TAREA 2: SISTEMAS DISTRIBUIDOS
Kafka: eventos
Profesor: Nicolás Hidalgo
Ayudantes: Cristian Villavicencio, Joaquín Fernandez y Nicolás Núñez

LEA EL DOCUMENTO COMPLETO ANTES DE EMPEZAR A DESARROLLAR LA TAREA

Objetivo

El objetivo de este trabajo es introducir a los estudiantes a los *sistemas de procesamientos basados en eventos de tipo stream*. Para ello, los alumnos deberán trabajar con **Apache Kafka** un sistema de *micro-batching* ampliamente utilizando en la industria hoy en día. Los estudiantes deberán comprender y aplicar las funcionalidades de Kafka, configurar sus componentes y reconocer las ventajas de este tipo de tecnologías.

Conceptos previos

Apache Kafka es una plataforma distribuida de transmisión de datos que permite *publicar*, *almacenar* y *procesar* flujos de registros, así como *suscribirse* a ellos, de forma inmediata. Está diseñada para administrar los flujos de datos de varias fuentes y distribuirlos a diversos usuarios.

Kafka posee algunos términos que debe conocer:

- **Broker**: Corresponde a un servidor de Kafka. Pueden existir varios conectados en una red y son capaces de comunicarse entre ellos utilizando un mecanismo propio. Siempre existe un broker líder; en caso de que este se caiga, otro entra al mando.
- **Producer**: Es aquel que produce/publica datos en algún flujo.
- **Topic**: Es un canal donde se publica el flujo de datos. Puede relacionarse con una estructura de datos como lo es una cola.
- **Consumer**: Es aquel que consume los datos publicados en un topic. Sin embargo, los datos al ser consumidos **no** son borrados del topic.
- **Partition**: Es parte del flujo generado en un topic. Podría decirse que son *mini-topics* y pueden repartirse entre distintos brokers. Cada partición puede contener información distinta, sin embargo, se utilizan para ordenar una jerarquía con la información.
- **Consumer group**: Dado que los datos de un topic de Kafka no son borrados, existe este mecanismo para guardar el *offset* de la lectura de los datos. Es decir, que si un consumer group lee la información del consumer group X, Kafka se encargará de guardar la posición del último valor leído, para así no perder el orden. Dos *consumer groups* distintos tendrán distintos *offset*.

Para más detalles, revisar la documentación oficial de apache Kafka: `https://kafka.apache.org/`. Por otro lado, se le recomienda revisar el siguiente repositorio: `https://github.com/Naikelin/async-events-kafka`

Problemática

El gremio de sopaipilleros de Chile, encargado de establecer políticas legales para la venta en carritos de sopaipillas, ha crecido a un ritmo agigantado. Los anticuados métodos de trabajo para dar soporte a las tareas del gremio requiere de una actualización que implica la utilización de plataformas informáticas capaces de gestionar dichos procesos de la manera más eficiente y escalable.

La gestión de los procesos internos del gremio es compleja. Los miembros que participan en este reciben de manera constante peticiones (escritas a mano) con tareas que deben realizarse. Por ejemplo, realizar la inscripción de un nuevo miembro. Estas tareas tradicionalmente son repartidas según su tipo a los diferentes encargados de gestionar y llevar a cabo las mismas.

Uno de estos procesos nombrados, corresponde a la inclusión de nuevos miembros. Este proceso es engorroso y tardío; requiere dejar una petición formal en el gremio, el cual puede ser resulto en cuestión de meses. Esta petición viaja a la dirección encargar de procesar y evaluar nuevos miembros, los cuales se fijan en los antecedentes de los dueños de los carritos postulantes, para así aprobar una lista de nuevos miembros. Esta lista es mostrada de manera periódica, una vez al mes. Este proceso se puede acelerar pagando una comisión, llamado *Inscripción Premium*. Lo que se busca es automatizar este proceso utilizando sistemas informáticos.

Por otro lado, los maestros sopaipilleros, poseen carritos modernos (es una de las condiciones para ser parte del gremio). Estos carritos poseen sistemas inteligentes con internet y GPS. Además, poseen un sistema capaz de registrar ventas y posteriormente ejecutar código, sin embargo, el gremio no puede aprovechar esta ventaja. Se busca poder aprovechar estos sistemas, utilizando sistemas informáticos automatizados que permitan:

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

Propuesta de arquitectura

La arquitectura que da soporte a el sistema previamente descrito debe estar compuesto por:

