Kafka
Organización

Organización general

Zookeeper

Orquestador de alto rendimiento para aplicaciones distribuidas.

Gestiona los clusters.

Requiere ser instanciado como ‘Entry point’ (Primera instancia).

Intercambia datos con los brokers, productores y consumidores.

Responsable de la selección del ‘broker leader’ donde guarda la ‘partición leader’.

Zookeeper

Su esquema resulta como una horizontal a toda la arquitectura de **Apache Kafka**.

Broker

Es el encargado de instanciar un servidor en **Kafka**.

Gestiona las peticiones de entrega de mensajes.

Contiene las particiones de los tópicos.

Se comunica continuamente con el Zookeeper.

Normalmente acoge un promedio menor de 2000 particiones.

Broker - Escritura

Recibe las peticiones de escritura de los mensajes productores.

Determina el ‘offset’ a utilizar.

Confirma los mensajes.

Actualiza las réplicas.

Broker - Lectura

Recibe las peticiones de lectura de los mensajes consumidos.

Determina el ‘offset’ a utilizar.

Broker - Tolerancia a fallos

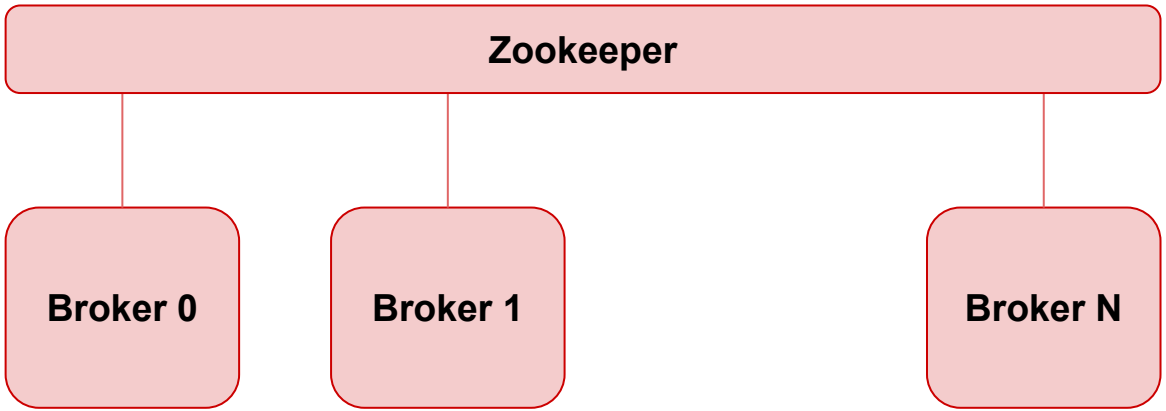
Si la ‘partición leader’ se cae, automáticamente se asigna otro broker.

La partición réplica se convierte en el nuevo ‘leader’.

Uso de ‘in-sync replica’.

Broker

Su esquema resulta como nodos que caen de la horizontal del **Zookeeper**.



Cluster

Permite agrupar ‘N’ **Brokers** siendo N>=1.

Necesidad de convivencia con **Apache Zookeeper**.

