Apache Kafka

Borja Moreno Pozo

INDICE

- ¿Ques es Apache Kafka?
- Arquitectura
- Ejemplo

Kafka

- “A high-throughput distributed messaging system”
- “Apache Kafka is publish-subscribe messaging rethought as a distributed commit log”
- Sistema distribuido de mensajes
- Desarrollado originalmente por Linkedin (Jay Kreps)

Características

- **Rápido**. Un único bróker de Kafka puede gestionar cientos de megabytes en lectura y escritura para miles de clientes
- **Escalable**. A través de un cluster escalable en tiempo. Los mensajes son particionados en el cluster
- **Durable**. Los mensajes son persistidos y replicados en el cluster
- **Distribuido**. Diseñado conceptualmente como un componente distribuido que garantiza durabilidad y tolerancia a fallos del propio sistema

Casos de uso

- Paso de mensajes
- Seguimientos de actividad
- Monitorización de métricas
- Agregación de logs
- Procesamiento de streams
- Fuente de datos
- Commit log

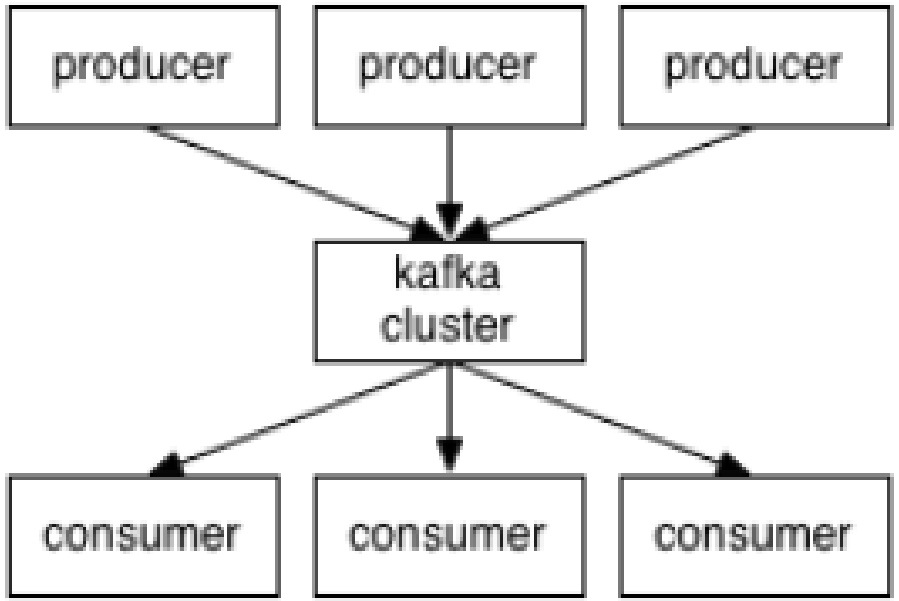
Conceptos básicos

- Kafka almacena los mensajes en categorías llamadas ***topics***
- Los mensajes son generados por procesos llamados ***producers***
- Los procesos que leen los mensajes generados por los producers se llaman ***consumers***
- Los mensajes dentro de un cluster de Kafka serán gestionados por uno o más servidores llamados ***brokers***
- La comunicación entre los distintos componentes de un cluster Kafka se realiza a través de un protocolo binario sobre TCP

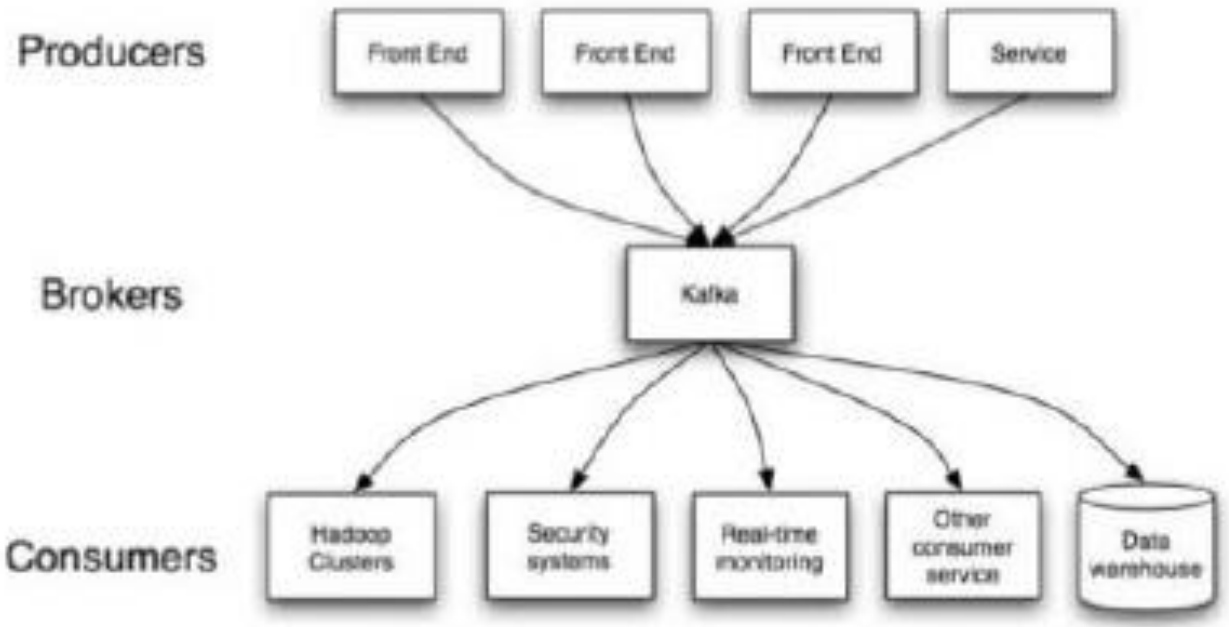
Componentes

- **Topic**: Un topic es la forma de categorización de los mensajes que tiene Kafka. Es donde son publicados los mensajes desde los producers. Son los elementos que permiten el particionado.
- **Broker**: Un cluster de Kafka esta formado por uno o mas servidores y cada servidor puede tener corriendo varios procesos de Kafka a su vez. Cada uno de estos procesos es lo que llamamos bróker
- **Zookeeper**: Funciona como coordinador para los sistemas distribuidos
- **Producers**: Publican los datos en los topics, seleccionando la partición adecuada dentro de un topic. Pueden implementarse desde cualquier lenguaje de desarrollo.
- **Consumer**: Son los componentes que se suscriben a un topic para consumir los mensajes. Pueden implementarse desde cualquier lenguaje de desarrollo.

