ENMILOCALFUNCIONA
THOUGHTS, STORIES AND IDEAS.

# Aprendiendo Apache Kafka (Parte 2) : Conceptos Básicos

Publicado por Víctor Madrid el 30 October 2018

Arquitectura de Soluciones | Kafka

En este **segundo artículo** de la serie **"Aprendiendo Apache Kafka"** se van a detallar qué y cómo se comportan los **componentes o partes** que conforman la plataforma **Apache Kafka** (este artículo en concreto abordara los elementos básicos). Es decir, nos vamos a ir "metiendo en harina" de verdad porque son las cosas con las que vamos a trabajar. Recordar que en el primer artículo únicamente presentábamos la "herramienta" con sus cosas buenas, sus cosas malas y ahora tocan sus elementos / conceptos.


Como son unos pocos os muestro una lista (para que os vayan sonando los nombres :-) ) :

- Zookeper
- Broker
- Clúster
- Mensaje
- Esquema

- Topic / Tema
- Partición
- Offset

> **IMPORTANTE**
>
> Muchos de los conceptos que aquí se explican esta muy relacionados entre sí, además de ser "complejillos" de entender de primeras (a mí me paso la primera vez ;-) ) y tienen mucha teoría asociada.
>
> Por lo tanto, se aconseja leer este artículo unas "cuentas veces" para entender bien el funcionamiento de cada componente, su interrelación y sus particularidades.
>
> Además, si os habéis dado cuenta este artículo esta separado del resto de la serie (y no es un apartado de otro ), de esta forma creo que es una buena forma de tener un punto donde encontrar lo que necesitéis cuando surjan las dudas y poder volver a él tantas veces como sea necesario

En este punto conviene aclarar varios conceptos que serán explicados en otros artículos, pero de momento me interesa que sepáis que son de forma conceptual

- **Productor :** Tipo de cliente de Kafka que se encarga de publicar los mensajes
- **Consumidor :** Tipo de cliente de Kafka que se encarga de consumir los mensajes
- **Grupo de consumidores :** Agrupación de uno o más consumidores que trabajan para leer de un topic en base al cumplimiento de algún objetivo / funcionalidad concreta

Este artículo esta divido en 2 partes:

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

Arrancamos este artículo....

# 1. Componentes

En este punto conviene aclarar que elementos están más relacionados con la infraestructura de la plataforma y cuales están más relacionados con su funcionamiento específico.

## Zookeeper

### Definiciones

- *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*

> **Símil :** La "torre de control" de un aeropuerto respecto a los aviones o bien haciendo honor a su nombre "el cuidador de un Zoo" respecto a los animales

### 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*
- *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*

### Configuración

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

### 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)*

**Diagrama Conceptual** "Zookeeper SIN brokers"


## Broker

### Definiciones

- *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*

> **Símil :** En función de lo que hayamos elegido para el Zookeeper ("torre de control de un aeropuerto" o "cuidador del Zoo"), los brokers serán los aviones o los animales

### 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*
- *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*

### Configuración

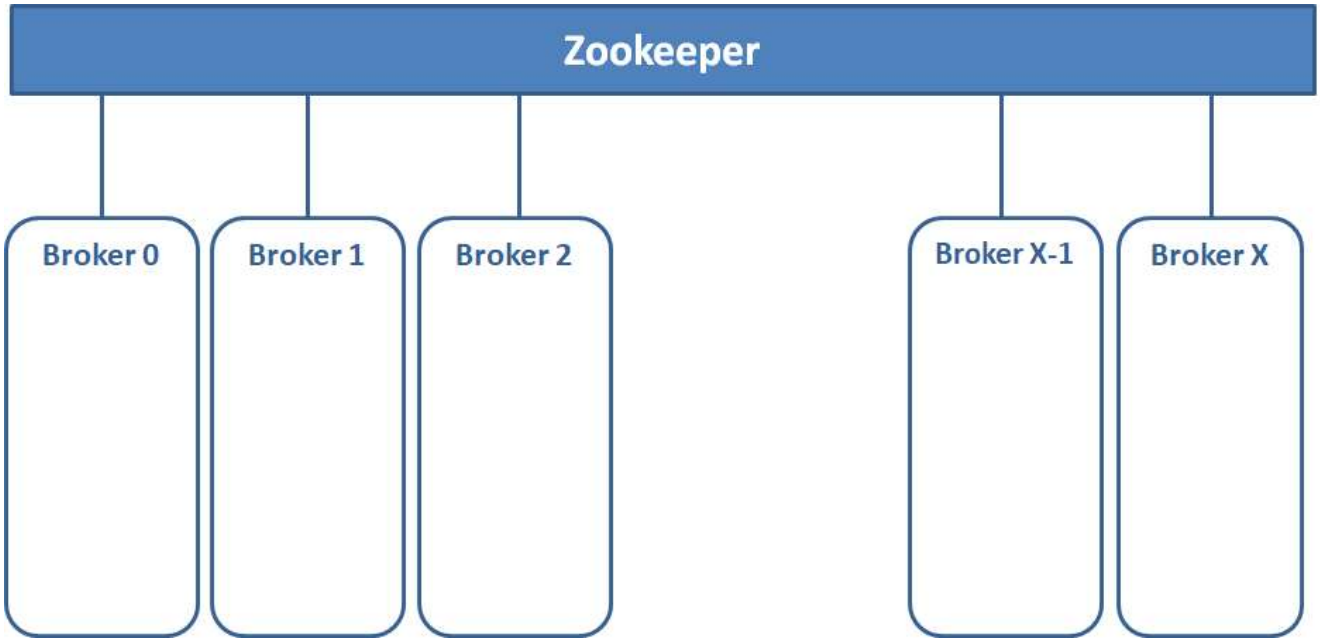
- ***Identificador:***
- *"Número" único para distinguirlo del resto de brokers (este punto es importante)*

