APACHE KAFKA. Introducción teórica.

"Apache Kafka is an open-source message broker project developed by the Apache Software Foundation written in Scala. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. The design is heavily influenced by transaction logs".

I. INTRODUCCIÓN.

Podemos describir el software ***Apache Kafka*** como un sistema de mensajería de publicación/suscripción distribuido rediseñado como un registro de confirmación distribuido. Está diseñado para ser rápido, escalable, duradero y con tolerancia a errores, proporcionándonos una plataforma unificada, de alto rendimiento y baja latencia para la gestión de entrada de datos en tiempo real. Kafka nació como proyecto OpenSource en LinkedIn Corporation para solucionar las limitaciones de los sistemas de mensajería tradicional basados en JMS.

Apache Kafka fue diseñado con las siguientes características:

- ***Real Time***. Los mensajes producidos por los productores están inmediatamente disponibles para los consumidores. Un solo Broker puede trabajar con cientos de megabytes de lecturas y escrituras por segundo desde miles de clientes.
- ***Escalable***. Está diseñado para permitir que un único clúster funcione como el eje central de datos, el cual puede ser ampliado sin tiempo de inactividad. Esto es gracias al diseño de la arquitectura basada en procesos sin estado.
- ***Mensajería persistente a estructuras de disco O1***. Suministra un rendimiento persistente en el tiempo, incluso cuando se produce el almacenamiento de varios TB de mensajes. Con Kafka los mensajes son persistidos en disco y replicados para evitar la pérdida.
- ***Alto rendimiento***. Kafka tiene la capacidad de tolerar cientos de miles de mensajes por segundo, incluso mediante la utilización de un hardware sencillo.
- ***Soporte para la participación de mensajes***. Esto se produce mediante los servidores de Kafka y el consumo distribuido en un clúster de máquinas consumidoras, manteniendo la ordenación por partición.
- ***Soporte para la carga de datos en paralelo en Handoop***. Kafka procura agrupar el procesamiento online y offline, suministrando un mecanismo para la carga paralela en Hadoop, así como la capacidad de partición en tiempo real del consumo en un clúster.

En la siguiente imagen se pueden observar los principales componentes de su arquitectura.

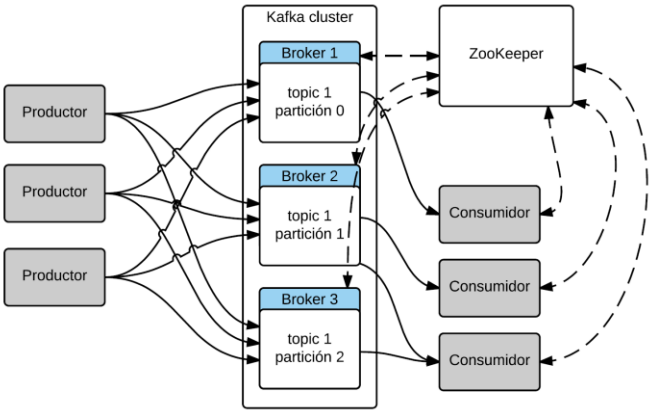


Ilustración 1 *Arquitectura Apache Kafka.*

Para poder entender la arquitectura de Kafka vamos a describir algunos conceptos interesantes que debemos conocer. La unidad mínima de información que utilizaremos, será el mensaje, que definiremos como un conjunto de bytes. Un streaming de mensajes de un tipo o categoría determinada será conocido como Topic. Kafka mantiene los Topic en un log particionado, cuya representación es la siguiente:

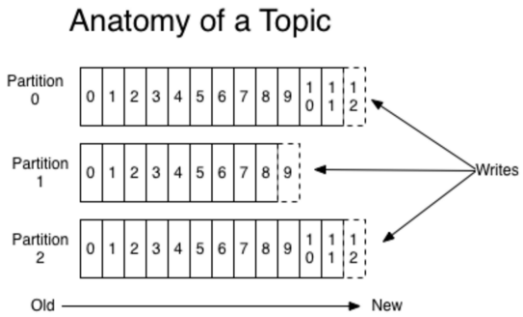


Ilustración 2 Estructura básica de una cola TOPIC.