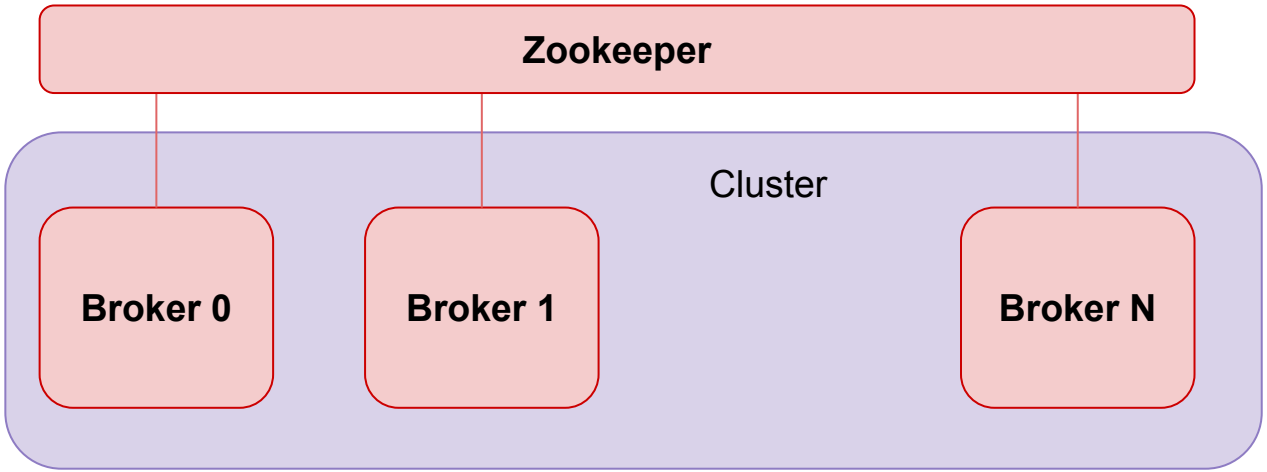
Posibilidad de agregar ‘N’ **clusters** :

- Segregación de datos
- Seguridad
- Recovery

Necesidad de elegir al menos 1 **cluster** principal siempre que N>1.

Cluster

Su esquema resulta como un agrupador de **brokers** por debajo de **Zookeeper**.



Mensaje

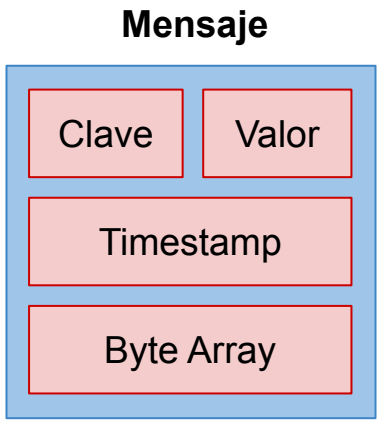
Unidad de datos que trabaja a nivel de byte arrays.

Estructura interna:
- Clave/Valor
- Timestamp

Contiene diferentes estados:
- Confirmado
- No confirmado

Mensaje

Su esquema resulta de la siguiente manera:



Tópicos

Categoría de clasificación para ‘N’ mensajes siendo ‘N’ >= 1.

Almacena de manera permanente los mensajes.

Necesidad de al menos un broker.

Permite el uso de un productor.

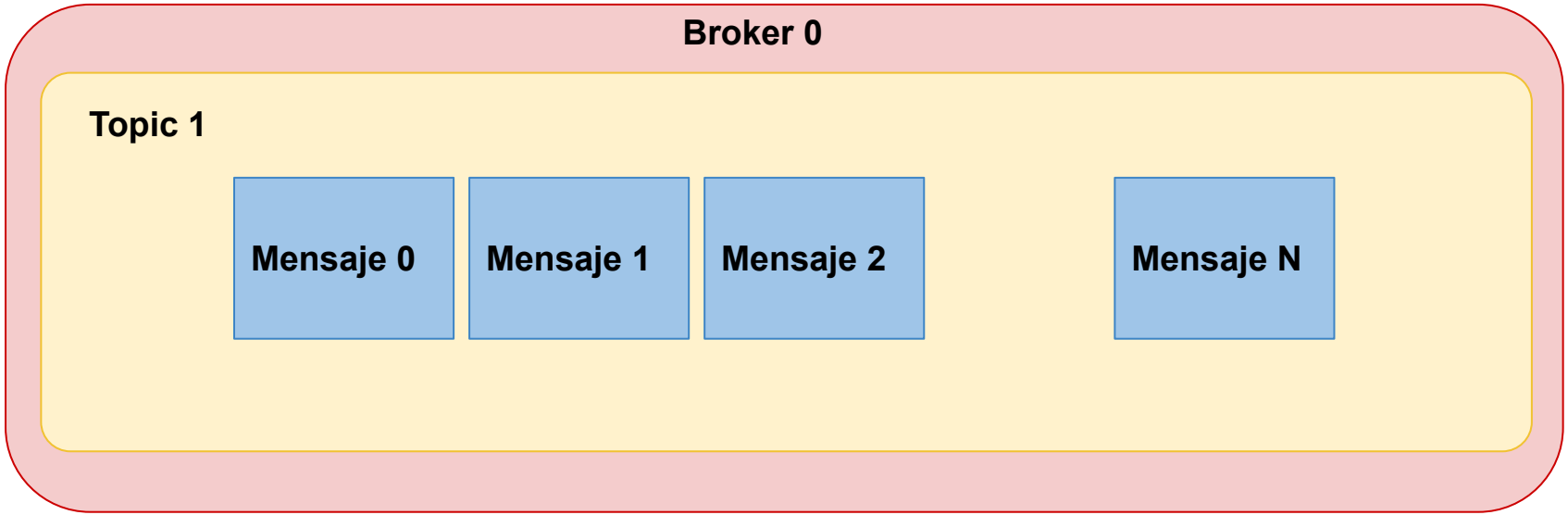
Puede tener 0 ó ‘N’ consumidores.

