ENMILOCALFUNCIONA
THOUGHTS, STORIES AND IDEAS.

Aprendiendo Apache Kafka (Parte 3) : Conceptos Básicos Para Desarrollo

Publicado por Víctor Madrid el 27 November 2018

Arquitectura de Soluciones | Kafka

En este **tercer artículo** de la serie **"Aprendiendo Apache Kafka"** se van a detallar qué y cómo se comportan los otros **componentes o partes** que conforman la parte más de **desarrollo** sobre la plataforma **Apache Kafka** .

A modo de recordatorio pongo los enlaces a los artículos anteriores :

- Introducción
- Conceptos Básicos donde se trataron : Zookeeper, Broker, Clúster, Mensaje, Esquema, Topic, Partición y Offset

Y ahora añadiremos otros casi igual de "divertidos" :

- Connect
- Streams
- Productor

- Consumidor
- Grupo de Consumidores

> **IMPORTANTE**
>
> La parte de conceptos de la plataforma se tratan en el segundo artículo, así que no te olvides revisarlo si te surge alguna duda.

Este artículo esta divido en 2 partes:

- **1. Componentes:** Detalles sobre cada uno de los elementos que componen la parte más de desarrollo.
- **2. Conclusiones:** Opinión sobre los resultados obtenidos.

Arrancamos este artículo....

1. Componentes

En este punto se aclararán que elementos están más relacionados con la parte que tendrá que ver con las aplicaciones y las funcionalidades que harán uso de la plataforma Kafka.

Connect

Definiciones

- *Componentes que pueden ser configurados para "escuchar" los "cambios" que ocurren en alguna fuente de datos (ficheros, base datos, una aplicación, etc.) y envía los datos automáticamente a un topic o viceversa.*
- *Herramienta proporcionada por Kafka (escalable, distribuida , tolerante a fallos y segura) que permite el streaming de datos entre la plataforma y otros sistemas de datos.*
- *Servicio basado en conectores para mover información de entrada y salida a la plataforma.*

Características

- *Pertenece a **Kafka Connect**.*
- *Facilita la **integración** adecuada entre la plataforma Apache Kafka y otros sistemas de streaming/batch/fuente de datos.*

*Para ello establece un **framework** común para la implementación, despliegue y gestión de conectores (los artefactos generados que verifican una funcionalidad concreta).*

- Un conector "source" (Source Connector) ataca una localización "source"/origen y transmite la información de la fuente a Kafka Connect (es decir, lee registro de alguna fuente de datos y los publica en un topic) -> Se le suele denominar "productor"

- Un conector "sink" (Sink Connector) ataca a una localización "sink"/destino y transmite la información de Kafka Connect a la fuente (es decir lee registros de un topic y los pone en alguna fuente de datos) -> Se le puede denominar "consumidor"
- Si se produce un fallo entonces Kafka Connect avisará al conector
- *Abstrae de los problemas comunes de la plataforma: conversión de datos (serialización), balanceo , tolerancia a fallos, gestión de esquemas, operaciones, gestión de offsets (lo realiza automáticamente) etc.*

Facilita la expansión de conectores disponibles para la comunidad con el Connector API -> Permite crear piezas reutilizables por la comunidad

- *Existen conectores ya implementados en el mercado para : JDBC, Hadoop, HIVE, elastic, Cassandra, S3 , etc.*
- Muy fácil de usar gracias a su configuración vía ficheros
- En algunos casos no necesita mucho código
- Facilita la evolución de las arquitecturas
- ***Objetivo** centrado únicamente el **movimiento de los datos** -> Copia datos desde/hasta otros sistemas de forma o no paralelizada*
- Tienes dos modos de funcionamiento : standalone y distributed*
- *Proporciona una interfaz REST API para la gestión de conectores (en el clúster)*

Cada instancia de un worker inicia un servidor web "incorporado"

- *Este aspecto afecta más al modo distribuido*
- *Proporciona mejoras en el uso del REST Proxy y el uso de esquemas*

Al proceso del sistema que ejecuta los conectores se le se suele denominar worker

- Suele trabajar con un formato de datos del tipo AVRO

Configuración

Nota : los parámetros de configuración se detallarán en los ejercicios prácticos.