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

### Funcionamiento

- ***Escritura :***
- *El broker con la partición leader del topic recibe las peticiones de escritura de los mensajes de los productores*
- *Determina offset a utilizar*
- *Confirma/commit los mensajes en el almacenamiento*
- *Solicita la actualización de las réplicas*
- ***Lectura :***
- *El broker con la partición leader del topic recibe las peticiones de lectura de los mensajes de los consumidores*
- *Determina el offset definido para ese consumidor*
- ***Fallos :***
- *Si el broker con la partición leader del topic se cae realizando alguna operativa, se le asignará otro broker y su partición replica se convertirá en leader debido al ISR (in-sync replica)*

**Diagrama Conceptual** "Zookeeper CON brokers"




- Un único Zookeeper
- X Brokers

# Clúster

### Definiciones

- *Agrupación de 1 o N Brokers*

### Características

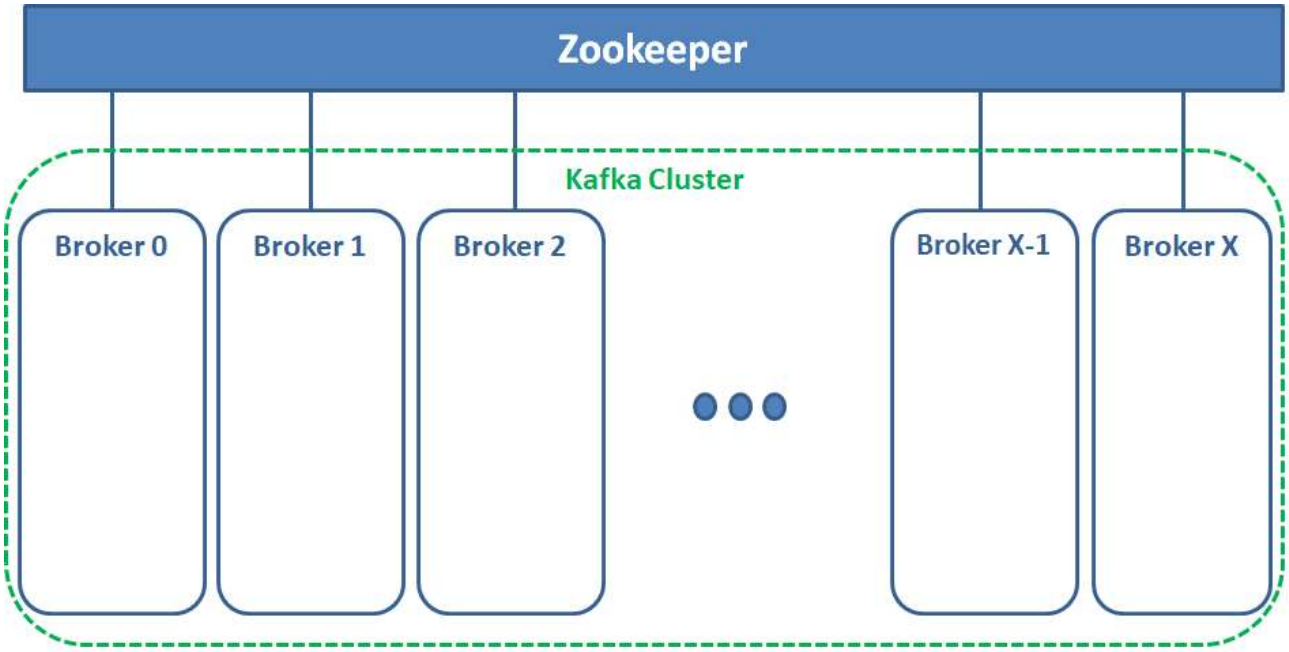
- *Se compone : N Brokers + Apache Zookeper*
- *Kafka se ejecuta como un clúster de uno o más servidores / brokers*
- *El clúster tiene el objetivo de guardar un conjunto de records / registros en categorías llamadas toopics*
- *Una partición es propiedad de un sólo broker en el clúster*
- *Ese broker se considera leader de la partición el resto tendrá réplicas de la partición*
- *Todos los consumidores o productores que operan en esa partición deben conectarse con el broker leader*
- *Se pueden tener múltiples clusters (ayuda a segregar datos, establecer requisitos seguridad, recuperación de desastres, etc.)*
- *Si se necesita mayor tolerancia a fallos se puede configurar un clúster con mayor factor de replicación*
- *Se aconseja utilizar un máximo de 7 brokers por cada ZooKeeper*
- *El tamaño dependerá de la configuración de retención y del tráfico*
- *El uso de red para cada broker debería estar por debajo del 75%*

- *Un clúster se puede replicar usando "Mirror Maker" -> Recuperación de desastres*
- Proporciona scripts específicos para su control : "Kafka-run-class.sh Kafka.tools.MirrorMaker", "kafka-mirror-maker", etc.

