



# Apache Kafka

V2022 Alfonso Tienda Braulio

# Sobre mí

# Alfonso Tienda

Freelance IT Solutions Architect

657657103

atienda@iprocuration.com

<https://www.linkedin.com/in/alfonsotienda/>

# Prologo



# Conocimientos, intereses

- ¿Por qué Kafka?
- ¿Cual es vuestro rol más habitual?

Repaso conocimientos:

- Docker
- Java
- Maven
- WSL2

# INTRODUCCION

# Patrón Publish Subscribe



# Patrón Publish Subscribe

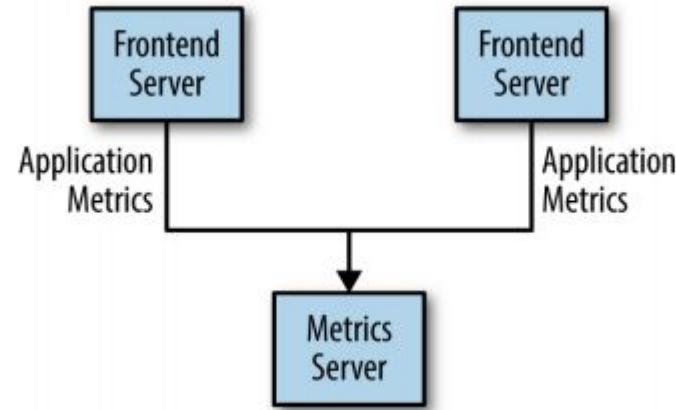
Publish/subscribe es un patrón caracterizado por que el que envía (Publisher) no lo hace para un receptor específico.

El publisher clasifica un mensaje y el receptor (subscriber) se suscribe para recibir ciertas clases de mensajes.

En ocasiones este patrón se apoya en Brokers como punto central para facilitar las cosas.