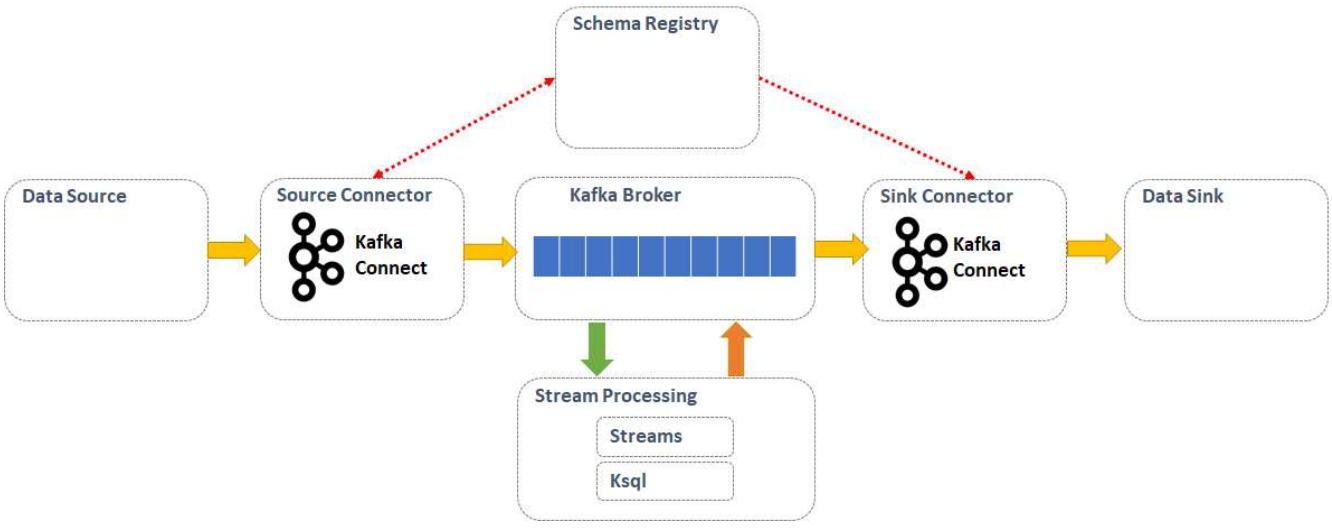
Funcionamiento

- ***Partes:***
- **Connectors :** Componente que coordina el streaming de datos a partir de las tareas (una instancia de un conector es un "job" lógico)
- **Task :** Implementación de la forma de copiar los datos hacia o desde la plataforma
- **Worker :** Proceso de ejecución que utiliza connectors (conectores) y tasks (tareas) -> Instancia de trabajo o bien un proceso de Java
- ***Estados de un connector:***

- **UNASSIGNED:** El connector/task no ha sido asignado todavía al worker
- **RUNNING:** El connector/task esta ejecutándose
- **PAUSED:** The connector/task ha sido parado
- **FAILED:** The connector/task ha fallado
- ***Modo Standalone:***
- Se ejecuta en un único proceso
- Muy útil para pruebas, desarrollos básicos y casos de uso con consumidores / productores que no necesitan ser distribuido -> Sigue un enfoque más ETL
- No requiere Zookeeper -> sólo requiere un broker
- La configuración es guardada en un fichero y especificada como argumentos de linea de comandos
- El offset es registrado en el fichero local
- No es escalable ni tolerante a fallos
- Utiliza el script : connect-standalone
- Requiere para funcionar 1 fichero de configuración del worker y al menos un fichero de configuración de conectores
- ***Modo Distributed:***
- Se ejecuta en múltiples procesos en la misma máquina o distribuidos en varias máquinas
- La configuración es guardada en un fichero y especificada como argumentos de linea de comandos
- El offset es registrado en el topic
- Escalable y tolerante a fallos -> Gracias al soporte de grupos de consumidores

> **IMPORTANTE**
>
> Este punto no os preocupéis que se tratará de forma individual en su artículo particular, donde se entrará más en detalle en sus características y usos.