### Funcionamiento

- ***Establecer un controlador del grupo:***
- *De todos los brokers que componen el clúster se selecciona uno como controlador del grupo*
- *Responsable de las operaciones administrativas, incluyendo particiones y monitorización de fallos*
- *Si falla entonces otro broker elegido por los broker activos asumirá el liderazgo*

**Diagrama Conceptual** "Zookeeper con un clúster de brokers "




- Un único Zookeeper
- X Brokers formando un clúster

## Mensaje

### Definiciones

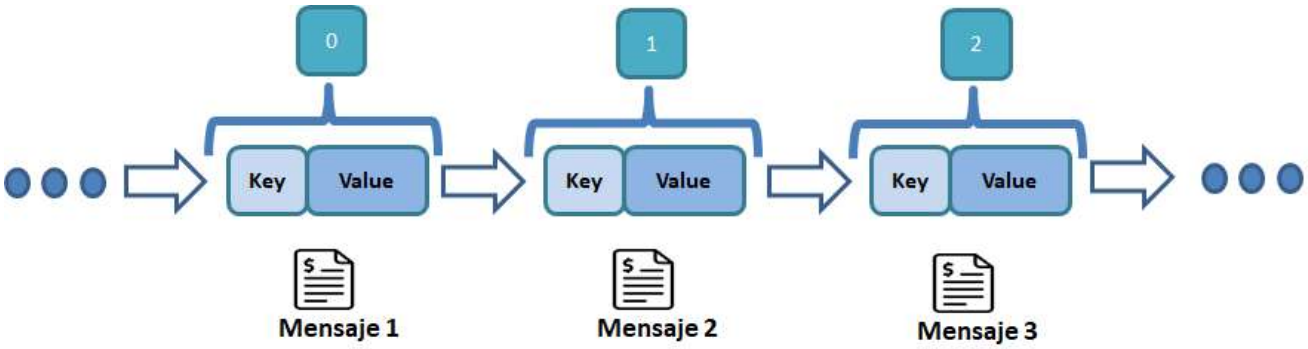
- *Unidad de datos con la que trabaja Kafka* (sería la unidad de comunicaciones entre aplicaciones)
- *Array de bytes*
- *Par (Clave[opcional],Valor) y un timestamp*

> **Símil :** Un mensaje sería similar a un registro de base de datos

### Características

- *También se usa el término "registro" / "record" / "message"*
- *No tiene una estructura, formato o significado específico para Kafka*
- *Esto puede cambiar con el uso de esquemas*
- *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*
- *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*

**Diagrama Conceptual** "Stream de Mensajes"




# Esquema

## Definiciones

- *Estructura opcional que se aplica sobre los mensajes garantizando una distribución concreta de la información*
- *Formato de los datos con los que se trabajará*

> **Símil :** Si pensamos en un tabla de base de datos, el esquema sería la forma en la que se construye teniendo en cuenta la posición que ocupa cada columna así como el tipo de información que almacenará (teniendo en cuenta tipología, restricciones, etc.)

## Características

- *Por ejemplo : JSON , XML y Apache Avro (framework de serialización para Hadoop)*
- *Facilita la compresión*
- *Facilita la independencia entre la lectura y la escritura de los mensajes*
- *Facilita la realización de mantenimientos sobre las aplicaciones*
- No es obligatorio utilizarlo -> Depende de los requerimientos

# Topic / Tema

## Definiciones

- *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*

> **Símil :** Guardar un fichero en un directorio concreto o bien un registro en una tabla de base de datos.

> Un topic sería la tabla y los mensajes serían sus registros o bien sería el directorio y los mensajes serían sus ficheros

## Características

- *Pueden existir 1 o más topics -> No hay limitación*
- *Kafka almacena los topics en logs*
- *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*
- *Tipo de escritura : los datos en el "commit log" es append-only*
- *Añadir por el final (es decir, el orden en el que son enviados)*
- *Tipo de lectura : ordenada de principio a fin*
- *Kafka dispone de una herramienta específica para su gestión*