Conceptos básicos



Conceptos básicos



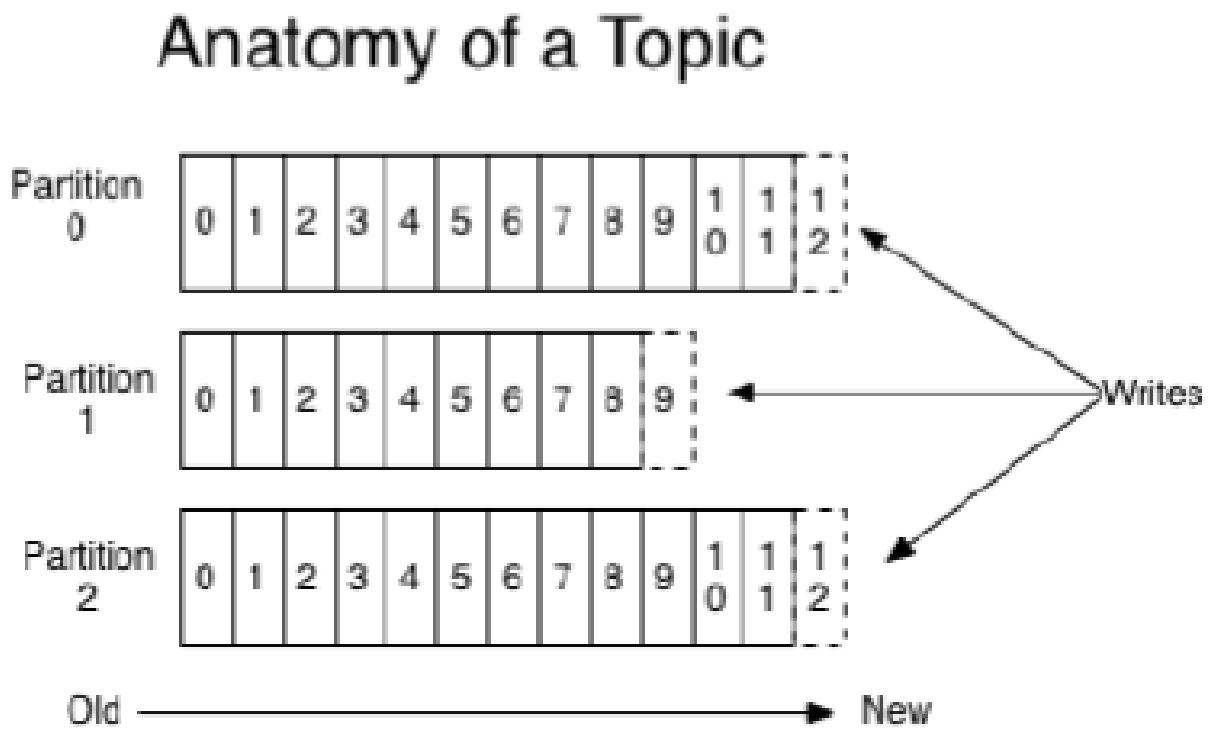
Kafka. Topics

- Un topic es una categoría donde los mensajes (para dicha categoría) son publicados por los productores
- En Kafka cada topic tiene asociado un log particionado
- Se definen un número determinado de particiones para cada topic.
- Cada partición es una secuencia de mensajes inmutables y ordenada que se van añadiendo continuamente
- Cada partición debe ser almacenada en los servidores que la mantengan
- Las particiones pueden ser replicadas

Kafka. Topics

- Los mensajes almacenados en las particiones están ordenados por un identificador numérico secuencial denorminado offset
- Los consumidores solo necesitarán la información del offset para ir leyendo de Kafka
- Para reprocesar los consumidores ‘retrocederán’ en este offset
- Kafka retiene todos los mensajes publicados hayan sido o no consumidos
- El tiempo de retención es configurable mediante la propiedad de log retention
- Si el log retention está definido a 1 semana los mensajes estarán disponibles para ser consumidos durante esa semana. Después serán borrados