Al menos debe de tener 1 partición internamente.

Ordenamiento del más viejo al más nuevo.

Tópicos

Dentro de un **broker**, se representa como una ‘carpeta’ de mensajes.



Partición

Secuenciación de mensajes ordenados de manera inmutable.

Unidad de replicación del **tópico**.

Facilita la redundancia de mensajes a través de las réplicas.

Se añaden los mensajes de manera ‘append-only’.

Asigna un offset automáticamente a todas las **particiones** cuando se inserta un mensaje.

Permite la paralelización de los **consumidores**.

Partición

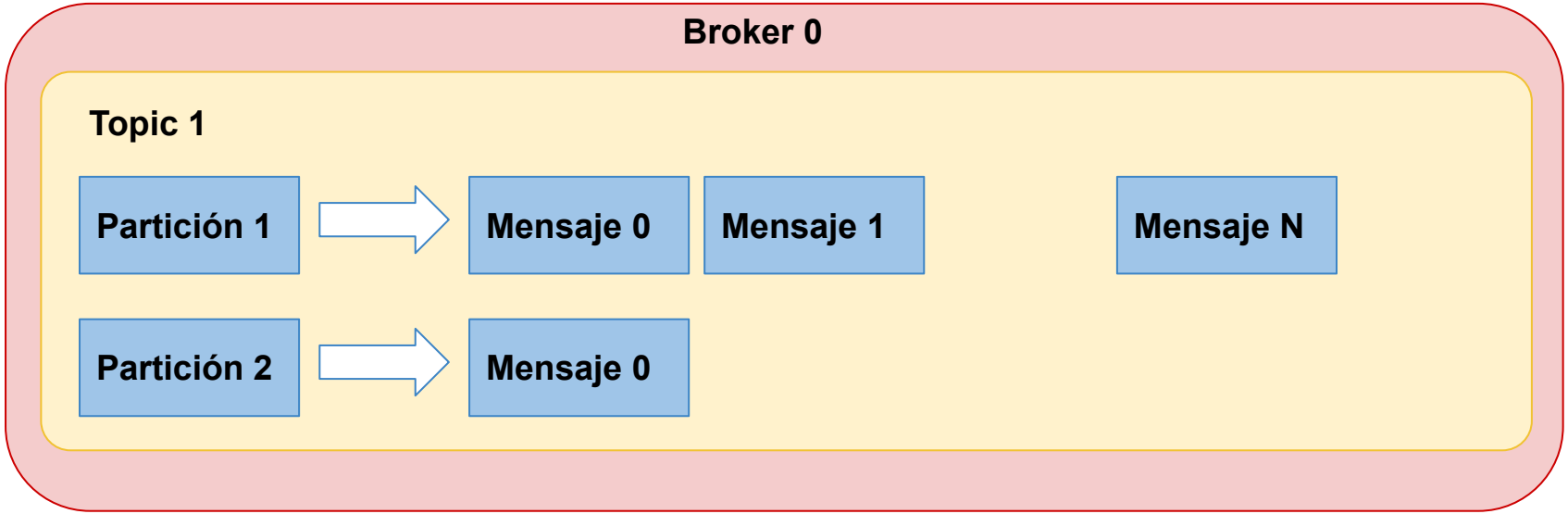
Necesidad de al menos un **broker**.