## Estructura

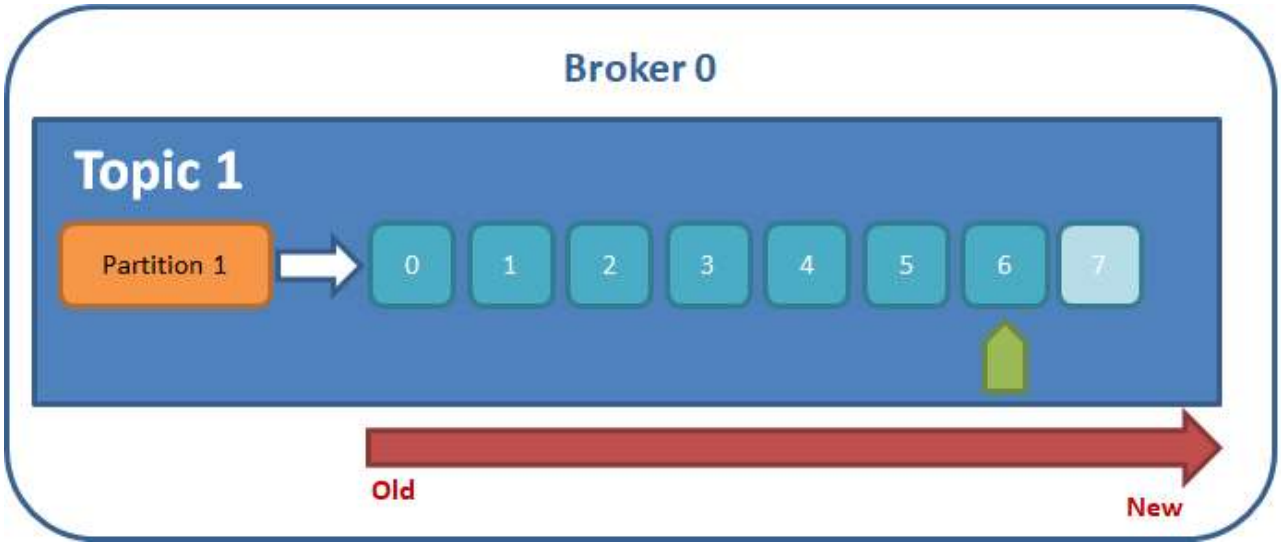
Cada topic tiene:

- ***Topic Log :*** *Estructura de almacenamiento en el disco de mensajes*
- *Se trata de un fichero de log*
- *Los mensajes son estructurados y organizados*
- ***Partición :*** *Partes en las que se puede dividir el "Topic Log" (ver concepto de Partición porque es más complejo)*
- *1 a N particiones (por defecto es 1)*

### Configuración

- ***Identificador :***
- *"nombre" con el que se identificará de forma única el topic dentro de Kafka*
- ***Nº de particiones:***
- *Cantidad de fragmentos en los que se dividirá el "topic log" de un topic*
- *Depende de la cantidad de brokers que componen la topología*
- *Ejemplo : Si se dispone de 5 brokers y un valor de partición de 3 significa que : 3 brokers tendrán una parte de la partición de un topic log y 2 brokers no tendrán partición de topic log*
- ***Factor de replicación:***
- *Nº de replicaciones / copias que de las particiones que mantendrá en el clúster*
- *La cantidad de réplicas necesarias tiene que ser menor o igual al número de brokers*
- *Consejo : usar un factor de replicación de al menos 2 (normalmente es un valor de 3) con 3 se sobrevive a un fallo de AZ (Availability Zone)*
- *Permite obtener la tolerancia a fallos*
- ***Criterios de retención:***
- *Reglas que facilitan la invalidez de los mensajes de un topic*
- *El incumplimiento de alguno de estos criterios hace los mensajes "caduquen" y sean eliminados*
- *Asegura disponer de la cantidad mínima de mensajes con los que trabajar*
- *Tipos*
- *Retención durante cierto tiempo (por ejemplo 7 días)*
- *Retención hasta alcanzar cierto tamaño*
- *Compactación : sólo se conserva el último mensaje producido con una clave específica*

**Diagrama Conceptual** "Un único tema/topic con 1 partición en un broker"




- Un único nodo / broker
- Nombre de la partición : "Topic 1"
- Número de particiones : 1 (Por defecto)
- La partición dispone de un stream de mensajes

# Partition / Partición

## Definiciones

- *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

> **Símil :** Sería como tener el contenido de registros de una tabla de base de datos "troceados" y repartidos en tablas más pequeñas que se encuentran en servidores diferentes -> distribuido

## 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"*
- *El broker leader controla todas las peticiones de lectura y escritura sobre una partición del topic*
- *Conviene monitorizar al broker leader*
- *Los brokers followers replican a los leaders y toman el control si el leader establecido "muere"*
- *Cada broker réplica puede comunicarse con el Zookeeper*
- *Al grupo de particiones réplicas sincronizadas se las llama ISR (In-Sync Replicas)*
- *Facilita la redundancia y escalabilidad*
- *Facilita de fragmentar las lecturas y escrituras en el log del topic -> se consigue Mayor velocidad*

- *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)*
- *Consejo : el tamaño de cada partición debería sea menor de 25GB*

## Configuración

- Existen algunos parámetros para establecer los tiempos de actualización de las réplicas, el mínimo número de réplicas, etc.

Nota : los parámetros de configuración se detallarán en los ejercicios prácticos en el resto de artículos

## Funcionamiento

#### Partition leader:

- *Cuando se replica una partición en diferentes brokers se seleccionará una de ellas como leader*
- *Por lo tanto, existirá una partición leader y el resto serán particiones followers que se comportarán como réplicas de la leader*
- *La escritura de un mensaje se hará en base al offset de la partición leader asignada*
- *Si una partición leader falla una partición ISR será elegida como nueva leader*
- *El consumidor sólo puede leer hasta "High Watermark" -> Los consumidores no pueden leer datos no replicados*
- *La lectura de un consumidor de un mensaje se hace en el orden en el que son guardados en el log*
- *Cada broker gestiona su cuota de datos y peticiones compartiendo el liderazgo de la partición*