Diagrama Conceptual : "Esquema de uso de Kafka Connect"



Streams

Definiciones

- ***Procesamiento en tiempo real de datos*** *(mensaje a mensaje) de forma continua y concurrente*
- Un stream se define como un conjunto de datos ilimitado y actualizado continuamente
- Permite analizar y procesar datos contenidos en la plataforma Kafka
- *Proceso basado en la lectura de mensajes de un topic del tipo fuente/"source", realizar alguno tipo de operativa sobre esos datos (análisis, limpieza, transformación) y posteriormente escribir lo obtenido en un topic del tipo "destino"*

Características

- *Pertenece a* ***Kafka Streams***
- Permite **reutilizar** las **características** de la plataforma Kafka : paralelización, tolerancia a fallos, etc
- **Alternativas** al procesamiento de flujos de datos (stream) :
- **Opción 1 :** disponer de un productor y consumidor Kafka personalizados
- **Opción 2 :** hacer uso de Flink, Storm, etc.
- **Librería cliente** para la **construcción de aplicaciones** Java simple, ligera y con dependencias sólo a la plataforma
- Se pueden considerar prácticas como despliegue y empaquetado
- Cualquier aplicación con la incluya se considera una aplicación de Stream processing
- Tiene en consideración aspectos como : tiempo del evento, tiempo de procesamiento, tipo de windowing (trabajar con cierta cantidades de datos), joins, aggregations etc.
- **Funciona** garantizando que cada registro se procesará una vez y solamente una vez (**"exactly once"**)

Configuración

Nota : los parámetros de configuración se detallarán en los ejercicios prácticos

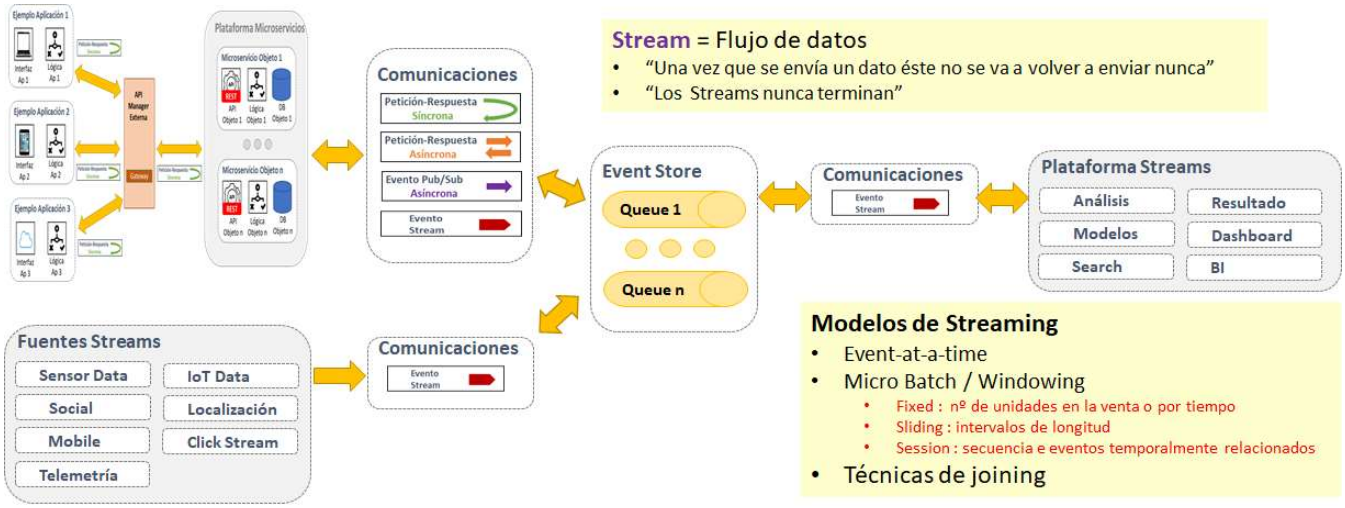
Funcionamiento

- ***Tipos:***
- **Source Processor:** Lee registros de uno o varios topics Kafka y los envía como stream a otros procesadores -> Genera un input stream
- **Sink Processor:** Lee el stream de un procesador y los envía a un topic Kafka específico o bien un sistema externo

> **IMPORTANTE**

> Este punto no os preocupéis que se tratará de forma individual en su artículo particular, donde se entrará más en detalle en sus características y usos.