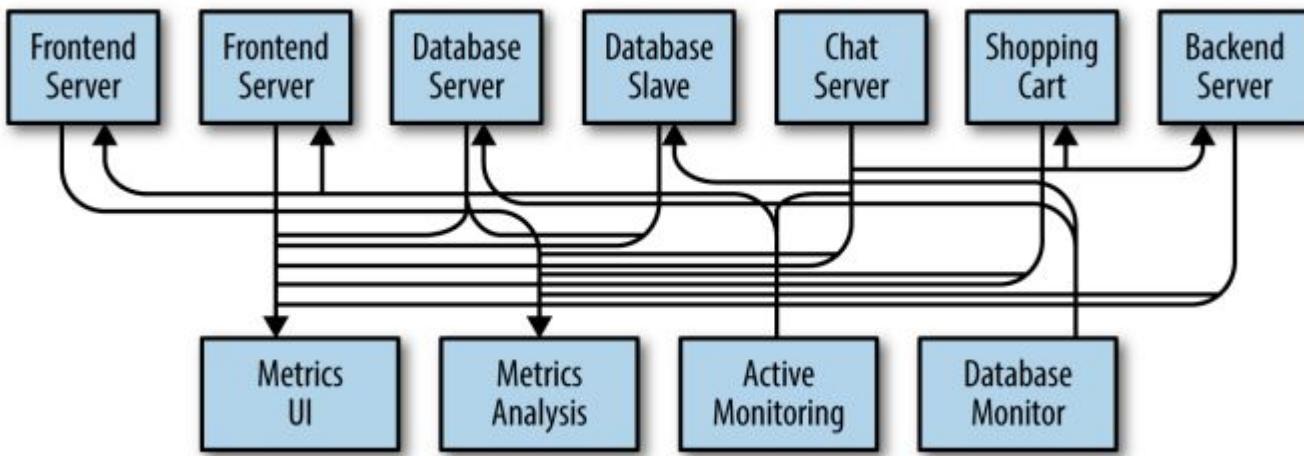
# Publish subscribe





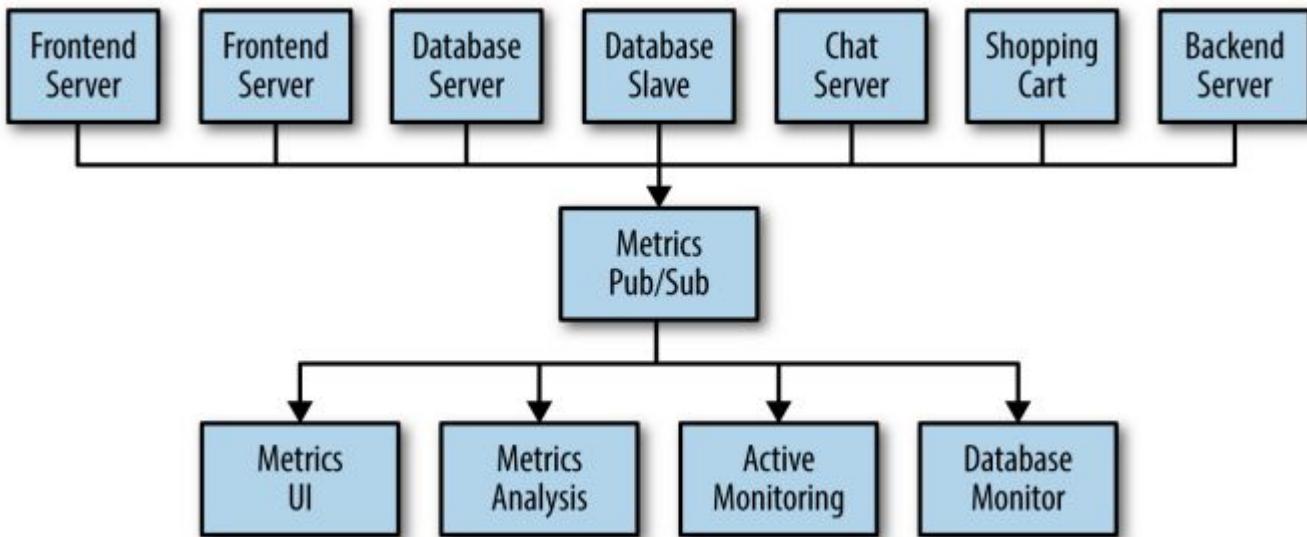
# Publish subscribe

---

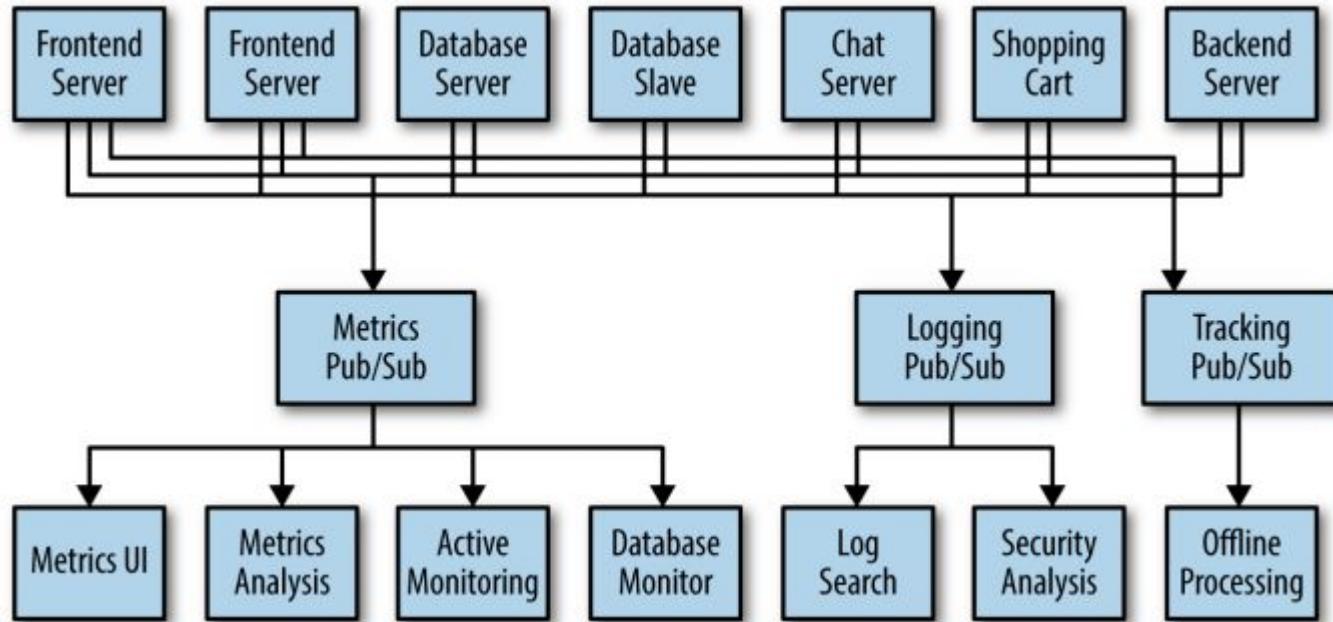




# Publish subscribe



# Sistemas de colas individuales

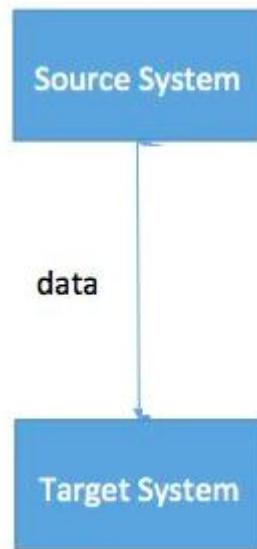




# Introducción a Kafka



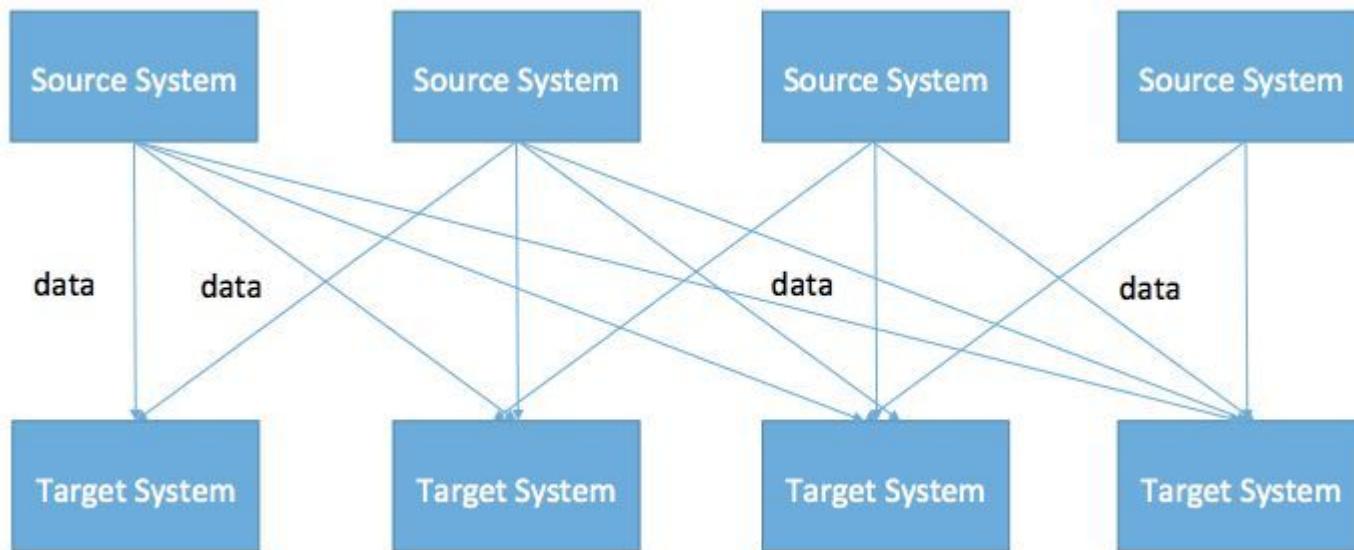
# Hasta ahora...