#### Ordenación:

Para el caso : 1 Partición

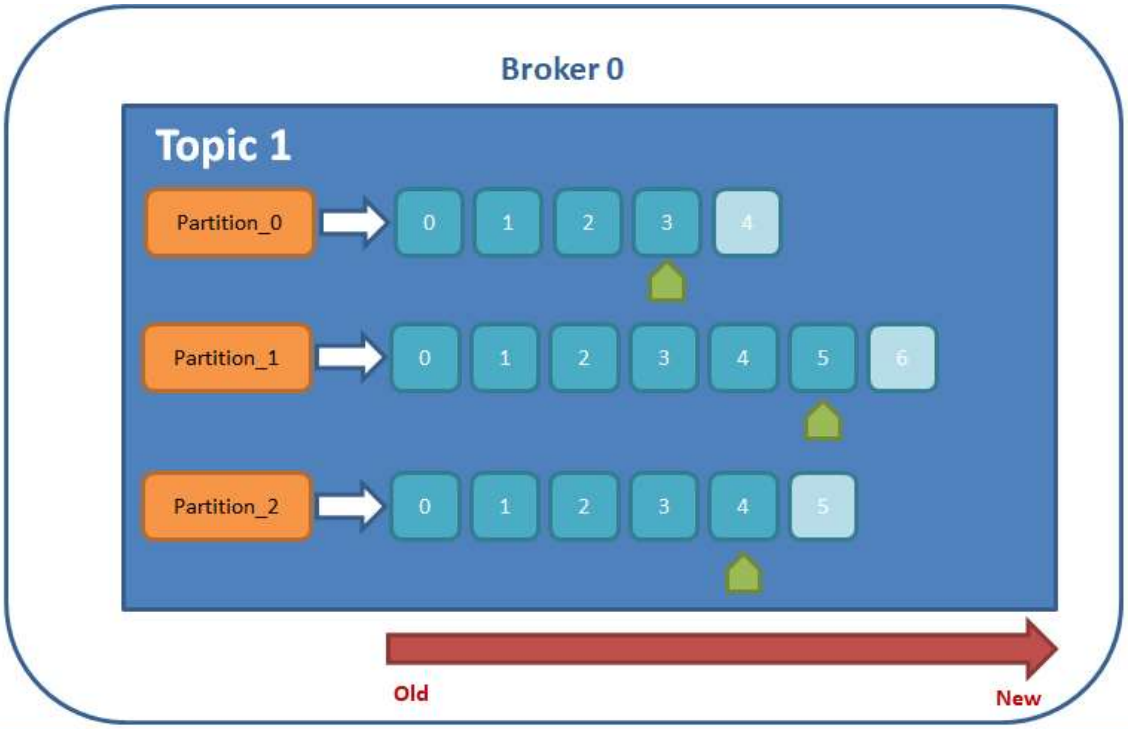
- *Mantiene el orden de los registros*
- *Es la opción más básica*

Para el caso : N Particiones

- *Se garantiza el orden a nivel de sharding / particiones -> pero no se garantiza el ordenamiento temporal a nivel de todo el topic*
- *Agrupar mensajes en una partición en base a la clave*
- *Se puede configurar una instancia de consumo por partición dentro de un grupo de consumidor*

- *En Kafka la replicación sólo se realiza a nivel de partición*

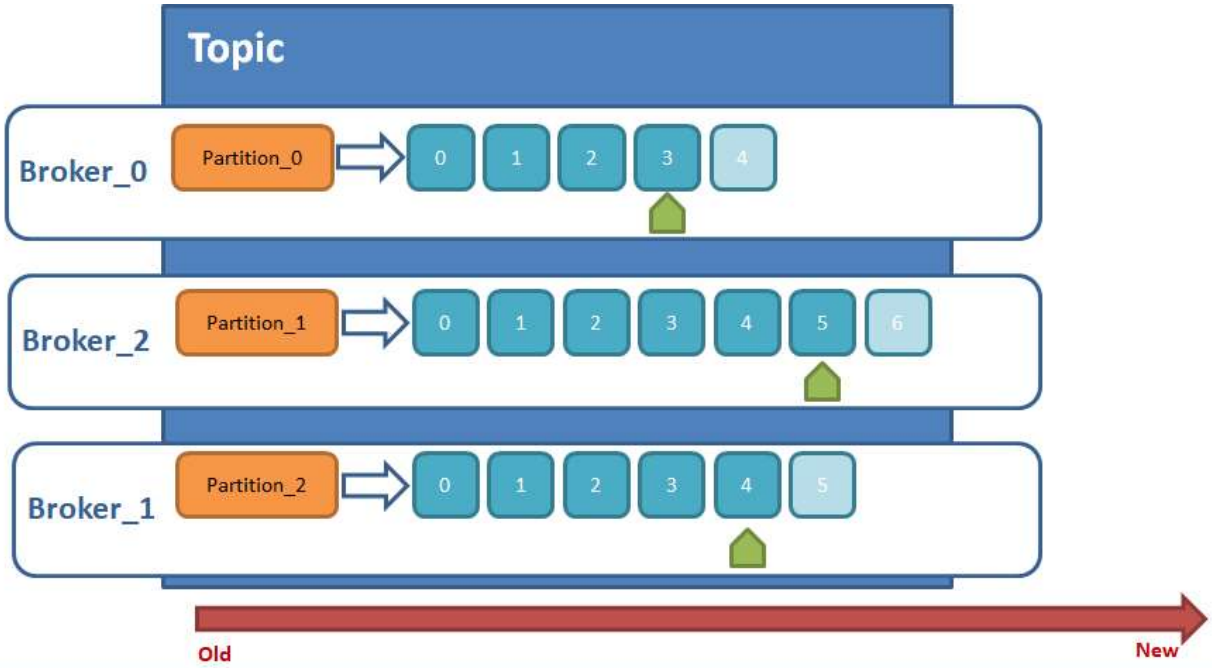
**Diagrama Conceptual** "Un único tema/topic con 3 particiones en un broker"