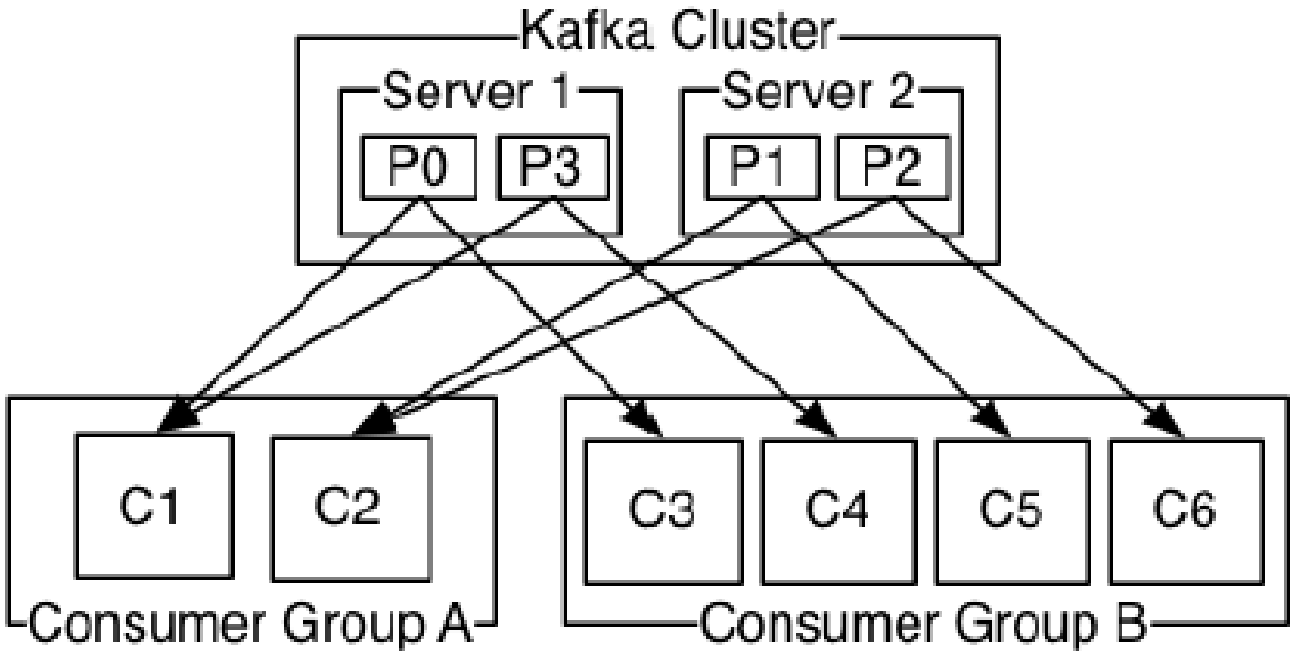
Kafka. Topic


Anatomy of a Topic
Partition 0
0 1 2 3 4 5 6 7 8 9 10 11 12
Partition 1
0 1 2 3 4 5 6 7 8 9
Partition 2
0 1 2 3 4 5 6 7 8 9 10 11 12
Writes
Old
New

Kafka.Topics. Escritura y administración

- Los producers eligen sobre que partición desean escribir los mensajes:
  - Roun-robin
  - Semántica de partición
    - Keyed messages
    - Temporal
- Cada partición tiene un server que actúa como leader y varios que actúan como followers
- El leader gestiona todas las lecturas y escrituras. Los followers replican los mensajes insertados

Kafka.Topics. Consumo



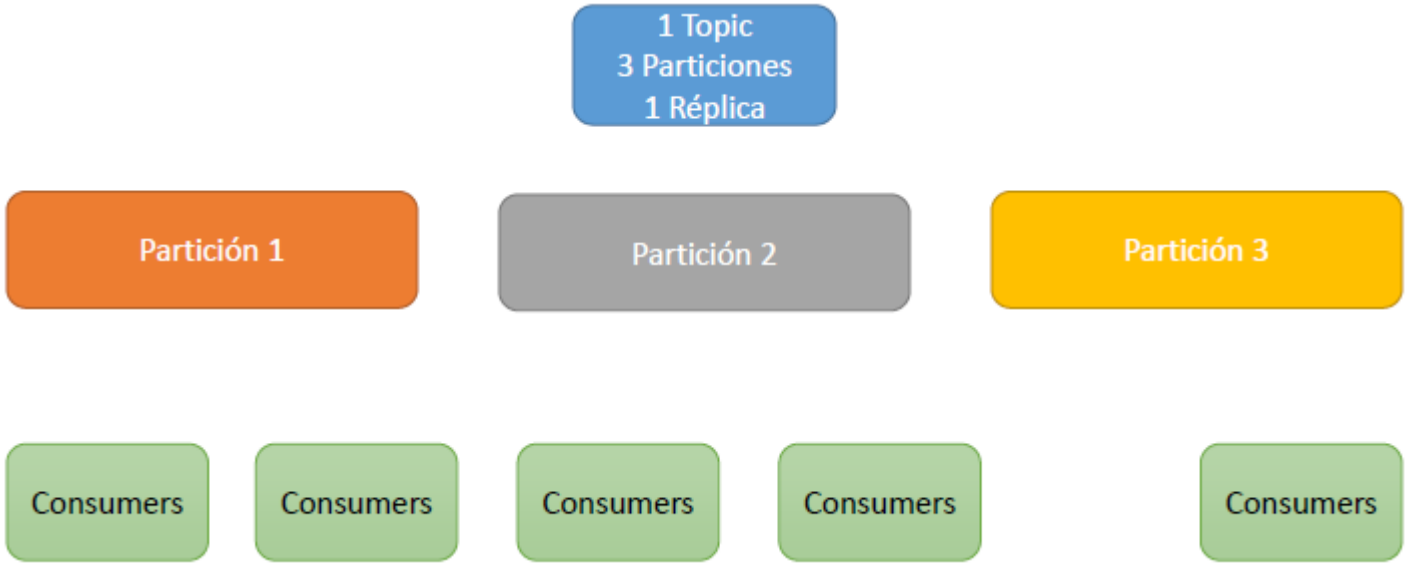
Kafka.Topics. Escritura y lectura

- Kafka garantiza la ordenación de los mensajes
- Cada partición en el consumer group es asociada a un único consumer garantizando la lectura ordenada de los mensajes
- El sistema sigue siendo paralelizable porque el resto de particiones pueden ser consumidas por otros consumidores
- Kafka por lo tanto garantiza ordenación de mensajes a nivel de partición, no entre diferentes particiones de un topic