Diagrama Conceptual : "Esquema de uso Streaming Processing"



Productor

Definiciones

- *Tipo de **cliente** de **Kafka** que se encarga de **publicar los mensajes***
- ***API para generar un stream de mensajes***

Características

- *También se denomina "publicador", "publisher", "productor", "editor" o "escritor"*
- ***Genera un mensaje sobre un topic específico** (no debería de tener en cuenta la partición utilizada o bien usar el criterio asignado) añadiéndolo por el final*
- *El mensaje se compone : nombre del topic, offset y el nº de partición al que enviar*
- ***Múltiples productores pueden escribir sobre diferentes particiones del mismo topic***
- *Cada productor tiene su **propio offset***

Configuración

Nota : los parámetros de configuración se detallarán en los ejercicios prácticos

Sobre todo hay que tener en cuenta los siguientes aspectos :

- **Tipo de comunicación :** síncrona (sync) o asíncrona (async)
- Tener en cuenta el tiempo del evento , el tiempo de espera y el tiempo de procesamiento
- **Tamaño del Batch (agrupación de mensajes)**
- Se mide en bytes totales y no en nº de mensajes
- Nunca debe exceder la memoria total (Por defecto 16384)

- Hay que tratar que el productor se encuentre activo la mayor parte del tiempo posible
- Se aconseja determinar un tamaño adecuado a las características del topic y a los criterios de funcionalidad a implmentar
- Tener un mayor tamaño (mayor cantidad de bytes de elementos que componen el byte) implica tener mayor rendimiento pero provoca mayor latencia (tiempo que lleva procesar un elemento)
- Se le puede aplicar compresión

Funcionamiento

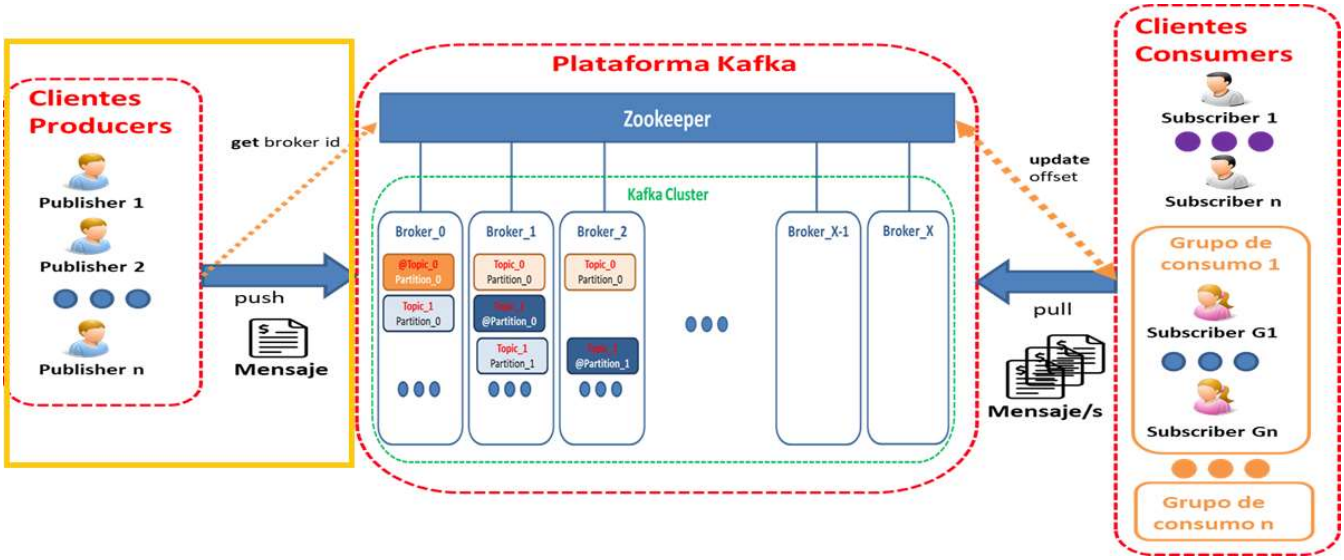
Carga/Reparto de trabajo :

