, se va haciendo un escalamiento horizontal a medida que los tópicos van teniendo particiones, por ejemplo cuando se registra un nuevo usuario se hacen dos particiones donde uno corresponde si el usuario es "premiumz el otro corresponde a usuario "básico". Mientras que en escalamiento vertical para esta tarea debido a que es un sistema completo para implementar en el gremio, se agregarían mejoras de hadware a los computadores ya existentes, con el mismo sistema y que trabaje con la misma información, formando así un escalamiento vertical.

- ¿Que características puede observar de Kafka como sistema distribuido? ¿Cómo se reflejan esas propiedades en la arquitectura de Kafka?

R: Tiene varias características como: Escalabilidad horizontal, bajo tiempo de respuesta,durabilidad de los mensajes y consistencia. La escalabilidad horizontal se ve reflejada en la replicación, que en kafka funciona en base a las particiones que conforman un tópico, esto tambien ayuda al bajo tiempo de respuesta y a la consistencia ya que las particiones garantizan una consistencia total y una buena disponibilidad, la durabilidad de los mensajes se debe a que los mensajes son eliminados de kafka una vez son consumidos, si no que dispone de un tiempo de retención configurable después de lo cual se descartan para liberar espacio, esto evita el reprocesado de dichos mensajes.

6. Conclusión

Se desarrollo con éxito lo pedido en la actividad, sin embargo el tópico de ver la posición de los carritos prófugos no se realizo del todo ya que falto el aviso cuando el carrito no se reportaba. El sistema funciona correctamente junto con el sistema Kafka y sus respectivos tópicos a las cuales tienen distintas funcionalidades para que el sistema sea rápido. El sistema Kafka se adecuo muy bien a la actividad ya que el Gremio de sopaipilleros necesitaban un sistema que tuviera bajo tiempo de respuesta, una durabilidad de los mensajes ya que un mensaje podria servir para diferentes cosas , etc . Finalmente se puede concluir que el sistema Kafka es un sistema muy versátil y que se adecua en varias situaciones.

7. Enlaces

1. **Documentación Kafka** : `https://kafka.apache.org/documentation/`
2. **Documentación Kafka** : `https://www.subhadig.net/running-kafka-in-docker.html`
3. **Documentación Kafka** : `https://stackoverflow.com/questions/42998859/kafka-server-configuration-listeners-vs-advertised-listeners`
4. **Repositorio Tarea** : `https://github.com/SebastianRiquelmeM/tareas_sistemas_distribuidos/tree/main/tarea_2`
5. **Enlace a video** : `https://youtu.be/xfOGM4rr9Jw`