Un **tópico** definido.

Pueden convivir diferentes **particiones** dentro de un tópico.

Partición

Dentro de una partición, se representa los mensajes con una ordenación inmutable.



Offsets

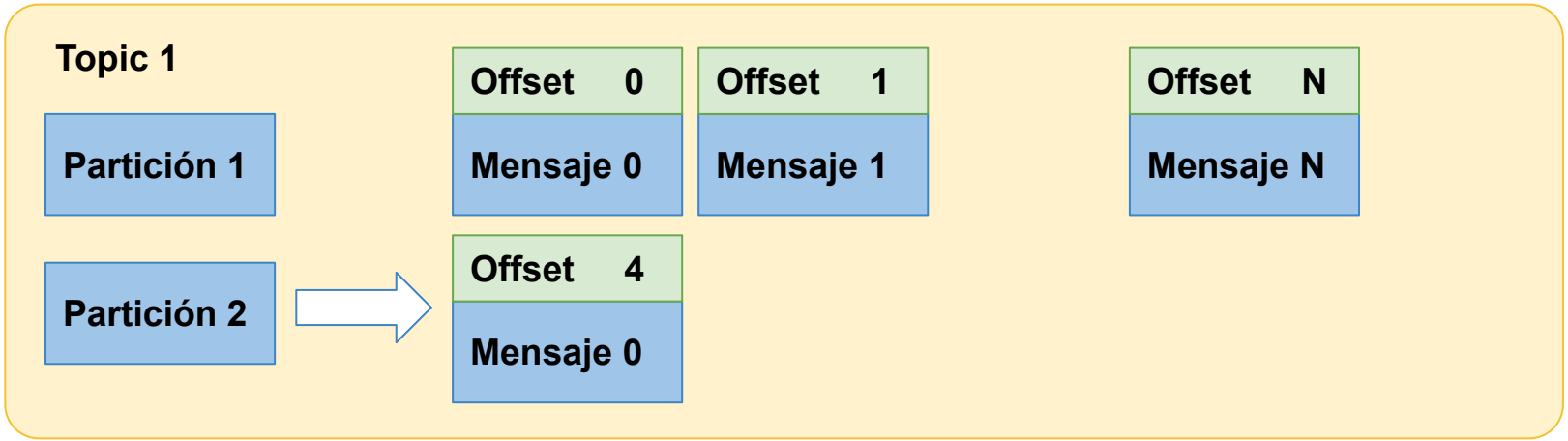
UUID de cada mensaje dentro de una **partición**.

Se asigna en los metadatos un valor entero incremental y secuencial.

Se indica la posición del **consumidor** dentro del **‘topic log’** o en una **partición**.

Offsets

Dentro de una partición, el índice del mensaje recibe un identificador llamado offset.
No tiene porque estar acorde con la posición del mensaje dentro de la partición.



Mensajes y Topics

Mensajes

¿Qué es?