# Problema

---





# Solución Kafka





# Solución kafka



- Desacoplar los sistemas o subsistemas de los flujos de datos
- Coreografía de microservicios



# Kafka: ¿Quién es?

Data really powers everything that we do.

—Jeff Weiner, *CEO of LinkedIn*

- Creado por LinkedIn, ahora es un proyecto Apache mantenido fundamentalmente por Confluent.
- Distribuido, arquitectura tolerante a fallos
- Escalabilidad horizontal:
  - Cientos de brokers
  - Millones de mensajes por segundo
- High performance - latencia inferior a 10ms
- Usado por grandes compañías (80% de Fortune500)



airbnb

Walmart 

Uber

Linked 

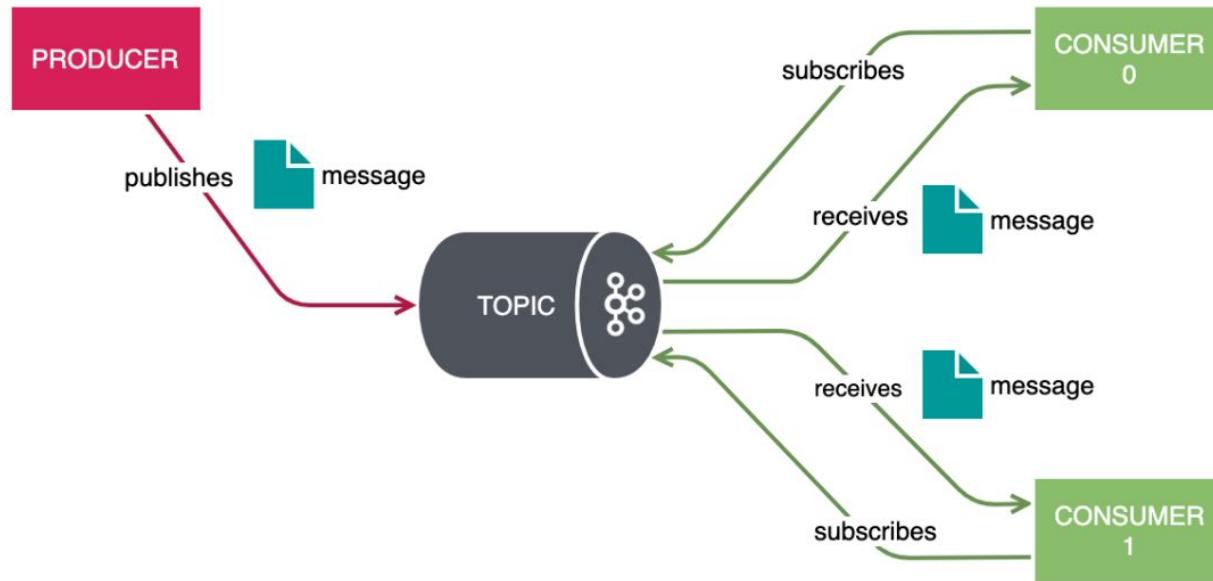
NETFLIX



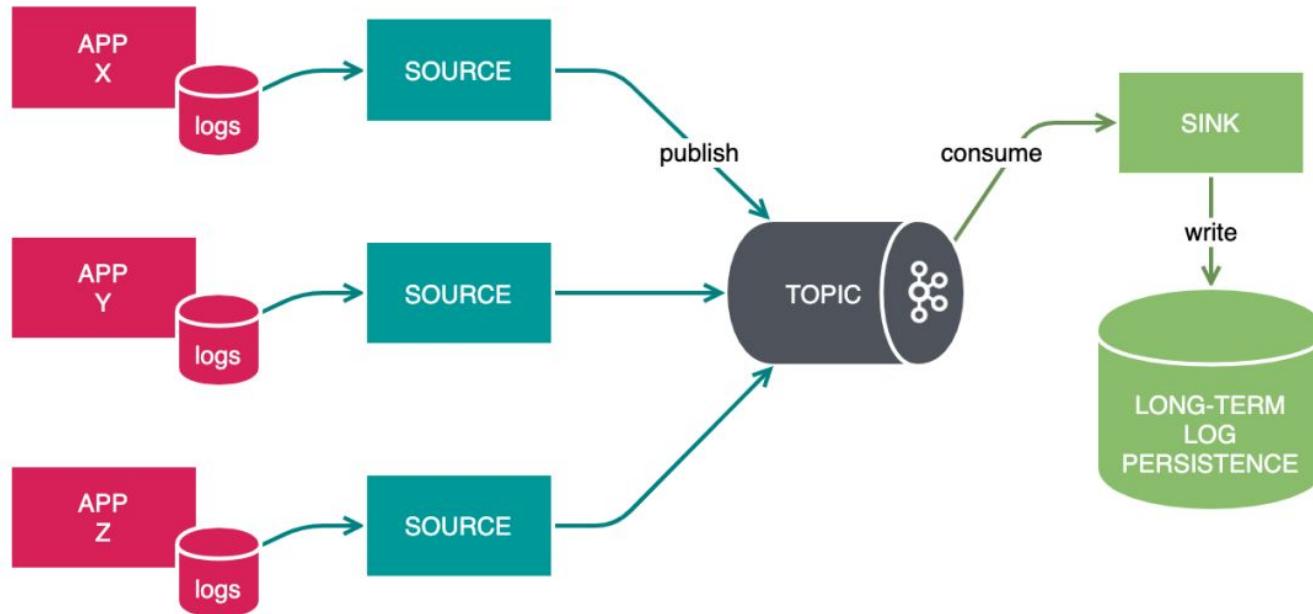
# Casos de uso

- Sistemas de mensajería
- Monitor de actividad
- Recopilar métricas o logs de distintas fuentes
- Procesamiento de streams (Kafka Streams Api / Spark)
- Desacoplamiento de sistemas
- Integración con Spark, Hadoop, Flink, Storm... para Big data.