- *Un topic se puede dividir en múltiples particiones -> balancear la carga*
- *Cada partición es ordenada -> secuencias inmutables de mensajes que se añaden*
- *A cada partición se le asigna un id secuencial llamado offset*

- Un único nodo / broker
- Nombre de la partición : "Topic 1"
- Número de particiones : 3

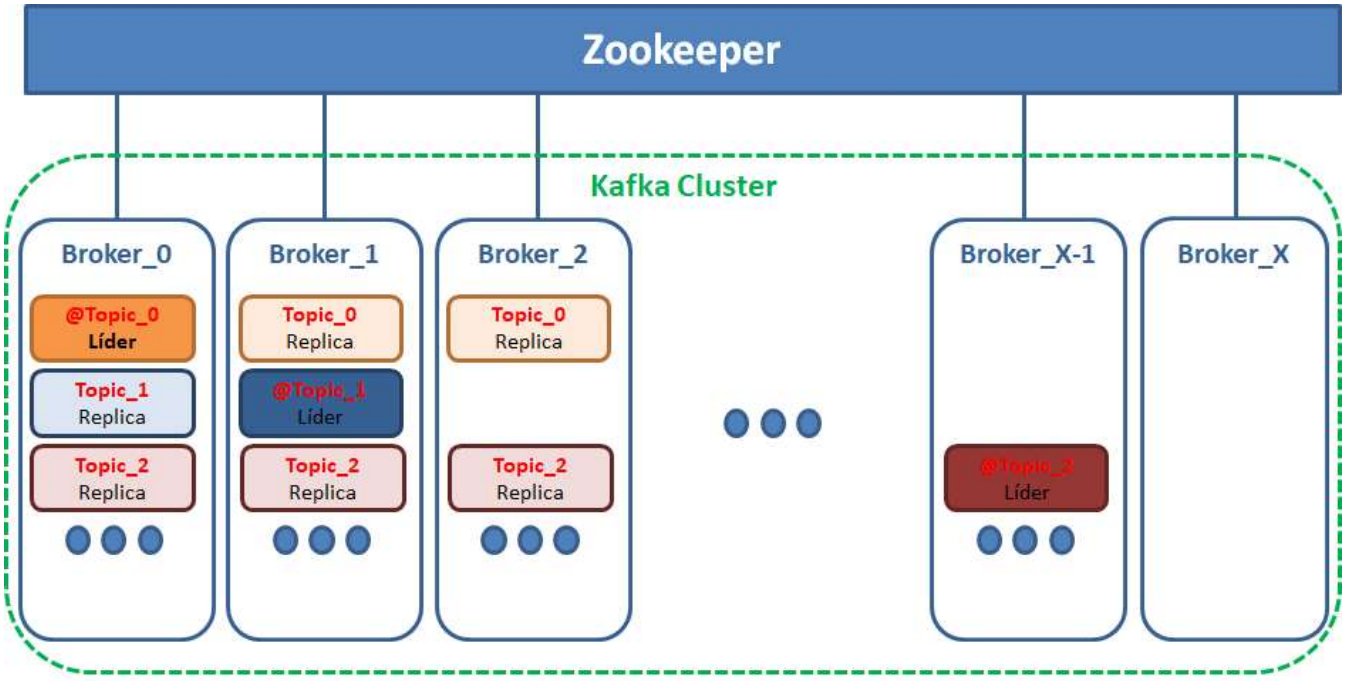
**Diagrama Conceptual** "Un único tema/topic con 3 particiones en 3 brokers SIN replicación"




- *Un topic se divide en particiones*
- *Cada partición se encuentra en un bróker actuando como leader -> Distribución*
- *Cada partición leader maneja todas las peticiones de lectura o escritura sobre ella*
- *Si existe factor de replicación, la partición leader tendrá "copias" en el resto de brokers -> particiones followers o replicas*
- *Si la partición leader falla una de las particiones followers pasará a ser leader*

- 3 nodos / brokers
- NO hay replicación
- Nombre de la partición : "Topic 1"
- Número de particiones : 3