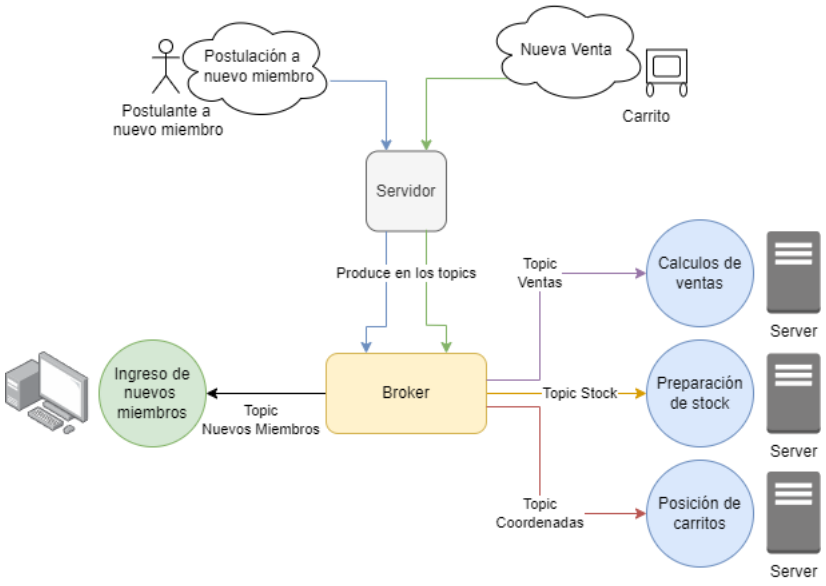
- **Servidor**: Levantar un servidor que pueda recibir peticiones, a través de un CRUD o algún otro medio que permita recibir la información. Este será el *corazón* del gremio de Sopaipilleros.
- **Kafka**: Levantar un broker de Kafka y configurar los tópicos que usted crea que sean necesarios. Cada tópico debe tener al menos 2 particiones.
- **Servicio de Base de Datos**: Se debe levantar alguna instancia que permita almacenar información. Puede ser una Base de datos relacional, no relacional o incluso un archivo de texto.
- **Peticiones**: El servidor debe ser capaz de recibir 3 tipos de peticiones:
    1. **Registro de un nuevo miembro para el gremio**: El servidor debe ser capaz de recibir el registro de un miembro para el gremio de sopaipilleros. El registro debe contener los campos de: *Nombre*, *Apellido*, *Rut*, *Correo del dueño*, *Patente carrito* y *Registro premium*. Cuando este registro sea "Premium" debe ser enviado en una nueva partición.
    2. **Registro de una venta**: Un maestro sopaipillero enviará una petición de registro de venta, que será procesada por distintos servicios a posterior. El registro debe contener los campos de *Cliente*, *Cantidad de Sopaipillas*, *Hora*, *Stock restante* y *Ubicación del carrito*.
    3. **Aviso de agente extraño**: Cualquier persona podrá denunciar a un carrito *prófugo*. El registro contendrá simplemente *coordenadas* y las enviará por el tópico encargado de enviar las ubicaciones de los carritos, pero utilizando una partición distinta.

IMPORTANTE: Usted deberá relacionar lo anterior con lo que usted sabe del problema y de Kafka, para repartir entre distintos **tópicos/particiones/endpoints** la información.

- **Procesamiento**: Cada una de las peticiones deberá ser procesada a posterior por distintos servidores. Debido a que Kafka genera flujos de datos encolados, ninguno de los clientes de procesamiento (ó *consumers*) deberá correr necesariamente en paralelo al servidor. Se deben crear los siguientes programas:

  1. **Procesamiento de ventas diarias**: Se ejecuta una vez al día y calcula, para cada maestro sopaipillero, ventas totales, promedio de ventas a clientes (cuantas les vende en promedio a un cliente) y clientes totales. Los valores los imprime por pantalla o los guarda en un archivo de texto.
  2. **Procesamiento de stock para reposición**: Nunca se deja de ejecutar y constantemente está leyendo las consultas, las guarda por lotes en un arreglo de tamaño 5 para posteriormente leerlas y preparar las entregas. Entrega un aviso por pantalla.
  3. **Procesamiento de ubicación**: constantemente está leyendo las entradas y las va reemplazando en tiempo real para mostrar la posición de cada carrito. Si un carrito no envía su posición luego de 1 minuto, desaparece. Las posiciones son simplemente un aviso por pantalla que organiza y asocia a los carritos con sus coordenadas. También mostrará los registros de los carritos *prófugos*.

  IMPORTANTE: El consumo de cada topic se debe realizar a través de un Consumer Group, por cada topic.
  La arquitectura propuesta, **sin tanto detalle**, puede ser interpretada en la siguiente figura:



Preguntas

Además de los puntos anteriores pedidos, se le pide contestar las siguientes 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.

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

Entrega

Para la entrega de esta tarea, usted deberá realizar:

- Un repositorio con todos los códigos utilizados.
- Un video donde solamente se muestre el funcionamiento del sistema.
- Un informe donde explique, con sus palabras, brevemente el código desarrollado. Se deben tener las secciones: *Problema y solución*, *Explicación de módulos de código*, *Configuración de Kafka* y *Respuestas a las preguntas*.

Aspectos formales de entrega

- **Fecha de entrega:** 27 de Octubre 23:59 hrs.
- **Número de integrantes:** Grupos de 2 personas las cuales deben estar claramente identificadas en la tarea.
- **Pauta de evaluación:** La pauta se puede encontrar en el siguiente link `https://docs.google.com/spreadsheets/d/1HHyyVhfbkPshcZknqw082UpHNtfe2Gc1WfFfqMM96qU/edit?usp=sharing`.
- **Lenguaje de Programación:** para la implementación debe escoger entre los siguientes lenguajes: **Python**, **GO** o **JavaScript**.
- **Formato de entrega:** Repositorio público (Github o Gitlab), video de funcionamiento e informe en formato PDF.
- **Tecnologías complementarias:** En caso de usar tecnologías complementarias, añadir una descripción en el informe.
- **Contenedores de Kafka y Postgres**: se recomienda utilizar las siguientes imágenes: `https://hub.docker.com/r/bitnami/kafka/`, `https://hub.docker.com/r/bitnami/postgresql/`. En caso de utilizar otra imagen, deberá especificarlo en el informe.
- Las copias de código serán penalizadas con nota mínima. Referente apropiadamente todo segmento de código que no sea de su autoría.
- No se debe implementar un front o interfaz, solo basta con implementar una API REST o una interfaz interactiva en la terminal.
- Consultas: **nicolas.nunez2@mail.udp.cl** o **Naike#2258**