- Por defecto "Round-Robin"
- *Se distribuyen los mensajes uniformemente en las particiones del topic*
- En base a algún criterio
- *Los mensajes se asignarán a una partición en concreto en base a unas necesidades específicas -> requiere un desarrollo custom*
- *Se usará la clave del mensaje y un particionador (reglas de negocio propias) que generara un hash de la clave y lo mapeará a una partición específica en concreto (por ejemplo mediante un sistema de prioridades)*
- *Se asegurará que este mapeo sea determinista -> la misma clave elige la misma partición cada vez*

Nivel de consistencia o replicación :(request.required.acks) -> Afecta a la durabilidad de los mensajes

- ***ACK=0***
- *El productor NO esperan ningún ACK desde los brokers*
- *Los mensajes añadidos al topic son considerados enviados*
- *El mensaje se pierde si la partición leader se cae*
- *No garantiza la durabilidad*
- ***ACK=1***
- *La partición leader escribe el mensaje en su log local pero sin confirmar la escritura a los followers/replicas*
- *Si el leader falla después de enviar el ACK, entonces el mensaje puede perderse*
- ***ACK=all (-1)***
- *La partición leader espera la confirmación de escritura de todos los ISR antes de enviar el ACK al productor*
- *Garantiza que los mensajes NO serán perdidos si uno de los ISR se encuentra vivo*
- *Debería usar como mínimo una réplica*

Diagrama Conceptual : "Esquema de la plataforma en lo relacionado con productores"



Consumidor

Definiciones

- *Tipo de **cliente** de **Kafka** que se encarga de **consumir los mensajes***
- ***API para consumir un stream de mensajes***

Características

- *También se denomina "subscriber", "suscriptor" o "lector"*
- ***Puede estar subscrito a uno o más topics** -> cierta independencia del broker/nodo y las particiones*
- *Cada consumidor es responsable de gestionar su propio offset en su partición*
- *Múltiples consumidores pueden leer mensajes desde el mismo topic*
- *Cada consumidor realiza el seguimiento de sus punteros vía tuplas (offset, partition, topic)*
- ***Consumer lag :** ¿Cuánto de lejos está el consumidor de los productores?*
- Existen diferentes scripts que pueden utilizarse para conocer el offset, reasignar particiones etc.

Configuración

Nota : los parámetros de configuración se detallarán en los ejercicios prácticos

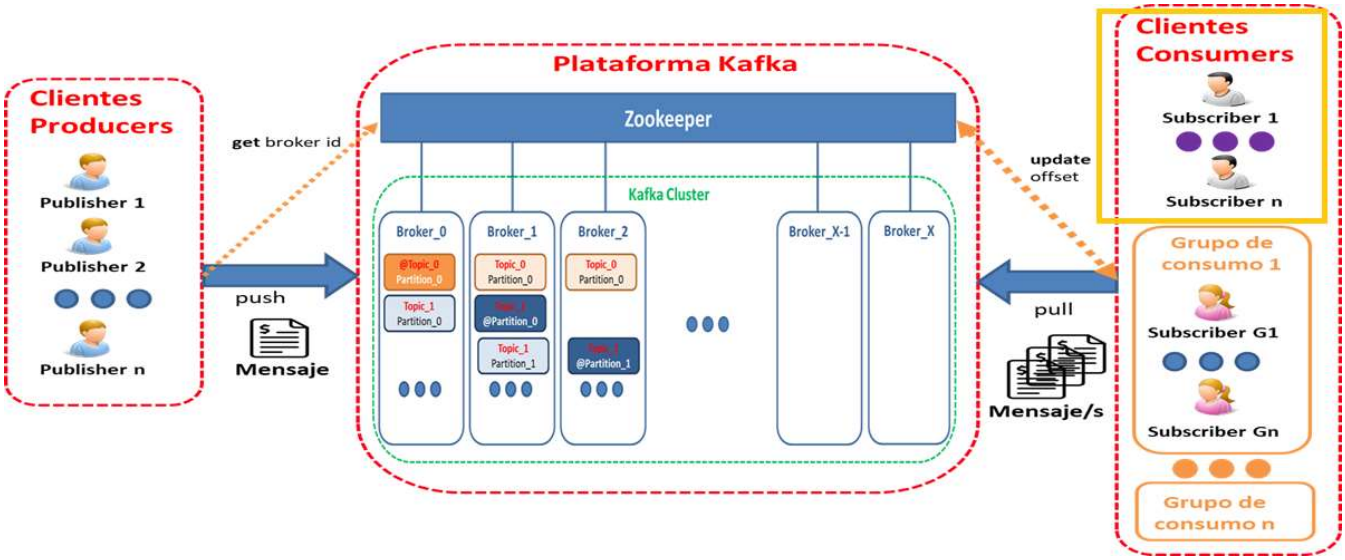
Sobre todo hay que tener en cuenta el siguiente aspecto: Tamaño del Fetch

