Lab 6 – Kafka en Twitter – Java Guide

CC5212-1 – April 23, 2025

Esta es una guía opcional para ayudarte a programar tu solución en Java. Este documento consta de dos partes. La primera parte es crear una clase `EarthquakeFilter.java` que lea records desde un topic de entrada y escriba solo aquellos records para terremotos (earthquakes) en un topic de salida. La segunda parte es crear una nueva clase llamada `BurstDetector.java` que acepte un topic de entrada y detecte eventos basados en bursts que cumplan más de x records escritos en un intervalo de y segundos (Consulta las instrucciones del laboratorio para detalles adicionales).

Paso 1: `EarthquakeFilter`

Copia la clase `PrintEarthquakeTweets.java` a `EarthquakeFilter.java` en el mismo package y:

- Adapta el código para que espere *dos* argumentos, de lo contrario que imprima `"Usage [inputTopic] [outputTopic]"`.
- Elimina la línea `System.out.println(record.value());`.

A continuación, debes insertar dos fragmentos de código en los lugares adecuados (es posible que también necesites importar nuevos packages):

crear un producer: Crea un nuevo producer para enviar records a Kafka con la configuración `props`.

```
Producer<String, String> producer = new KafkaProducer<String, String>(props);
```

Enviar un mensaje: Usa el producer para enviar un record al topic especificado por `args[1]` (segundo argumento de línea de comandos), utilizando la partición (`0`), copiando el timestamp, key y value del record de entrada al record de salida.

```
producer.send(new ProducerRecord<>(args[1], 0, record.timestamp(), record.key(), record.value()));
```

Paso 2: `BurstDetector`

Copia la clase `PrintEarthquakeTweets.java` a `BurstDetector.java` en el mismo package y:

- Elimina la línea `System.out.println(record.value());`.

A continuación, debes insertar el siguiente código en los lugares apropiados. Los fragmentos pueden presentarse desordenados (y posiblemente necesites importar nuevos packages).

Definición de constantes: Según las instrucciones del laboratorio: `FIFO_SIZE` corresponde a x, `EVENT_START_TIME_INTERVAL` corresponde a y, y `EVENT_END_TIME_INTERVAL` corresponde a $2y$. Estos últimos tiempos se consideran en milisegundos.

```
public static final int FIFO_SIZE = 50;
public static final int EVENT_START_TIME_INTERVAL = 50 * 1000;
public static final int EVENT_END_TIME_INTERVAL = 2 * EVENT_START_TIME_INTERVAL;
```

Inicialización de la cola FIFO: Utilizamos una `LinkedList` para implementar una cola first-in first-out (FIFO) que almacene los últimos `FIFO_SIZE` records.

```
LinkedList<ConsumerRecord<String, String>> fifo = new LinkedList<ConsumerRecord<String, String>>();
```

Encolar en la cola FIFO: Añade un record a la cola FIFO.

```
fifo.add(record);
```

Desencolar de la cola FIFO: Obtener y eliminar el primer/antiguo elemento de la cola FIFO.

```
ConsumerRecord<String, String> oldest = fifo.removeFirst();
```

¿Está llena la cola FIFO?: Verifica si tenemos `FIFO_SIZE` elementos en la cola FIFO:

```
if(fifo.size()>=FIFO_SIZE){
        ?
}
```

Inicialización de variables de estado: Usaremos estos para rastrear si estamos en un evento y la cantidad de eventos hasta ahora.

```
boolean inEvent = false;
int events = 0;
```

En evento: Flag que indica si estamos dentro de un evento activo.

```
inEvent = true;
```

Fuera de evento: Flag que indica que no estamos procesando un evento activo.

```
inEvent = false;
```

Contar nuevo evento: Incrementa el contador de eventos detectados.

```
events++;
```

Medir el intervalo (gap): Encuentra el tiempo transcurrido (ms) entre el registro actual y el más antiguo.

```
long gap = record.timestamp() - oldest.timestamp();
```

Verificar el intervalo: Evalúa algunas condiciones sobre el intervalo (gap) y si estamos en un evento o no.

```
if(gap <= EVENT_START_TIME_INTERVAL && !inEvent){
        ?
} else if(gap >= EVENT_END_TIME_INTERVAL && inEvent){
        ?
}
```

Imprimir detalles del inicio del evento: Imprime el ID del evento que está comenzando, junto con la hora de inicio, el primer mensaje y (opcionalmente) la tasa de ráfaga (burst rate). (El primer evento detectado debe tener el ID 1, el segundo evento ID 2, etc.)

```
System.out.println("START event-id:"+ events +": start:"+oldest.timestamp()+
        " value:"+oldest.value()+" rate:"+FIFO_SIZE+" records in "+gap+" ms");
```

Imprimir detalles del final del evento: Imprime el ID del evento que está finalizando y (opcionalmente) la tasa de ráfaga (burst rate).

```
System.out.println("END event:"+ events +" rate:"+FIFO_SIZE+" records in "+gap+" ms)");
```