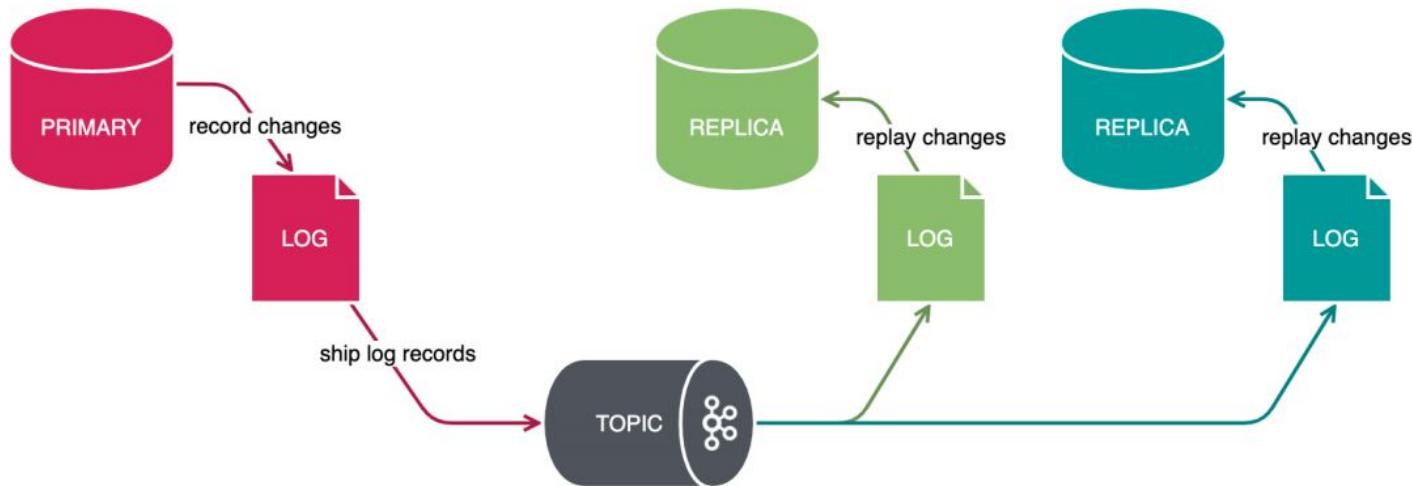
# Casos de uso: Publish - Subscribe



# Casos de uso: Agregación de logs

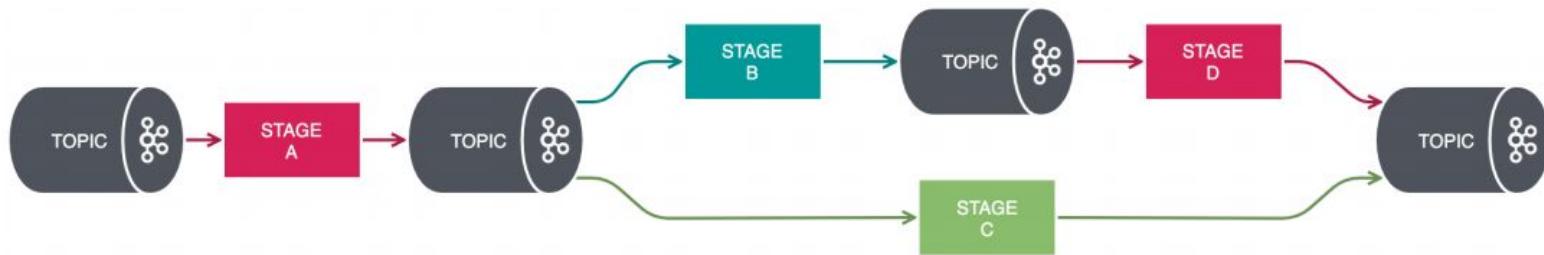


# Casos de uso: Envío de logs



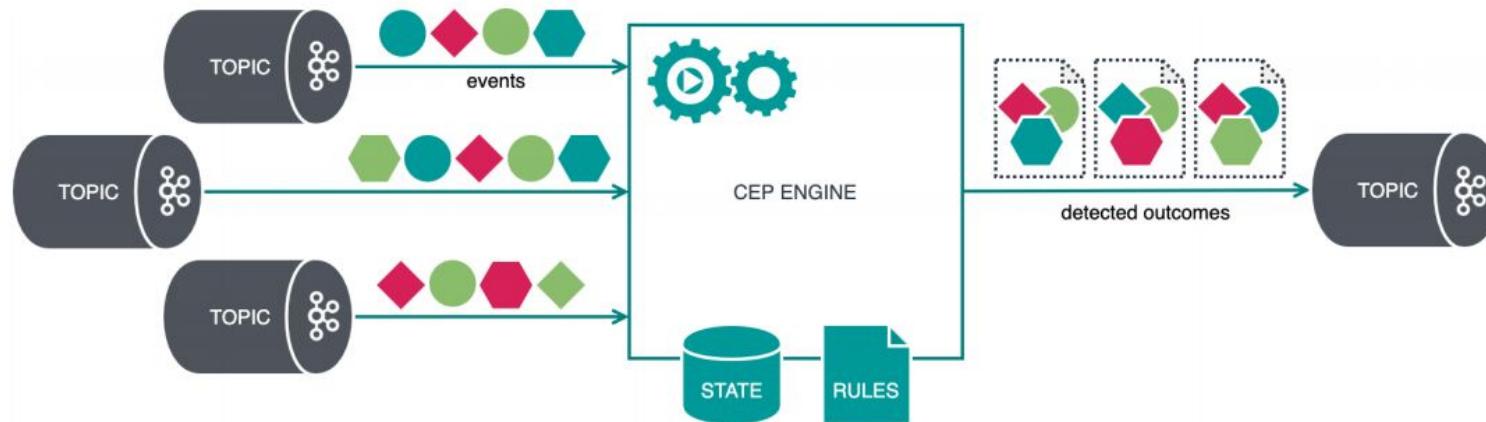


# Casos de uso: SEDA Pipeline



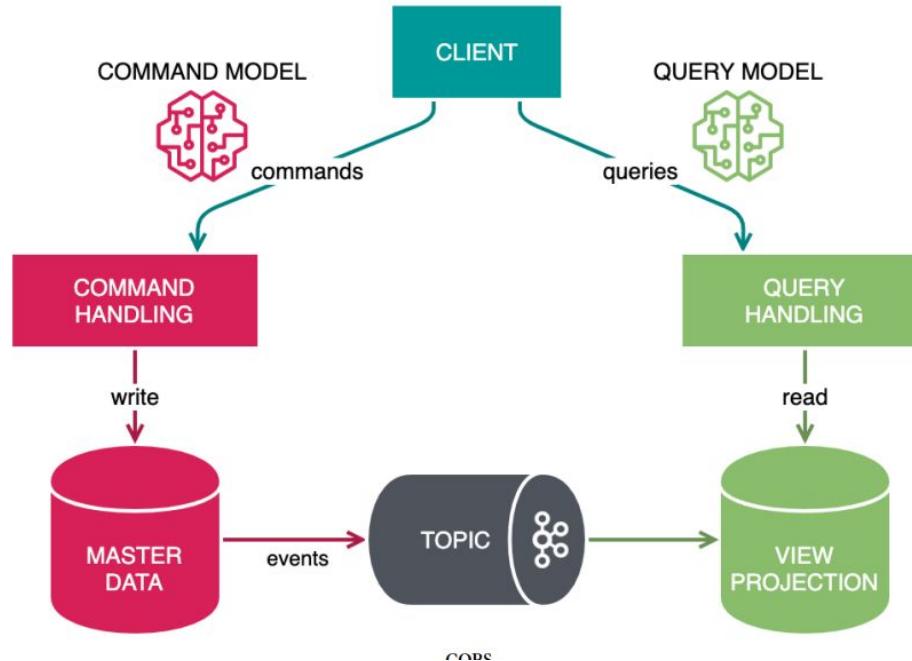


# Casos de uso: Procesamiento de eventos complejos





# Casos de uso: CQRS





# Casos de uso

**NETFLIX** Sistema de recomendaciones en tiempo real

**Uber** Recopila información de usuarios y conductores para predecir la demanda y fijar precios

**LinkedIn** Previene SPAM y mejora recomendaciones

# **Ventajas y desventajas de Kafka**



# Ventajas de Kafka

Permite desacoplar aplicaciones entre sí (útil en arquitecturas de microservicios)

Sistema escalable horizontalmente, tolerante a fallos y con baja latencia (Big Data)

Absorción de picos de carga que pueden ocurrir en el sistema

Permite construir aplicaciones que reaccionan a eventos en tiempo real

Referencia en la industria, lo usan las empresas más grandes del mundo

Garantías de entrega de mensajes exactly-once (exactamente una vez)