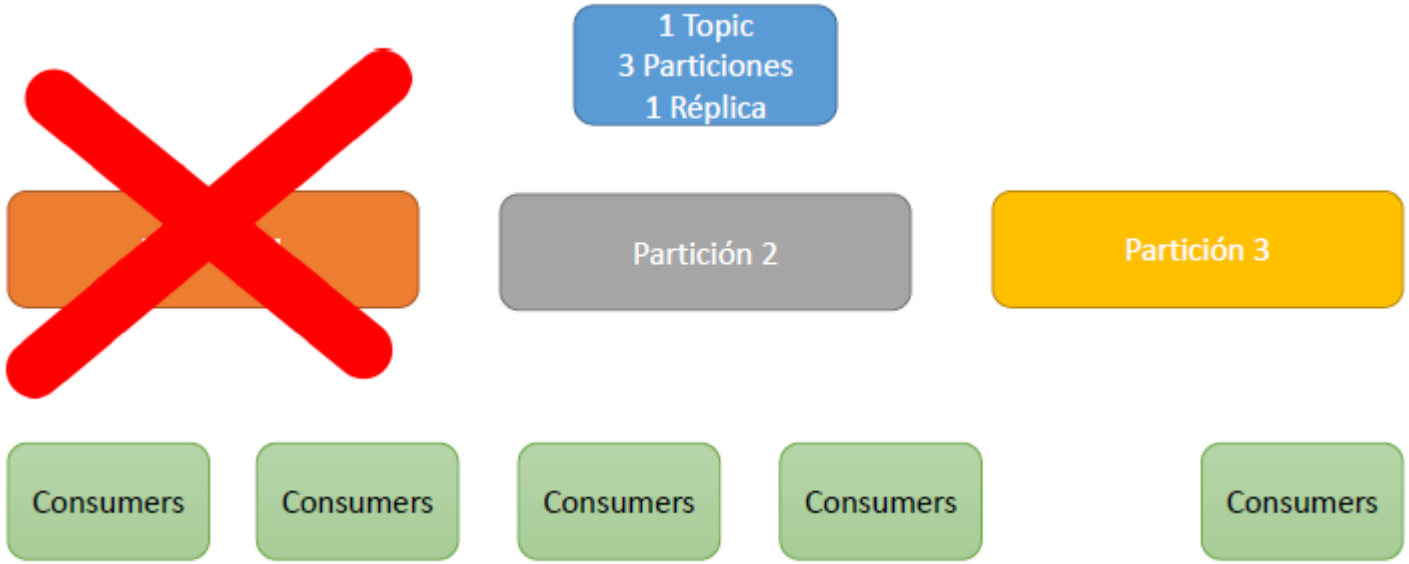
Kafka.Tolerancia a fallos

- Kafka garantiza:
  - Los mensajes mandados por un productor a una partition determinada serán añadidos en el orden que hayn sido enviados
    - Si el mensaje M1 es enviado antes que el mensaje M2, el primero será almacenado antes (menor offset) dentro de la partición
  - El consumidor verá los mensajes en la partición en el orden que hayan sido almacenados
  - Para un topic con un factor de replicación N, se tolerará hasta N-1 fallos de servidores sin perdida de los mensajes almacenados