Cada partición es una secuencia de mensajes inalterable, donde los mensajes son añadidos a una de las particiones según van llegando, y se les va asignando un número secuencial, llamado offset, que identifica de forma única a cada mensaje dentro de su partición. Todos los mensajes son mantenidos en las particiones para su consumo durante un período de tiempo determinado. La gestión de la forma de funcionamiento la denominaremos: ***Ciclo de vida de las colas TOPIC***. Apache Kafka nos proporciona diferentes parámetros de configuración que afectan drásticamente al modo de funcionamiento de las TOPIC. Una de las primeras configuraciones que podemos

modificar es la política de retención de los mensajes en las colas. Las diferentes posibilidades son:

- Forever.
- For a period of time.
- Until the log gets a certain size
- After they are deleted.
- As the lastest keyed versión…

Conocidas las posibilidades, hay muchas maneras de realizar las diferentes configuraciones. En un primer nivel la opción "cleanup.policy" nos permitirá configurar "delete" o "compact". El valor por defecto es el de "delete". Para modificar las políticas referentes a delete, podemos utilizar los siguientes parámetros:

- ***delete.retention.ms***. Define el tiempo para borrar el último mensaje.
- ***retention.bytes***. Por defecto es 0, no se controla el tamaño máximo antes de eliminar los mensajes.
- ***retention.ms***. Por defecto es 7 días. Es el tiempo máximo que un mensaje puede permanecer en la cola.

Las colas Topic se implementan dentro del ***Broker*** que es un proceso servidor que está corriendo en los servidores del clúster de Kafka. Cada servidor puede tener uno o varios Brokers ejecutándose.

Al contrario que otros sistemas de mensajería, los Brokers de Apache Kafka ***no tienen estado***s, es decir, no saben si un mensaje ha sido leído o no. Toda la lógica de los mensajes la debe llevar el consumidor. Por este motivo es complicado saber cuándo eliminar un mensaje del bróker, ya que desconocemos el estado del consumidor. Kafka resuelve esta problemática eliminando los mensajes que hayan estado en el bróker una determinada cantidad de tiempo. Esta forma de funcionar permite también que otros consumidores lean mensajes hasta el periodo de retención configurado.

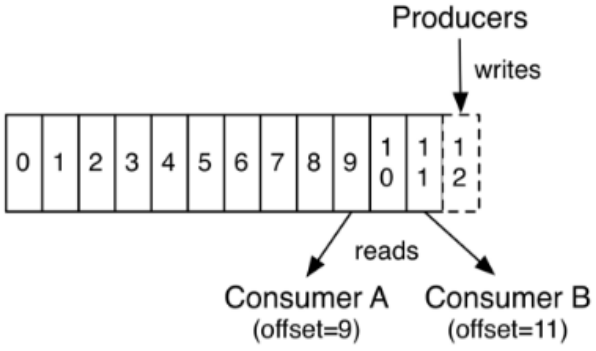


Ilustración 2 Representación gráfica concepto offset.

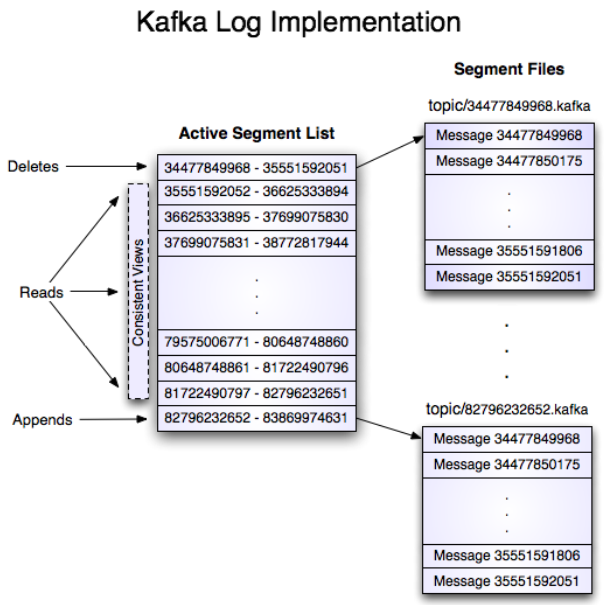


Ilustración 3 *Kafka log implementation.*

Los ***productores*** son los responsables de publicar los mensajes en uno o varios Topics. Los mensajes son enviados cada vez a una de las particiones generalmente siguiendo el algoritmo round-robin o el criterio que hayamos definido. Un tema interesante referente a los productores, es que tienen la posibilidad de escoger el método de serialización más conveniente para enviar el contenido del mensaje.