Transacciones a partir de Kafka 0.11



# Desventajas de Kafka

Kafka no es una tecnología que esté diseñada para manejar mensajes muy grandes (+ 1MB)

Pocas opciones de monitorización potentes

Opciones en Stream processing más limitadas que otras opciones como Apache Flink

Operaciones de rebalanceo de datos en particiones a veces necesarias, lo que impacta en el rendimiento del sistema

# Records



# Records / Registros / Mensajes ...

- Un registro es la unidad de persistencia más elemental en Kafka. En el contexto de la arquitectura impulsada por eventos, que es principalmente la forma en que se debe usar Kafka, un registro normalmente corresponde a algún evento de interés. Se caracteriza por los siguientes atributos:
  - Key: clave no única opcional, una especie de clasificador
  - value:: el payload del elemento
  - Headers: conjunto de pares clave-valor
  - Partition number
  - Timestamp

# Topics, partitions y offsets



# Topics

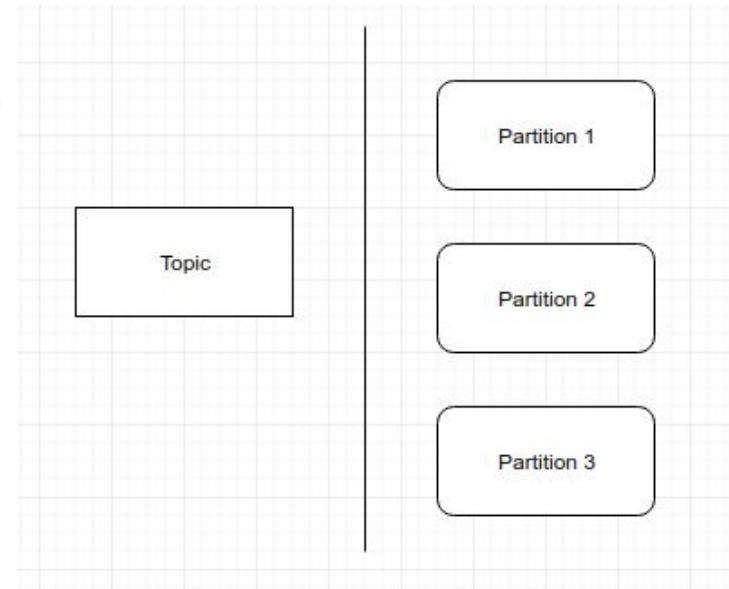
- Es un flujo de datos en concreto.
- Como una tabla en base de datos
- Puedes tener tantos como quieras
- Están identificados por un nombre (name)