Funcionamiento

- ***Lee** los **mensajes** de un **topic/s** en el **orden** en el que se han producido*
- *Cuando un consumidor conecta a un topic conecta a un broker leader*
- *Para la lectura utiliza el offset de la partición asignada -> le indica el mensaje a leer*

- *Se almacena el offset del último mensaje consumido para cada partición, así un consumidor puede detenerse y reiniciar sin perder su lugar*
- *Hace un seguimiento de los mensajes que ya se han consumido mediante el control de la compensación "offset" de los mensajes*
- ***IMPORTANTE :** Los consumidores no pueden leer mensajes no replicados -> el offset de mensajes menor o igual que el offset "High Watermark"*
- *Cuando procesa los datos debería de confirmar el offset*
- *Notifican al broker cuando procesa con éxito un registro y avanza el offset*
- *Si falla antes de enviar el offset de commit al broker entonces el consumidor puede continuar desde el ultimo offset comprometido*
- *Si falla después de procesar el registro pero antes de enviar el commit al broker, entonces algunos mensajes pueden ser reprocesados*
- ***At least one :** los mensajes nunca se pierden pero quizás existan duplicados*
- ***At most one :** los mensajes son perdidos pero nunca habrá duplicados*
- ***Exactly one :** los mensajes sólo son entregados una y sólo una*
- *Comprobación de que los mensajes (registro de entregar) son idempotentes*
- *Se puede ejecutar más de un consumidor en un proceso JVM usando threads /hilos -> Cada uno en su propio hilo*
- *Los mensajes no son quitados de la partición después de ser procesados a menos que exista algún criterio para hacerlo*

Diagrama Conceptual : "Esquema de la plataforma en lo relacionado con consumidores"



Grupo de consumidores

Definiciones

- ***Agrupación de uno o más consumidores** que trabajan para leer de un topic en base al cumplimiento de algún objetivo / funcionalidad concreta.*

Características

- *Cada grupo se considera un "suscriptor" (como si fuera un sólo consumidor)*
- *Puede ser un subscriptor de uno o más topic*
- *Múltiples suscriptores = múltiples grupos*
- *Cada grupo dispone de su propio offset único por partición del topic*
- *Un grupo pueden leer desde diferentes ubicaciones en una partición*
- *Diferentes grupos de consumidores pueden leer desde diferentes ubicaciones en una partición*
- *Los consumidores de un grupo mantienen un balance de carga uniforme*
- *Múltiples grupos de consumo pueden leer desde diferentes particiones de forma eficiente*

Configuración

- ***Identificador:***
- *"Nombre" único para distinguirlo del resto de grupos*
- *Cada consumidor puede tener un identificador de grupo*

Nota : los parámetros de configuración se detallarán en los ejercicios prácticos

Funcionamiento

Consumo de mensajes :