Definiremos a un ***consumidor*** como a un proceso que puede subscribirse a uno o a más Topics y leer los mensajes publicados desde los Brokers. Kafka para manejarlos proporciona un único nivel de abstracción donde recoge el modelo de mensajería de queuing(colas) y publish-subscribe(editor-subscriptor). Este nivel de abstracción se denomina ***consumer group*** o grupo consumidor.

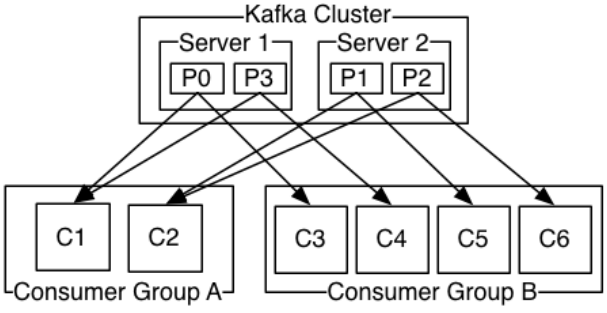


Ilustración 4 Consumers group.

Referente a su funcionamiento podemos remarcar los siguientes puntos:

- Los consumidores se inscribirán a un grupo consumidor. El grupo consumidor podrá tener de *'1'* a *'n'* consumidores.

- Cada mensaje publicado en un Topic será entregado solamente a uno de los consumidores que pertenecen a un mismo grupo de consumidor. De esta forma se consigue funcionar como un sistema de colas tradicional.

- En el caso de que existan varios grupos de consumidores subscritos a un mismo Topic, cada mensaje publicado en un Topic es entregado a todos los grupos consumidores, y dentro de cada grupo consumidor a un solo consumidor. Esta es una de las formas de conseguir el modelo productor-consumidor.

Referente a conseguir mantener el orden de los mensajes en a la hora de consumirlos, Kafka utiliza una solución innovadora basada en SLAs y particiones que permite tanto el escalado como mantener el orden.

Una de las principales características que señalábamos de Kafka es que es un sistema escalable gracias a que podemos dividir una Topic en diferentes particiones. Sabiendo esto, los productores de mensajes pueden publicar los mensajes en base a una clave y por tanto son enviados a la misma partición. Finalmente, a cada partición se subscribe un grupo consumidor, momento en el que solo un consumidor del grupo comienza a procesar los mensajes de esa partición, mientras se procesa también el resto de mensajes de las otras particiones consumidos por otros consumidores, logrando garantizar el orden de entrega de los mensajes, mientras se sigue balanceando y escalando el sistema.

Solamente en el caso de no poder dividir el Topic en varias particiones, o de no poder enviar los mensajes a la misma partición usando una clave, volveríamos a tener la limitación de un único consumidor y dejaríamos de escalar el sistema.

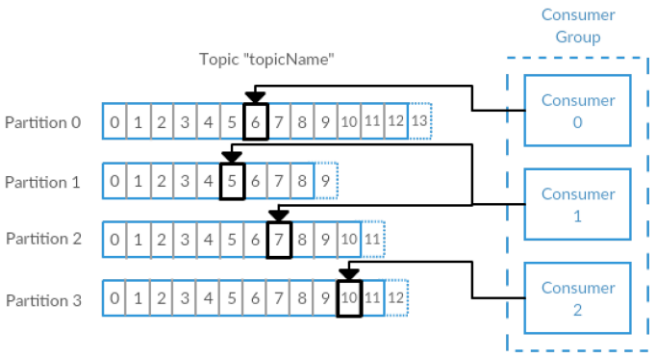


Ilustración 5 Un consumer group leyendo de una Topic.

La lectura de mensajes llevada a cabo por los consumidores es extremadamente rápida gracias a la implementación de ***zero-copy*** llevada a cabo por *java.nio.channels.FileChannel#transferTo*. Este método usa la llamada al sistema "sendfile" que proporciona de una manera muy eficiente trasnferir datos de un fichero a otro fichero (incluidos sockets). No me voy a extender más en este concepto, pero podemos encontrar una muy buena explicación en:

http://www.ibm.com/developerworks/library/j-zerocopy/

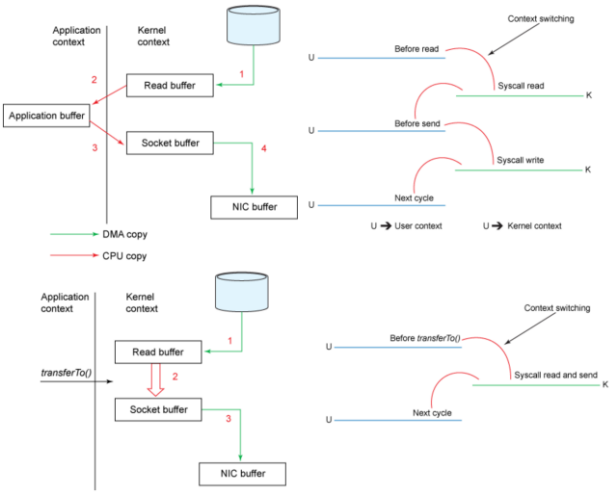


Ilustración 6 Forma tradicional frente a Zero-Copy.

Terminaremos esta breve introducción enumerando algunos de los casos de uso posible para Apache Kafka:

- ***Log aggregation***. Recolección de logs de los diferentes servidores y centralización de los mismos en un punto central. Para realizar esta tarea Kafka presenta una baja latencia con numerosos conectores.

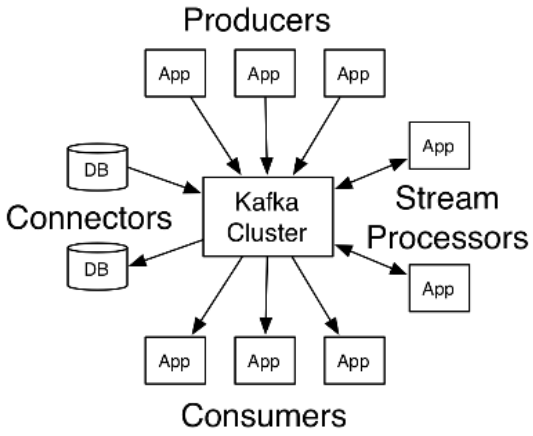
- ***Stream processing***. Permite realizar análisis online, por lo que suele usarse para enriquecimiento de datos y análisis de los mismos.

- ***Commit logs***. Puede utilizarse para realizar un "commit" externo de los datos en sistemas distribuidos o incluso para replicación de los clusters.

II. ARQUITECTURA.

Como hemos descrito en el apartado anterior Apache Kafka presenta una arquitectura sencilla basada principalmente en cuatro componentes:

- ***Productores***. Que generaran los flujos de mensajes.
- ***Consumidores.*** Consumirán los mensajes.
- ***Broker.*** Servidor de Kafka que recibirá y proporcionará los mensajes mediante la agrupación de los diferentes flujos en Topics.
- ***Zookeeper.*** Proporciona alta disponibilidad y tolerancia a fallos para el servicio de coordinación del entorno distribuido.



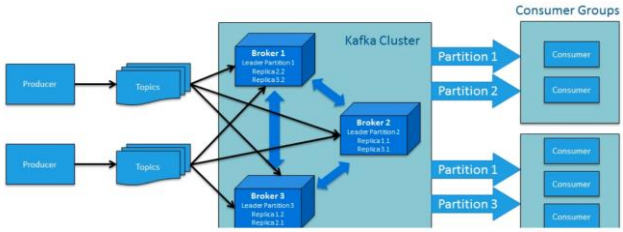


Ilustración 7 Arquitectura genérica Apache Kafka.

La forma que tiene de almacenar los mensajes en los diferentes Brokers también es muy sencilla. Cada partición de un Topic corresponde a un registro lógico. Físicamente, el registro se implementa como un conjunto de archivos compuestos por segmentos de igual tamaño. Cada vez que un productor publica un mensaje en una partición, el broker simplemente añade el mensaje al último segmento. El segmento se vuelca de forma controlada a disco después de n mensajes. Después de esto los mensajes pasan a estar accesibles por los consumidores.

Para tolerar un número ***N*** de fallos, necesitaremos ***"2N+1"*** replicas. Siguiendo esta máxima un esquema típico sería el siguiente:

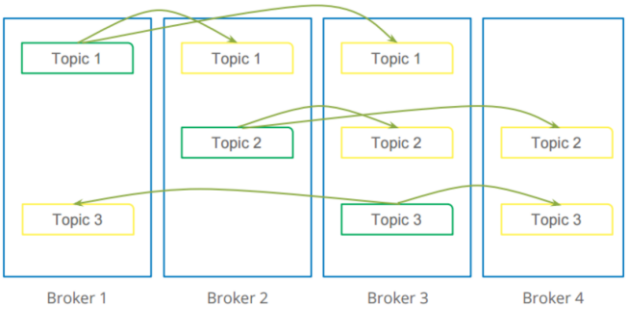


Ilustración 8 Primary-backup replication.