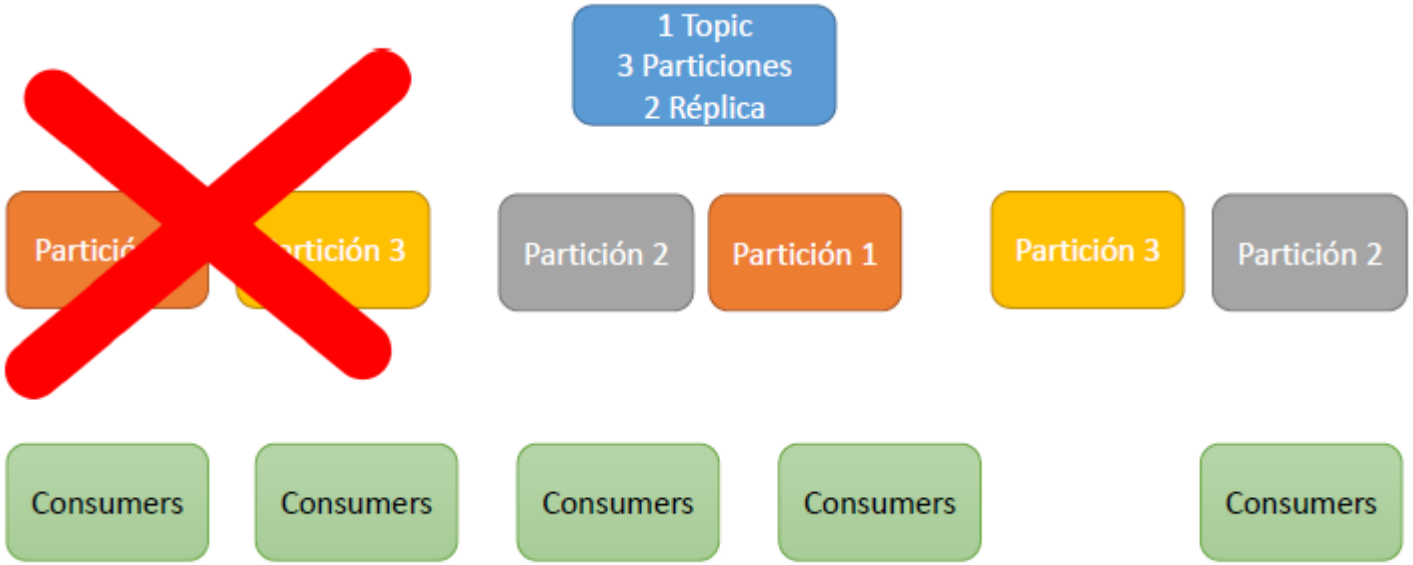
Kafka.Tolerancia a fallos



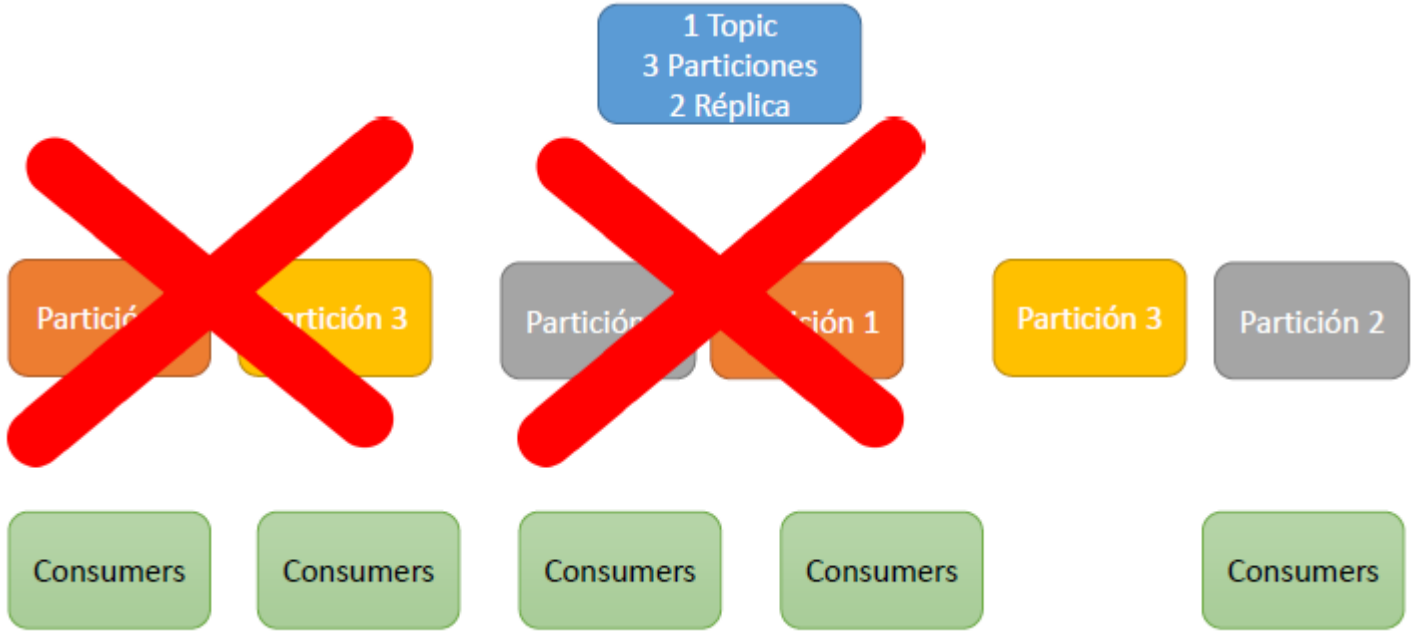
Kafka.Tolerancia a fallos



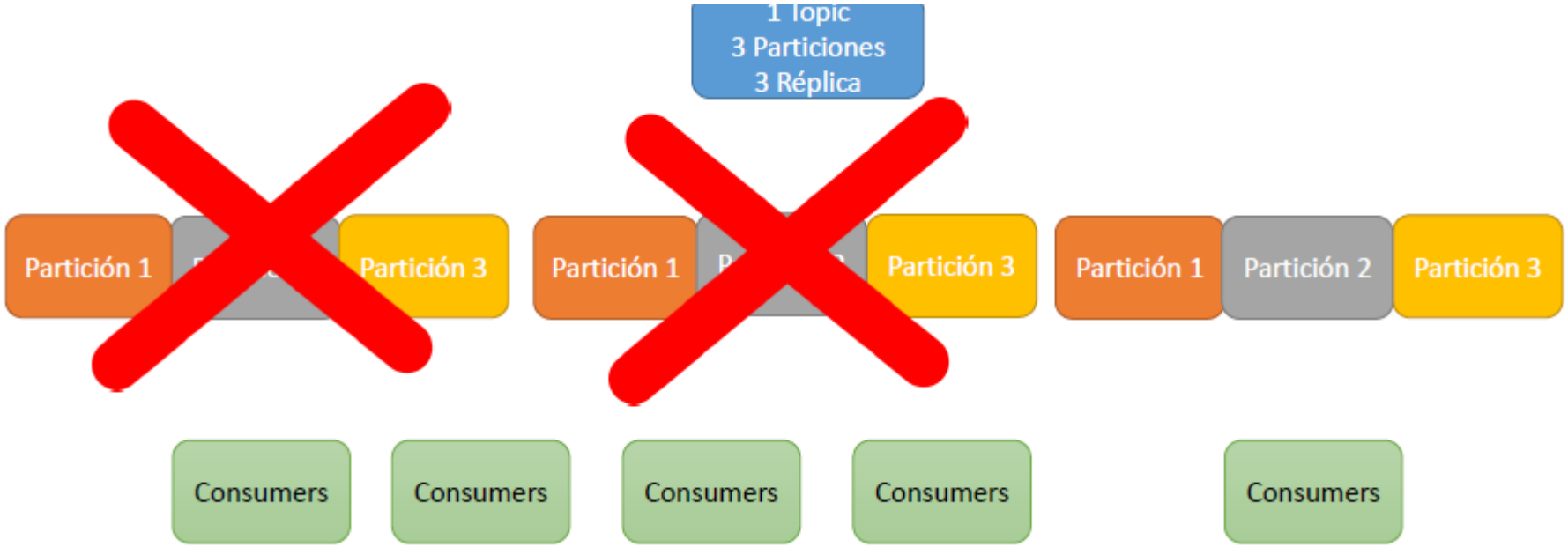
Kafka.Tolerancia a fallos



Kafka.Tolerancia a fallos



Kafka.Tolerancia a fallos



Kafka.Connect

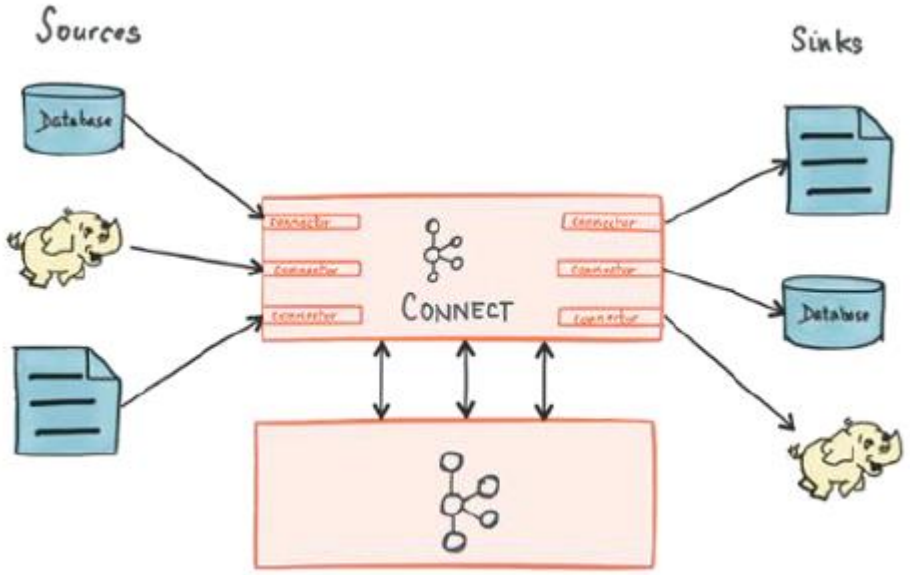
- Permite definir conectores sobre fuentes externas a Kafka para leer y/o escribir sobre ellas
- Características:
  - Ingesta de datos en topics de Kafka
  - Framework común
  - Modo distribuido y standalone
  - Interfaz REST
  - Gestión automática de los offsets
  - Distribuido y escalable por defecto
  - Valido para procesamiento batch y streaming

Kafka.Connect configuración

- Configuracion de un cluster Kafka Connect:
  - Group-id: nombre del cluster
  - Config.storage.topic: topic utilizado para guardar la información del conector y las tareas provenientes de la fuente
  - Offset.storage.topic: permite guardar los offset
  - Status.storage.topic: permite guardar el status
- La información se guarda en topics

Kafka.Connect configuración

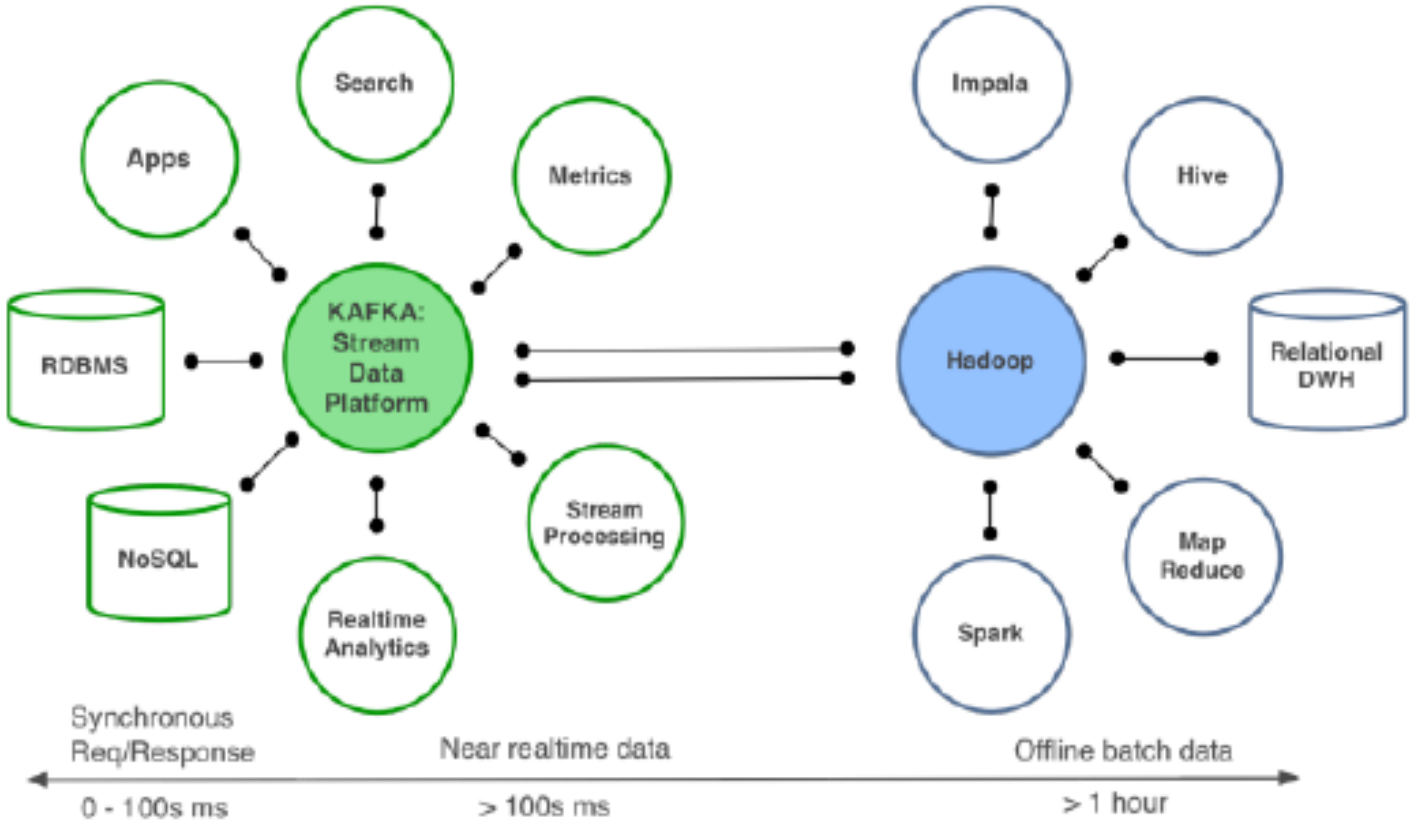
- Configuración de un conector
  - Name: nombre único para cada conector
  - Connector.class: clase Java para el conector. Sigue un modelo de Sources and Sinks similares a Flume
  - Task.max: Número de tareas máximas definidas para este conector. Las tareas son las que se encargan de ir leyendo/escribiendo de/hacia la fuente de datos
  - Topics: lista de topics usadas por el conector como entrada