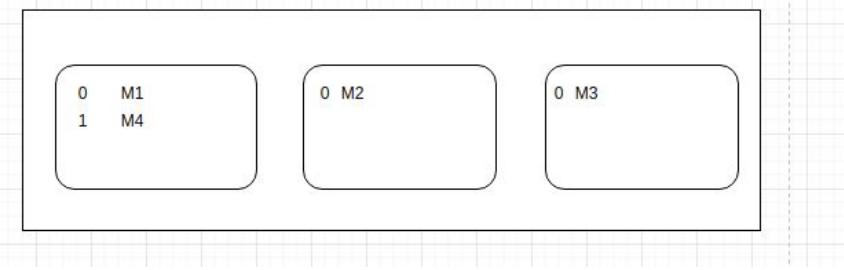
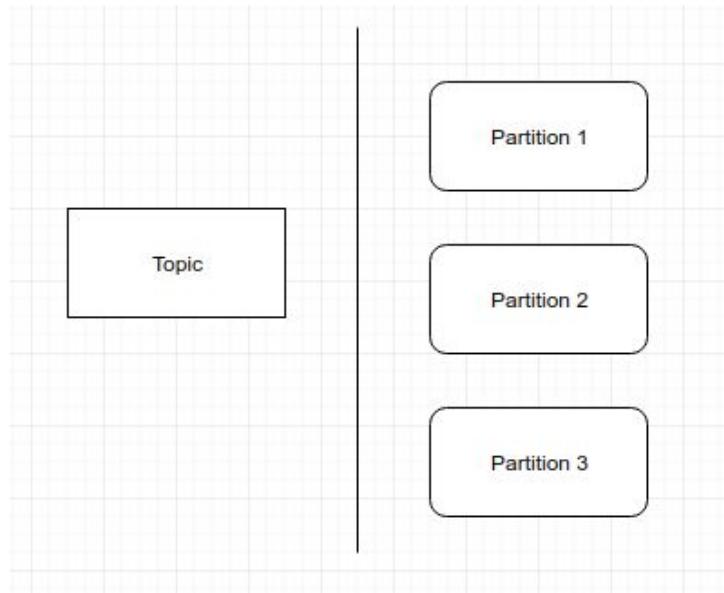
# Partitions y offsets

- Un “topic” está dividido en particiones (partitions)
- Cada partición está ordenada
- Cada elemento en una partición tiene un id incremental llamado offset
- Al crear un topic específico el número de particiones





# Offsets



0 1 2 3 4 5 6 7 8 9 10 ...

0 1 2 3 4 5 6 7...

0 1 2 3 4 5 6 7 8 9 10 11 12...



# Ejemplo: Sensores de temperatura

- Varios elementos de una fábrica envían temperaturas
- El topic será sensor\_temp
- Cada sensor envía la temperatura cada 20 segundos y contienen el id del sensor y la temperatura
- Elegimos 5 particiones en el topic

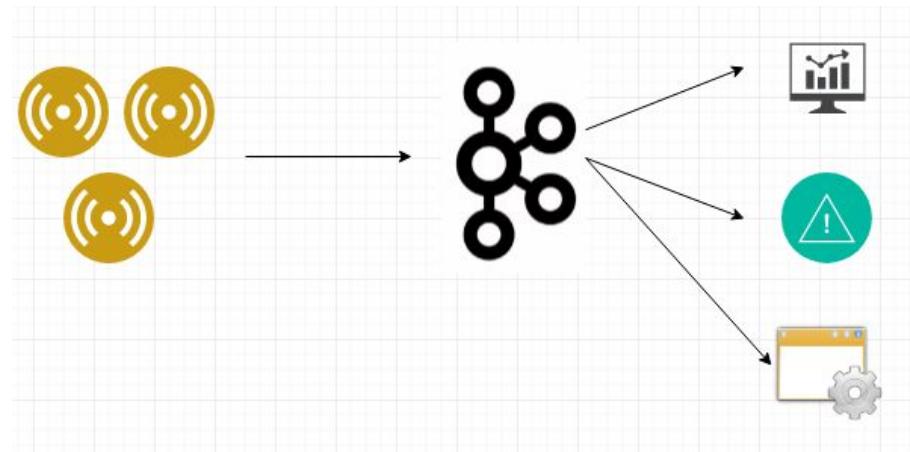




# Ejemplo: Sensores de temperatura

Usos:

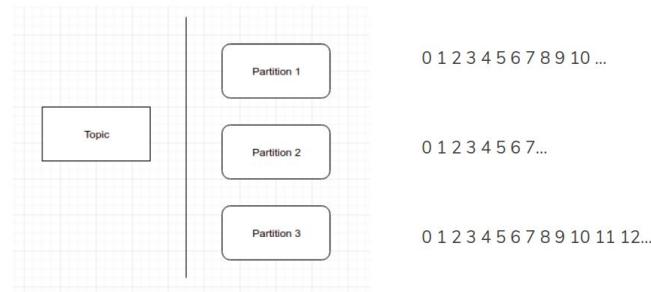
- Dashboards
- Notificaciones
- Acciones...





# Offset

- Los offsets tienen sentido dentro de una partición. El o1 de la partición 3 no es lo mismo que el o1 de la 2...
- El orden se garantiza dentro de una misma partición
- Los datos se mantienen durante un tiempo limitado (1 semana por defecto), pero los offset se siguen incrementando
- Una vez los datos se han escrito en una partición son inmutables, no se pueden modificar



# **Brokers y factor de replicación**



# Brokers

- Un Cluster Kafka se compone de múltiples brokers.
- Cada broker está identificado con un id
- Cada broker contiene ciertas particiones de ciertos topics, cada broker tiene ciertos datos, pero no todos los datos.
- Cuando te conectas a un broker, te conectas a todo el cluster
- Buena práctica seguir la regla de número de clusters =  $2n + 1$

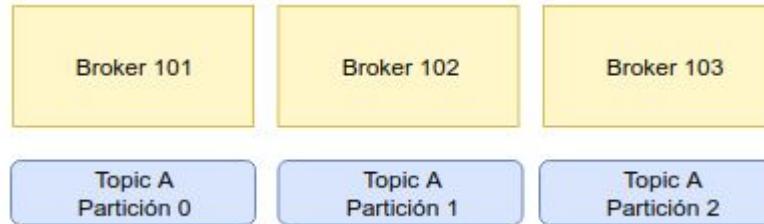




# Brokers y topics

Topic A, 3 particiones

Kafka intenta distribuirlas uniformemente.