Recordemos que ZooKeeper es un sistema de archivos distribuido, jerárquico que facilita el acoplamiento débil entre los clientes y proporciona una vista de consistencia eventual mediante sus znodes, que son como los archivos y directorios en un sistema de archivos tradicional. Proporciona operaciones básicas tales como la creación, supresión, y la comprobación de la existencia de znodes. Proporciona un modelo orientado a eventos en los que los clientes pueden observar los cambios a znodes específicos, por ejemplo, si un nodo nuevo se añade a un znode existente. ZooKeeper logra una alta disponibilidad mediante la ejecución de varios servidores ZooKeeper, con cada servidor conteniendo una copia en memoria del sistema de archivos distribuido para dar servicio a los clientes las solicitudes de lectura.

ELABORANDO!!!!!!!

III. BIBLIOGRAFÍA.

- Apache Kafka, el sistema de mensajería distribuido de Linkedin. https://loquemeinteresadelared.wordpress.com/2014/07/15/apache-kafka/
- Running a Multi-Broker Apache Kafka 0.8 Cluster on a Single Node. http://www.michael-noll.com/blog/2013/03/13/running-a-multi-broker-apache-kafka-cluster-on-a-single-node/
- Reilable RT processing @Spotify https://www.jfokus.se/jfokus14/preso/Reliable-real-time-processing-with-Kafka-and-Storm.pdf
- Building a Replicated Logging System with Apache Kafka. Gouzhang Wang. http://www.vldb.org/pvldb/vol8/p1654-wang.pdf
- Kafka: a Distributed Messaging System for Log Processing. Jay Kreps, Neha Narkhede, Jun Rao. http://research.microsoft.com/en-us/um/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf
- Arquitectura Big Data. Mario Pérez Esteso. http://www.python-madrid.es/media/slides/Presentaci%C3%B3n_Meetup_Python.pdf
- Apache Kafka: Next Generation Distributed Messaging System. https://www.infoq.com/articles/apache-kafka
- Análisis de arquitecturas de procesado de Streaming big data. Mario Pérez Esteso. http://www.dit.upm.es/~posgrado/doc/TFM/TFMs2014-2015/TFM_Mario_Perez_Esteso_2015.pdf
- Estudio y mejora del rendimiento de Backend. Rafael Alejandro Mollá Sirvent.
- Zookeeper & Kafka install: Asingle Node and Single Broker cluster – 2016. http://www.bogotobogo.com/Hadoop/BigData_hadoop_Zookeeper_Kafka_single_node_single_broker_cluster.php
- ZOOKEEPER & KAFKA INSTALL : A SINGLE NODE AND A MULTIPLE BROKER CLUSTER – 2016. http://www.bogotobogo.com/Hadoop/BigData_hadoop_Zookeeper_Kafka_single_node_Multiple_broker_cluster.php
- Getting Started with Apache Kafka for the Baffled, Part 1. http://www.shayne.me/blog/2015/2015-06-16-everything-about-kafka-part-1/
- Getting Started with Apache Kafka for the Baffled, Part 2. http://www.shayne.me/blog/2015/2015-06-25-everything-about-kafka-part-2/
- Apache Kafka, el sistema de mensajería distribuido de Linkedin. https://loquemeinteresadelared.wordpress.com/2014/07/15/apache-kafka/
- Reassigning Kafka topic partition Leaders. https://www.ibm.com/support/knowledgecenter/SSCVHB_1.2.0/admin/tnpi_reassign_partitions.html
- How to Rebalance Topics in a Kafka Cluster. https://blog.imaginea.com/how-to-rebalance-topics-in-kafka-cluster/
- Apache Kafka architecture. http://events.linuxfoundation.org/sites/events/files/slides/The%20Best%20of%20Apache%20Kafka%20Architecture.pdf
- Apache Kafka. https://www.infoq.com/articles/apache-kafka
-
- Apache Kafka, el sistema de mensajería distribuido de Linkedin.
-

LIBROS.

- Apache Kafka Cookbook. Saurabh Minni. PACKT publishing.
- Kafka. The Definitive Guide. Neha Narkhede. O'REILLY.
-

VIDEOS

- Balancing Apache Kafka Clusters. https://www.youtube.com/watch?v=t2pN4_HfGbY&feature=youtu.be.
-