Kafka.Kafka streams

- Librería Kafka para procesamiento en streaming
- A partir de la versión 0.10.0.0
- Fácil de utilizar
- Permite leer y escribir sobre distintas fuentes (incluida la propia Kafka)
- Kafka delega en su modelo de particiones para alcanzar mayot grado de paralelismo

Kafka.Kafka streams



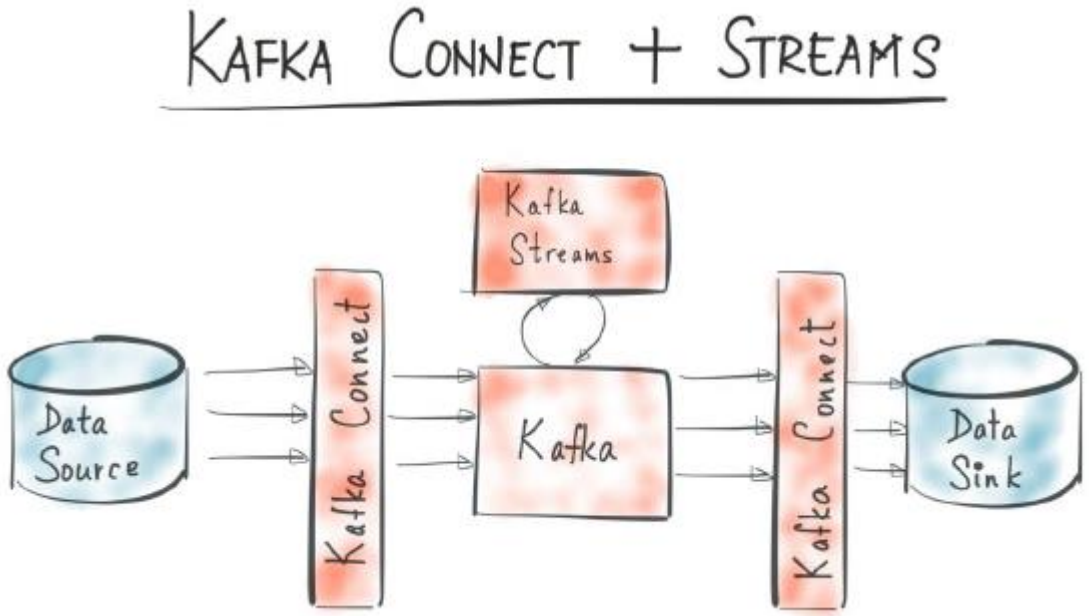
Kafka.Kafka streams

- Características
  - Cliente ligero
  - No tiene dependencias con ningún sistema externo a Kafka
  - Soportar tolerancia a fallos en local, lo que permite realizar de forma eficiente operaciones como joins o agregaciones por ventana
  - Procesamiento one-record-at-a-time. ¿Qué significa esto?
  - Ofrece primitivas de procesamiento propias
  - Ofrece DSL propia de alto nivel (High-Level Streams DSL)
  - Se puede implementar una API de procesamiento Low Level

Kafka.Kafka streams topología

- Características:
  - Un stream representa un conjunto de datos (data récords) continuos unbounded ordenado, tolerante a fallos y que se puede volver a reprocesar
  - Un data record contiene la información del evento y es definido por una pareja clave-valor.
  - La lógica de cálculo en Kafka Streams viene definida por una topología donde se recibe un data record en cada momento que es procesado hacia abajo por la topología.
  - Un stream processor es una nodo de procesamiento de la topología. Representa cada paso en la transformación del data record original. La salida de cada stream processor devolverá uno o más registros que continuarán su camino por la topología definida.

Kafka.Integración Kafka connect y streams



Kafka.Persistencia

- Kafka confía en el sistema de fichero para almacenar y cachear mensajes
- Más rápido → utilizar memoria
- Pero, si utilizamos lecturas y escrituras secuenciales en disco puede ser más rápido que accesos aleatorios a memoria
- Además, en Kafka se intenta mandar a un log persistente que a su vez almacena la información en el pagecache del S.O. → Mejor rendimiento

Kafka.Persistencia

- Tradicionalmente en sistemas de mensajería los mensajes son almacenados en una estructura Btree
  - Ventajas: complejidad ‘constante’ O(log N)
  - Inconvenientes: sólo se puede hacer una lectura en disco a la vez para leer una estructura → Difícil de paralelizar si no metemos más discos.
- Solución: Implementar un commit log donde las operaciones tienen complejidad O(1) y no son bloqueantes unas con otras

Kafka.Rendimiento

- Problemas usuales en acceso a disco:
  - Operaciones de E/S demasiado pequeñas
  - Demasiada copia de datos en las operaciones

- Soluciones:
  - Se agrupan las operaciones a nivel de granularidad de mensaje. Se optimiza el overhead de red a la hora de leer/escribir
  - Formato de mensaje optimizado y llamadas al sistema reducidas para copiar datos

Kafka.Replicación (Quorums, ISRs, …)

- Log replicada en distintos ficheros en disco
- Existe un leader por partición que se encarga de gestionar lecturas y escrituras
- Kafka mantiene de manera dinámica un conjunto de replicas sincronizadas (ISR)
- Dichas replicas son seleccionables para ser elegidas como leaders
- La escritura de un mensaje en una partición sólo es considerada comiteada cuando todas las réplicas están sincronizadas
- El conjunto de réplicas sincronizadas (ISR) son almacenadas en Zookeeper

Kafka.Replicación (Quorums, ISRs, …)

- Si una replica se cae, Kafka se encarga que antes de unirse al conjunto ISR, se vuelva a sincronizar
- Dadas f+1 réplicas un topic de Kafka tiene tolerancia a fallos f sin pérdidas de mensajes comiteados
- ¿Qué pasa si todas las réplicas mueren?
  - Se perderá servicio hasta que la primera réplica (que no tiene que estar en el grupo ISR) se recupere
  - Potencialmente esto genera problemas de inconsistencia
  - Si se quiere evitar problemas de inconsistencia se puede indicar que Kafka espere a que una réplica del ISR se recupere