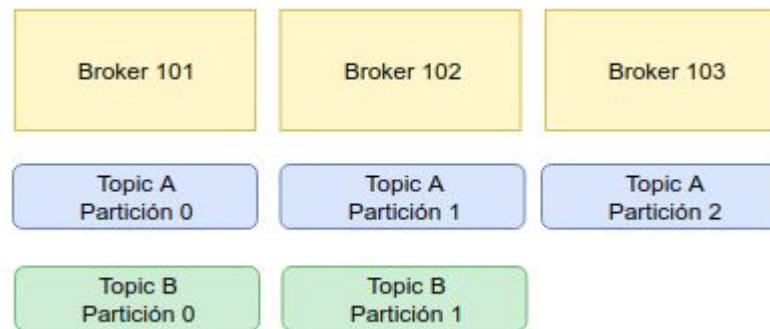




# Brokers y topics

Topic A, 3 particiones

Topic B, 2 particiones

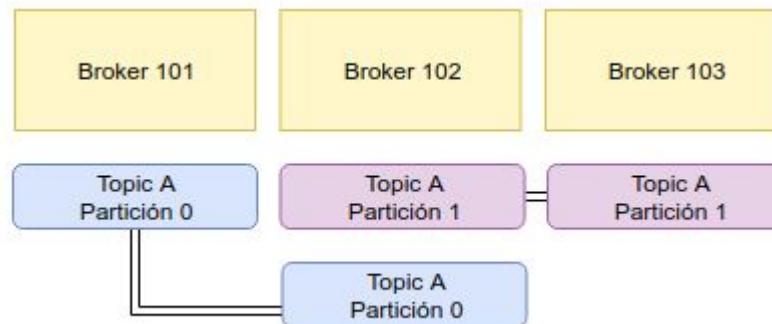




# Factor de replicación

Los topics deberían tener un factor de réplica  $\geq 2$  para que si un nodo (broker) cae se pueda seguir sirviendo el datos.

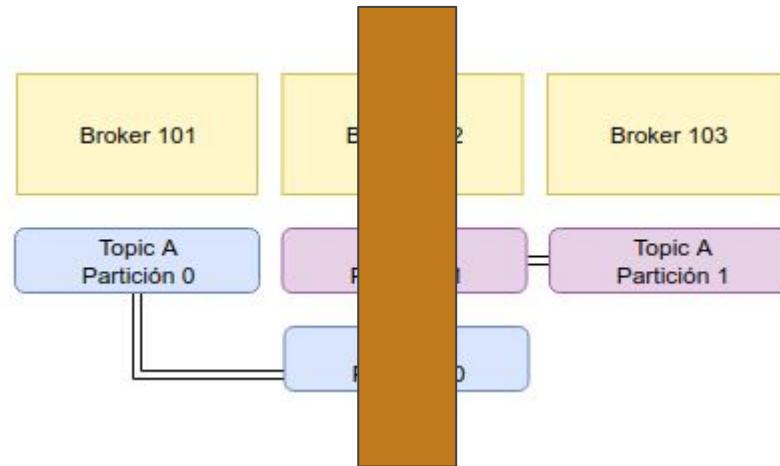
Topic A con factor de réplica de 2, dos copias de cada partición de los datos:





# Factor de replicación

En caso de fallo se sigue dando servicio





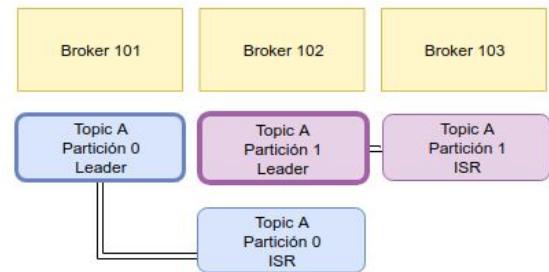
# Líder de la partición

Por cada partición sólo hay un líder de la misma (leader), los demás sirven para replicar los datos (ISR : in-sync-replica).

Sólo el leader recibe y sirve datos de la partición

Si cae el nodo lider, el siguiente ISR es el nuevo leader

Si vuelve a levantarse intentará volver a tomar el control



# Producers. Consumers

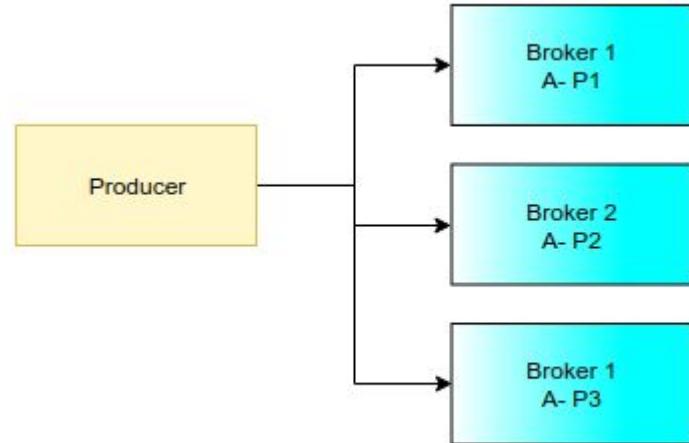


# Producers

Los producers escriben en los topics (que están formados por particiones)

Los producers saben automáticamente en qué broker y partición escribir. Eligen la partición por Round-Robin, realizando balanceo en cliente (como Ribbon)

En el caso de que un broker falle, los producers se recuperan automáticamente.





# Producers: keys

- Una key es alfanumérica, de cualquier tamaño.
- Si no se envía la key, los datos se envían en Round Robin
- Si se envía una key, todos los mensajes con la misma key van a la misma partición.
- Sirve fundamentalmente en aquellos mensajes que se han de ejecutar o trasmitir en orden.