**Diagrama Conceptual** "3 temas/topics con diferentes factores de replicación en X brokers"

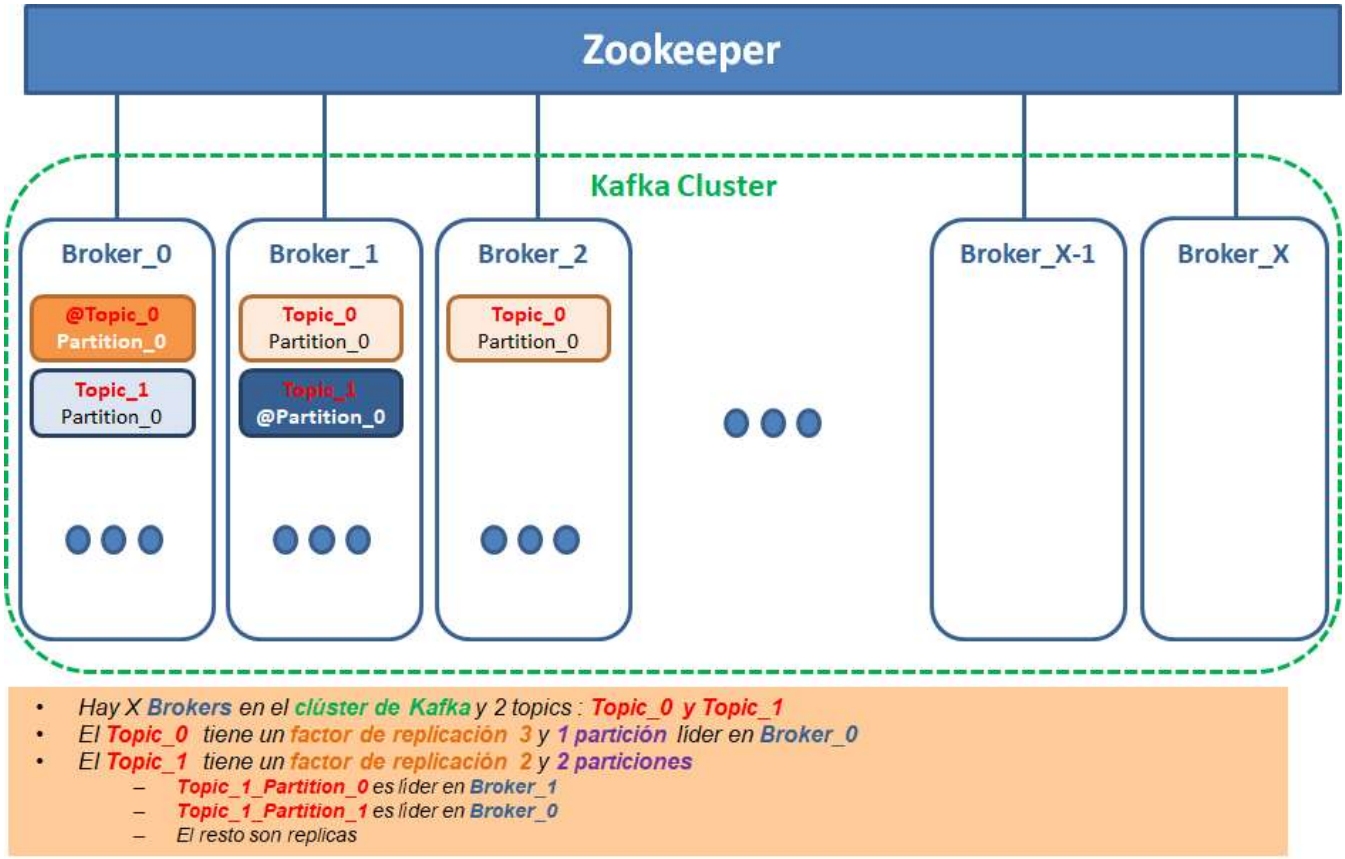



- *Se dispone de X Brokers en el clúster de Kafka y 3 topics : Topic_0, Topic_1 y Topic_2*
- *El Topic_0 tiene un factor de replicación 3 (con líder en Broker_0)*
- *El Topic_1 tiene un factor de replicación 2 (con líder en Broker_1)*
- *El Topic_2 tiene un factor de replicación 4 (con líder en Broker_X-1)*

- X nodos / brokers dentro de un clúster

- Número de topics: 3
- Cada topic sólo tiene un partición
- Cada partición tiene diferentes factores de replicación
- Existen particiones leader y particiones follower o réplica
- Cada partición tiene un partición leader en broker diferentes aunque podían haber coincidido en el mismo

**Diagrama Conceptual** "2 temas/topics con diferentes factores de replicación en X brokers"




- X nodos / brokers dentro de un clúster
- Número de particiones : 2
- Cada partición tiene diferentes factores de replicación
- Existen particiones leader y particiones follower o réplica repartidas por las particiones
- Cada partición tiene un partición leader en broker leader diferentes aunque podían haber coincidido en el mismo

# Offset

## Definiciones

- *Identificador único de cada mensaje en base a su ubicación dentro de una partición*
- *Bit de metadatos que se corresponde con un valor entero incremental / secuencia*
- *La posición del consumidor en el topic log o en una de sus particiones*

> **Símil :** Puntero / índice dentro de una lista

### Características

- *También se denomina "desplazamiento" o "compensación"*
- *Kafka guarda su información en un topic llamado "_consumer_offset"*
- *Tipología:*
- ***"Log end offset" (offset de fin de registro) :** offset del último registro escrito en la partición y donde los productores escribirán a continuación el siguiente mensaje*
- ***"High Watermark" :** offset del último registro que se replico con éxito en todos las particiones followers réplicas*

## 2. Conclusiones

Menudo "tostón" que acabo de soltar...jejeje. Es cierto, que en algunos momentos he sido demasiado teórico y he mezclado muchas cosas, pero ya me lo agradeceréis cuando vayamos avanzando en la complejidad de los ejercicios y sobre todo cuando vayáis asimilando los conceptos y dominándolos al ver como funcionan en "real" todas las cosillas que aquí se detallan.