Kafka.Log compaction

- Permite que Kafka retiene al menos el último valor para una clave de mensajes asociada a cada partición de un topic
- Permite a los consumidores tener una imagen del último valor de los mensajes almacenados sin necesidad de recuperar todo el estado de los mensajes
- Granularidad a nivel de topic
- Uso: se quiere recuperar el último estado de una aplicación que ha fallado previamente

Kafka.Log compaction

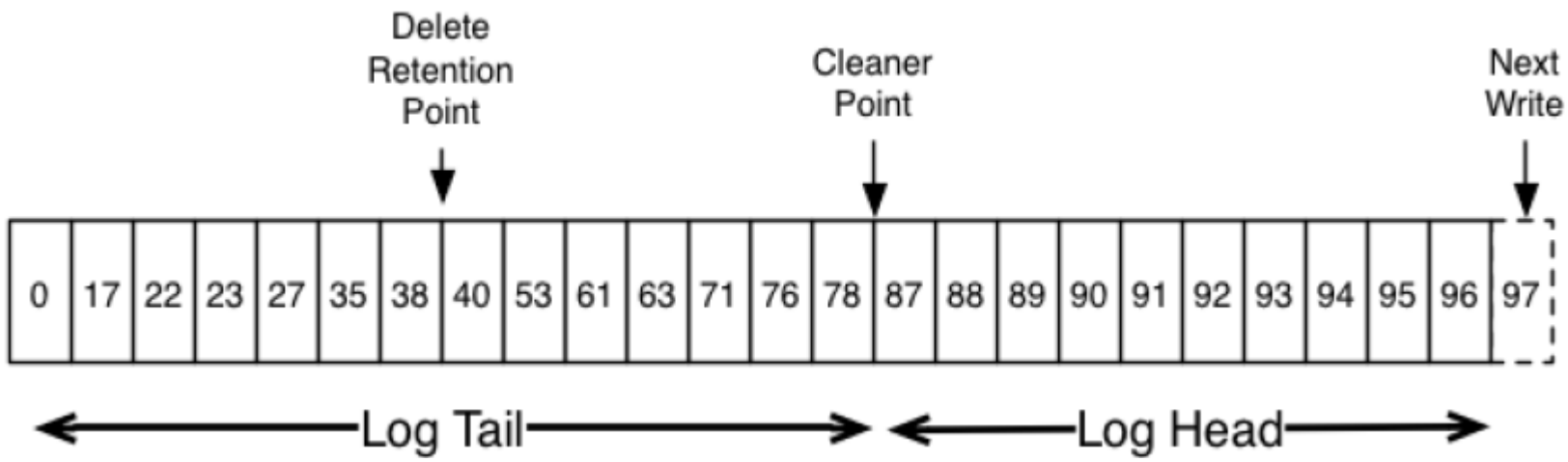
```
123 => bill@microsoft.com
          .
          .
          .
123 => bill@gatesfoundation.org
          .
          .
          .
123 => bill@gmail.com
```

Kafka.Log compaction

- Opción de tener la cola (tail) del log gestionada de manera diferente a la cabeza (head) → **clearner point**
  - Los mensajes de la cola conservan el offset original

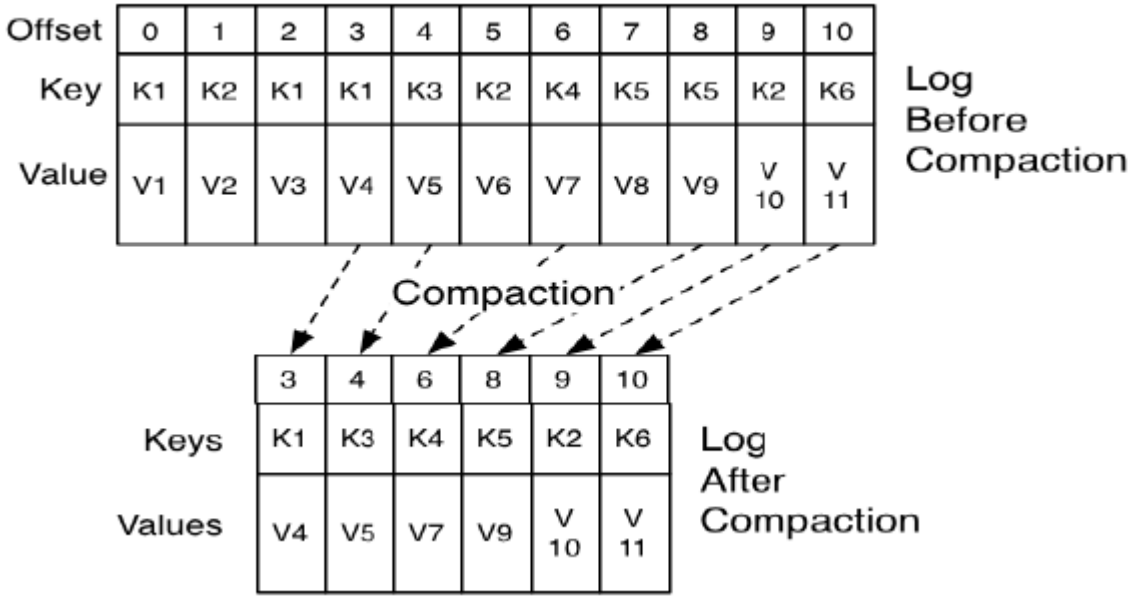
- Opción de indicar un punto de borrad a partir de una key (los mensajes con un offset inferior se borraran a partir de un tiempo) → **Delete Retention Point**

Kafka.Log compaction



Kafka.Log compaction

- La compactación se realiza en background copiando periódicamente segmentos



Kafka.Seguridad

- Autenticación utilizando SSL o Kerberos desde los brokers a los clientes
- Autenticación de conexiones desde los brokers a Zookeeper
- Encriptación de datos
- Autorización de lecturas/escrituras por los clientes
- Autorización plugable con servicios de terceros