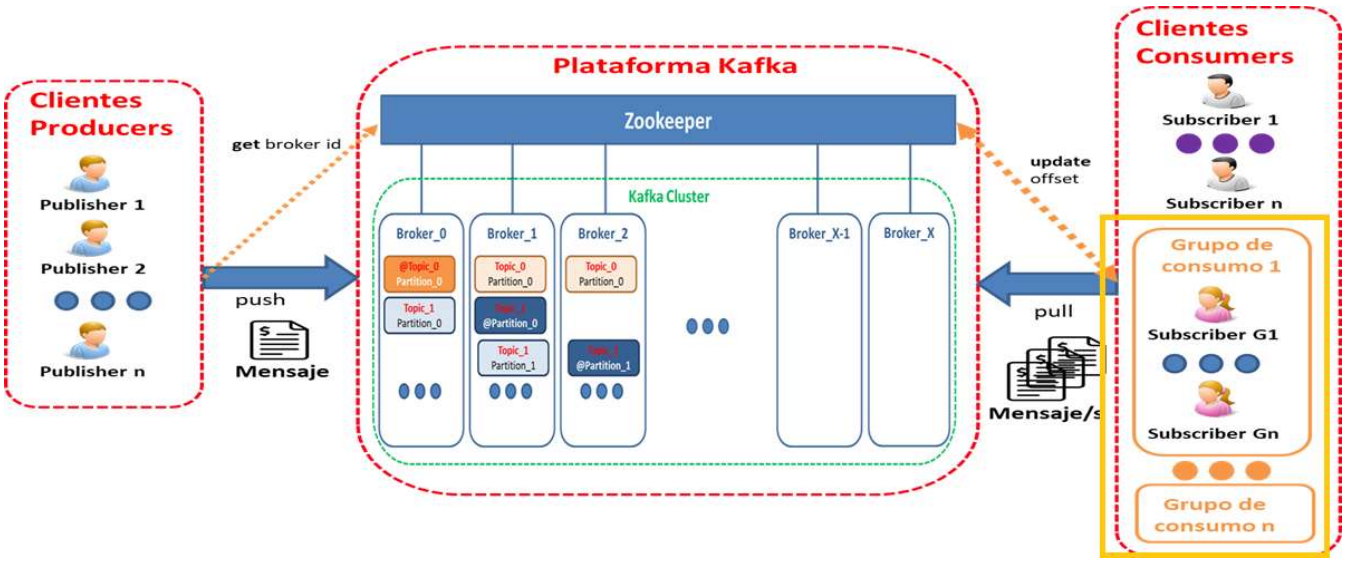
- *Un mensaje es entregado a un único consumidor en un grupo de consumo*
- *El grupo asegura que cada partición sólo sea consumida por un miembro*
- *Cuando procesa un mensaje debería hacer un commit del offset*
- *Si el consumidor muere, podrá iniciarse y comenzar a leer desde donde se quedó en función del offset almacenado en "_consumer_offset" o según lo analizado, otro consumidor del grupo de consumidores puede hacerse cargo*
- *Sólo un único consumidor del mismo grupo de consumidores puede acceder a una sola partición -> múltiples consumidores puedan leer cualquier flujo de mensajes sin afectarse/interferirse*
- *Cada consumidor en el grupo es un consumidor exclusivo de una parte "justa"/"apropiada" de las particiones*
- *Si un nuevo consumidor entra en el grupo entonces recibe una parte de las particiones*
- *Si un consumidor muerte , sus particiones se dividen/reparten entre los consumidores vivos "restantes" en el grupo*
- *Los consumidores de un grupo balancean la carga al procesar registro*
- *Un solo consumidor en el grupo se convierte en el coordinador del grupo*

- *Si un consumidor falla antes de enviar el offset de commit al broker, entonces un consumidor diferente puede continuar desde el último offset confirmado*
- *Si un consumidor falla después de procesar un mensaje pero antes de enviar el compromiso al broker, entonces algunos mensajes pueden ser reprocesados*
- *Si el procesamiento de un registro tarda un tiempo, un solo consumidor puede ejecutar múltiples subprocesos para procesar registros,pero es más difícil gestionar la compensación para cada subproceso/tarea.*
- *Si un consumidor ejecuta múltiples subprocesos, entonces dos mensajes en las mismas particiones podrían ser procesados por dos subprocesos diferentes, lo que dificulta garantizar el orden de entrega de mensajes sin una coordinación de subprocesos compleja.*
- *Esta configuración puede ser apropiada si se está procesando una sola tarea lleva mucho tiempo, pero trata de evitarla*
- *Si necesita ejecutar múltiples consumidores, entonces se ejecuta cada consumidor en su propio hilo. De esta manera, Kafka puede entregar los grupos de mensajes al consumidor y éste no tiene que preocuparse por solicitar el offset. Un hilo por consumidor facilita la gestión de las compensaciones. También es más sencillo gestionar la conmutación por error (cada proceso ejecuta X número de hilos de consumo), ya que puede permitir que Kafka haga el trabajo más duro.*