- Unidad de datos con la que trabaja Kafka (sería la unidad de comunicaciones entre aplicaciones)
- Array de bytes
- Par (Clave[opcional],Valor) y un timestamp

Características

- No tiene una estructura, formato o significado específico para Kafka
    - Esto puede cambiar con el uso de esquemas

Mensajes

Características

- *Cada mensaje puede tener un bit opcional de metadatos que se denomina "clave"/"key"*
  - *No tiene un significado específico para Kafka*
  - *Puede facilitar la escritura controlada de mensajes en las particiones -> Criterio de asignación de partición*
    - *Asegura que mensajes con una misma clave se escribirán en la misma partición -> Determinista*
- *Permite la agrupación de mensajes en batches / lotes*
  - *Un batch es una colección de mensajes producidos por el mismo topic y partición*
  - *Mejora la eficiencia*
- *Se persiste dentro de una partición que pertenece a un topic*

Mensajes

Características

- *Cada record tiene un periodo de retención*
- *Tiene diferentes "estados" :*
  - ***CONFIRMADO :** cuando todas las particiones ISR (In-Sync Replica) tienen escrito el mensaje en su log*
  - ***NO CONFIRMADO :** cuando al menos una partición ISR no tiene escrito el mensaje en su log*

Topics

¿Qué es?

- Categoría o criterio con el que se pueden agrupar/clasificar los mensajes en Kafka -> Una colección de mensajes conforman un topic
- Stream de mensajes con nombre
- Se puede ver como un "contenedor" de mensajes

Características

- Pueden existir 1 o más topics -> No hay limitación
    - Kafka almacena los topics en logs

Topics

Características

- Cada topic tiene una configuración específica y por lo tanto una estructura independiente dentro de Kafka (esta información la obtiene del Zookeeper)
- Almacena de forma permanente los mensajes a menos que se establezca algún criterio de retención -> permite el trabajo asíncrono y no en tiempo real (no pierde datos)
- Se utiliza para la escritura y lectura de los mensajes
    - Un productor introduce el mensaje en este componente
    - Sólo tiene un productor cada vez
    - Un consumidor extrae el mensaje en este componente
    - Puede tener 0 o N consumidores
- Kafka dispone de una herramienta específica para su gestión

Productores, Consumidores y Brokers

Productores

¿Qué es?

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

Características

- 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

Consumidores

¿Qué es?

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

Características

- 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?

Brokers

¿Qué es?

- Instancia o un servidor de Kafka
- Cada uno de los nodos que componen la topología
- Mediador de las comunicaciones para la entrega de los mensajes

Características

- Contiene la/s particion/es de uno o varios topic/s
    - Cada broker puede contener la partición leader "cuando se le asigna" o bien una partición réplica de un topic
    - Sólo un broker del clúster podrá tener la partición leader de un topic

Brokers

Características

- Esta en comunicación frecuente con el Zookeeper
- Cada broker tendrá un rendimiento diferentes dependiendo de las características HW y su configuración específica
    - Se aconseja que cada broker tenga un nº menor a 2000 particiones

Conectores

Conectores

¿Qué son?

- 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.

Conectores

Características

- 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

Conectores

Características

- 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
- Facilita la evolución de las arquitecturas

Streams

Streams

¿Qué son?

- 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"

Streams

Características

- Permite reutilizar las características de la plataforma Kafka : paralelización, tolerancia a fallos, 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")

Particiones

Particiones

¿Qué es?

- Secuencia de mensajes ordenada e inmutable
- Cada uno de los partes en las que se puede fragmentar/dividir el "Topic Log" de un topic
- Unidad de replicación de un topic

Características

- Cada partición se puede encontrar en uno o varios servidores/nodos/brokers diferentes -> Replicación
    - Define un broker con partición leader y cero o más brokers con particiones followers "replica"
- Facilita la redundancia y escalabilidad
- Facilita de fragmentar las lecturas y escrituras en el log del topic -> se consigue mayor velocidad

Particiones

Características

- Se mantiene el sistema de escritura "append-only" para cada una de las particiones de forma independiente
- Cuando se añade un mensaje en cada una de las particiones se le asigna un offset (ver más en detalle en su parte)
- Las particiones permiten la manipulación "paralela" de los consumidores dentro de uno o varios grupos al mismo tiempo (se explicará en su tutorial específico)

Introducción a ZooKeeper

ZooKepper

¿Qué es?