Recordar que la idea de este artículo era separar la parte de conceptos de los ejercicios prácticos para centralizar la teoría.

Ya tenemos lo básicos ... ya nos queda menos para ser expertos :-)

¡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 ;-)

**E** **Esperanza**
hace un año

Buenos Días Victor, muchas gracias por el articulo, muy ilustrativo. Te voy a hacer una consulta, en caso de que no se haga el commit por que haya tenido error, e.p. la Base de Datos no disponible. ¿Cuando me lee el listener otra vez este mensaje? Gracias



**J** **Juan**
hace 3 años

Alguien me puede dar infromacion de cual es el tamaño maximo de dato que se puede enviar y almacenar en u offset?



**M** **Mauri**
hace 4 años

Hola Victor, buenos días, antes que nada muy buenos los artículos, muy ilustrativos! muchas graciaspor el aporte!. Ahora una consulta, en la analogía que mencionas sobre **Broker** *"Símil : En función de lo que hayamos elegido para el Zookeeper ("torre de control de un aeropuerto" o "cuidador del Zoo"), los brokers serán los aviones o los animales"*; el broker en realidad no seria "la pista"? (así le decimos en Argentina, por donde aterrizan y despegan los aviones) y los aviones los mensajes? Muchas gracias.



**JP** **Jean Pierre** → Mauri
hace un año

Las pistas serian las particiones de un tópico, por ello la torre de control gestiona a que pista debe ir cada avión, lo mismo que un broker cuando gestiona a que partición debe ir un mensaje.



**Markens**
hace 4 años

En Diagrama Conceptual "3 temas/topics con diferentes factores de replicación en X brokers" has puesto que hay 3 particione
que es una errata. El Offset no he logrado
posición del consumidor, pero luego dice
Entiendo que no tienen porque coincidir, no?



0 0 Responder • Comparte ›

**Víctor Madrid** → Markens
hace 4 años

Tienes toda la razón es una errata, son 3 topics cada uno de ellos con 1 partición … he terminado mezclando las dos cosas en una...jejeje (lo corregiré en breve)

Lo del offset es un poco mas complejo pero digamos que existe un offset de lectura y otro de escritura que dependiendo de la configuración (individual o por grupo de consumo) puede variar y pasa lo que tu dices, que no tiene porque coincidir (por ejemplo la escritura puede ir mas rápida que la lectura), en próximos artículos lo detallare ya que con este pretendía ayudar a entender conceptualmente las partes básicas

Muchas gracias :-)

0 0 Responder • Comparte ›

**Jesus J. Puente**
hace 4 años

Gran árticulo!. Pero tengo una una duda: ¿Kafka es el que guarda los mensajes recibios o eso lo hace Zookeeper?.
Gracias por compartir tus conocimientos :-)

0 0 Responder • Comparte ›

**Víctor Madrid** → Jesus J. Puente
hace 4 años

Buenas Jesus,
Zookeeper se encarga de centralizar la coordinación de los brokers / aplicaciones (gestiona la configuración, registra los cambios que se producen, proporciona un servicio de descubrimiento de brokers, intercambia los offsets de los mensajes , etc.) pero NO se encarga de la persistencia de los mensajes ya que esto lo hace Kafka

0 0 Responder • Comparte ›

**Jesus J. Puente** → Víctor Madrid
hace 4 años

Muchas gracias por tu rapida respuesta. Estoy haciendo un proyecto con Spring Boot y queria utilizar Kafka y tus articulos me esta siendo de gran ayuda. Aprovecho para consultarte otra duda. He hecho pruebas y veo que si hay varios listeners (en diferentes puertos) sobre el mismo topic en la misma maquina, solo uno recibe el mensaje. ¿En el caso de que estuvieran en diferentes maquinas, con docker, por ejemplo, eso

tambien seria asi?. Muchas gracias por adelantado.

0 0 Responder • Comparte ›

**Víctor Madrid** → Jesus J. Puente

hace 4 años

Buenas
¿te refieres a por listeners a los consumidores? ¿has verificado si en tu ejemplo estas usando un "grupo de consumo"?

0 0 Responder • Comparte ›

**Jesus J. Puente** → Víctor Madrid

hace 4 años

Buenas,

No, no he creado ningun grupo. Simplemente he lanzado el mismo programa que tiene un simple listener dos veces, escuchando en diferentes puertos (con SpringBoot) y veo que solo uno de los programas recibe el evento. No se si esto sera un tema de SpringBoot o es el comportamiento normal.

0 0 Responder • Comparte ›

**Víctor Madrid** → Jesus J. Puente

hace 4 años edited

Tendría que ver la configuración de propiedades de consumidor y las del topic que estas utilizando. Para empezar a guiarte en el problema creo que puede tener que ver con estas causas :
* NO de deberías de tener la siguiente propiedad que establece el uso de grupos de consumo :

```
props.put(ConsumerConfig.GROUP_ID_CONFIG,
"ExampleConsumerGroup_1");
```

* La configuración del número particiones del topic : cuando existen particiones estas se reparten en los brokers y dependerá de donde estén escuchando los consumidores

* Los consumidores deberían de tener la propiedad bootstrap-server apuntando a los brokers (separados por ",") y ver si todos apuntan a todos o cada consumidor apunta a 1

```
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
```

ver más

0 0 Responder • Comparte ›