Tipología :

- ***Cola Tradicional :*** *Todos los consumidores pertenecen a un sólo grupo*
- ***Difusión :*** *Todos los consumidores pertenecen a diferentes grupos*
- ***Subscripción Lógica :*** *Muchos consumidores son instancias de un grupo*
- Consideraciones :
- *No puede haber más consumidores que instancias*
- *Cada consumidor lee una o más particiones de un topic*
- *Si el nº de grupos de consumidores supera del nº de particiones, entonces los consumidores exceden el número de particiones, por lo que los consumidores adicionales estarán inactivos.Estos consumidores inactivos estarán pendientes por si se produce un fallo*
- *Si hay más particiones que el nº grupo de consumidores, entonces algunos consumidores leerán desde más de una partición*

Diagrama Conceptual : "Esquema de la plataforma en lo relacionado con grupos de consumo"



2. Conclusiones

Se acaban de enseñar las piezas que más "quebraderos de cabeza" nos darán a la hora de trabajar con la plataforma Kafka y que serán las encargadas de definir nuestra arquitectura así como implementar la funcionalidad requerida. Por lo que si NO quieres ir perdido al inicio de cualquier desarrollo, será muy importante tener muy claro algunas cuestiones :

- ¿Cómo se deberían de enfocar ese patrón Publish/Subscribe para el caso de uso considerado?
- ¿Si va a ser necesario incorporar alguno o varios patrones EIP (Enterprise Integration Patterns) extras? ("No reinventar la rueda")
- ¿Qué es lo que vamos a enviar o con lo que estamos trabajos?
- ¿Cuánto va a ocupar en "bytes"?
- ¿si se va a seguir un enfoque de "topic" o bien de "contenido" en lo referente al enrutamiento lógico?
- ¿Si se requiere algún tipo de ordenación en la recepción de las comunicaciones?
- ¿Si se requiere algún tipo de estrategia de entrega (garantía) ?
- ¿Qué tipo de persistencia tendrá la información dentro de la plataforma?
- ...

Todas estas preguntas y muchas más nos ayudarán a acertar con la "mejor" solución para la funcionalidad requerida.

En este artículo finalizo toda la literatura , y en los próximos artículos por fin empezaremos con la parte práctica, donde en algunos casos será sobre lo que se ha enseñado en los artículos realizados hasta la fecha y en algunos casos se matizará con nueva información.

Ya estamos a un pasito menos de ser unos verdaderos "Kafka Developers" ;-)

¡Síguenos en Twitter para estar al día de próximas entregas de la serie!

Autor

VÍCTOR MADRID

Líder Técnico de la Comunidad de Arquitectura de Soluciones en atSistemas. Aprendiz de mucho y maestro de nada. Técnico, artista y polifacético a partes iguales ;-)

COMPARTE

ALSO ON **EN MI LOCAL FUNCIONA**

Phaser 3: Mi primer juego HTML5

hace 2 años • 1 comentario

Introducción ¿Qué programador no ha soñado con participar en la …

SonarQube: Ejecución de análisis e …

hace 2 años • 1 comentario

En el anterior post (Cómo montar un SonarQube en cloud que nos sirva de …

Observabilidad usando OpenTelemetry

hace 5 meses • 1 comentario

OpenTelemetry es el nuevo estándar de observabilidad avalado por la CNCF. …

R **Raquel**

hace 4 años

Enhorabuena Victor, muy buen articulo.
Me gustaría saber si el conector debe estar instalado en todos los brokers que formen parte de nuestro clusterde Kafka, y que se vayan a comunicar con una instancia única de Kafka Connect, que se encuentra aparte de cluster de Kafka.

Gracias de antemano.

Jesus J. Puente

hace 4 años

Buen y detallado articulo. Deseando que salga el siguiente con ejemplos practicos