- Software que proporciona un servicio de coordinación de alto rendimiento para aplicaciones distribuidas
- Manager/gestor del clúster (coordina la topología de los brokers / clúster)
- Almacén Clave-Valor distribuido con los aspectos de control de la plataforma

Características

- Proporciona un servicio centralizado para la gestión de la configuración, registro de cambios, servicio de descubrimiento, etc. (se entera de la incorporación de un broker, cuando muere un broker, cuando se incorpora un topic, estado de salud de las particiones, etc.)
- Requiere ser el primer elemento en arrancar si se requiere una coordinación distribuida

ZooKeeper

Características

- Intercambia metadatos con : brokers, productores y consumidores
  - Direcciones de brokers
  - Offsets de los mensajes consumidos
  - Descubrimiento y control de los brokers del clúster
  - Detección de la carga de trabajo y se produce asignación por cercanía o bien por tener una menor ocupación
  - Etc.
- Proporciona una vista sincronizada de la configuración del clúster

ZooKepper

¿Qué es?

- Secuencia de mensajes ordenada e inmutable
- Cada uno de los partes en las que se puede fragmentar/dividir el "Topic Log" de un topic
- Unidad de replicación de un topic

Características

- Cada partición se puede encontrar en uno o varios servidores/nodos/brokers diferentes -> Replicación
    - Define un broker con partición leader y cero o más brokers con particiones followers "replica"
- Facilita la redundancia y escalabilidad
- Facilita de fragmentar las lecturas y escrituras en el log del topic -> se consigue mayor velocidad

Particiones

Funcionamiento

- Responsable de la selección de la partición leader de un topic entre los brokers
- Responsable de la selección del broker leader donde se almacenará la partición leader
    - Elección del liderazgo del par (Broker, Particion del Topic)

Rol de ZooKeeper

Es **imprescindible**

Servicio **centralizado** que **envía notificaciones** en caso de **cambios**

Orquestador de alto rendimiento para aplicaciones distribuidas.

- Gestiona los clusters.
- Requiere ser instanciado como ‘Entry point’ (Primera instancia).
- Intercambia datos con los brokers, productores y consumidores.
- Responsable de la selección del ‘broker leader’ donde guarda la ‘partición leader’.

Estructuras de datos mantenidas por Kafka en ZooKeeper

Estructuras de datos mantenidas por Kafka en ZooKeeper

Desde la versión 0.9 Kafka proporciona la capacidad de almacenar y distinguir los cambios sucedidos en los respectivos “topic” (offsets) directamente en Kafka en lugar de depender del Zookeeper

Los datos en Zookeeper, aunque son similares a un típico sistema de directorios, pueden tener datos asociados a los nodos. Pueden ser, datos de configuración, detalles de estado, marcas de tiempo, etc...

Estructuras de datos mantenidas por Kafka en ZooKeeper

Desde la versión 0.9 Kafka proporciona la capacidad de almacenar y distinguir los cambios sucedidos en los respectivos “topic” (offsets) directamente en Kafka en lugar de depender del Zookeeper

Los datos en Zookeeper, aunque son similares a un típico sistema de directorios, pueden tener datos asociados a los nodos. Pueden ser, datos de configuración, detalles de estado, marcas de tiempo, etc...

Offsets.

Los “offsets” en Kafka se almacenan como mensajes separados con un formato concreto: xx_consumer_offsets, donde cada consumidor envía un mensaje al topic con intervalos periódicos.

El mensaje contiene, a parte de los metadatos relacionados y el “current offsets”, el grupo de consumidores, el número de partición, el topic asociado y otra información útil.

> * En caso que queramos leer, como consumidores, tales “offsets”, como: xx_consumer_offsets podemos hacerlo como en cualquier otro topic. Antes de hacerlo, debemos hacer que el topic sea visible para los consumidores, ya que este es un topic interno de Kafka y no es visible para los consumidores de forma predeterminada.