# Producers: ack's

Los productores eligen el ack:

- acks=0, el producer no esperará confirmación (¿pérdida de datos?)
- acks=1, el leader confirmará que lo ha recibido (pérdida limitada)
- acks = all, el leader y las réplicas confirman la recepción (sin pérdida de datos)



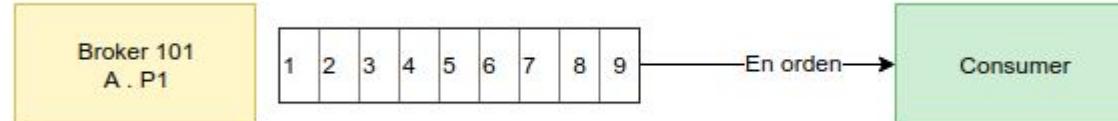
# Consumers

Los consumers leen datos de un topic (identificado por un nombre)

Los consumers saben de qué broker leer

En caso de fallo de un broker, los consumers saber recuperarse

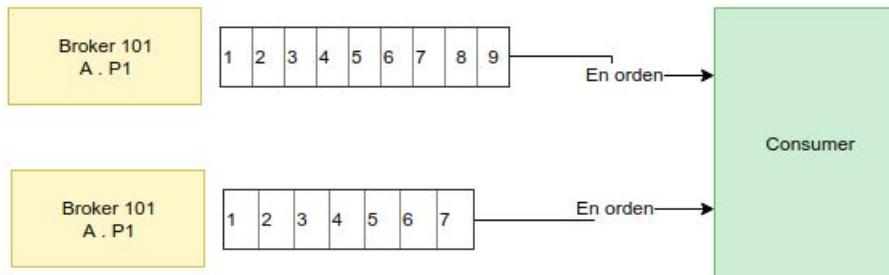
Los datos son leídos en orden **dentro de cada partición**





# Consumers

Los consumers pueden leer de varias particiones, no hay seguridad por el orden entre particiones.

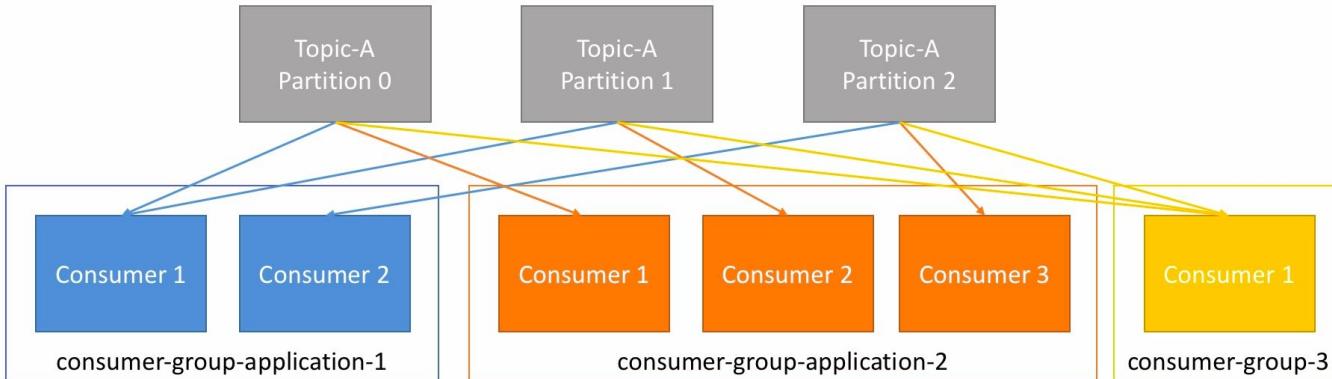




# Consumer Groups

Los consumers leen datos en consumer groups, garantizando que cada partición es consumida tan solo por un miembro.

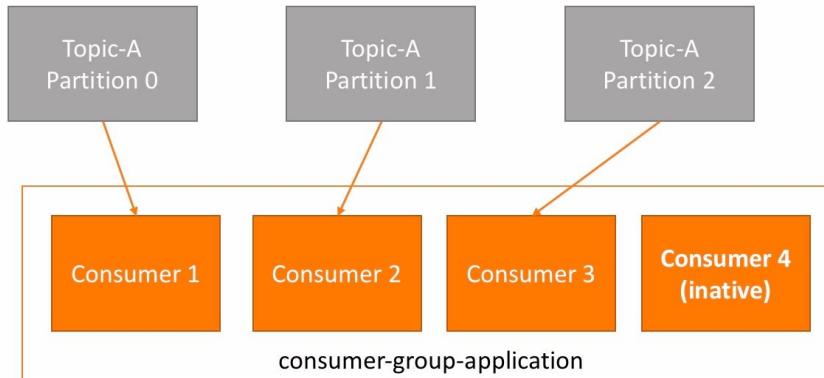
Cada consumidor en un grupo lee de particiones excluyentes, esto es sólo un consumidor, dentro del grupo, a cada partición como regla general.





# Consumer groups

Si tienes más consumidores que particiones, algunos consumidores estarán inactivos



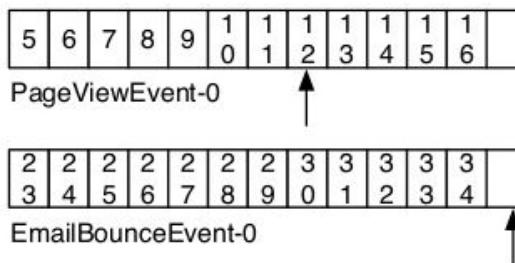


# Consumer Offsets

Kafka almacena los offsets que un consumer group ha leído, en un topic especial llamado \_\_consumer\_offsets.

Cuando un consumidor en un group ha procesado los datos recibidos de kafka, envía o comitea un mensaje con los offsets.

Si un consumidor cae, será capaz entonces de retomarlo donde lo había dejado.



Offset map  
GroupId: **audit-consumer**

PageViewEvent-0	12
EmailBounceEvent-0	35



# Delivery semantics

Un consumidor escoge cuando escribir o comitear los offsets. Hay 3:

- At most once:
  - Los offset se comitean tan pronto como el mensaje es recibido.
  - Si el procesamiento se corta, el mensaje se pierde.
- At least once:
  - El offset se comitea tras el procesamiento del mensaje.
  - El mensaje se recibe, se hace algo con él y se comitea el offset.
  - Si el procesamiento va mal, el mensaje se lee de nuevo.
  - El problema es que puede haber procesamientos duplicados (at least...), por lo que este procesamiento ha de ser idempotente
- Exactly once:
  - Sólo funciona de kafka a kafka, usando Kafka Streams API.

# Hardware y SO



# Hardware

- Disk throughput - HDD / SSD
- Capacidad de disco - Calcular
- Version 3: S3 .... Tiered storage
- Memoria: JVM - 150k mensajes / sec - 5gb heap - 8gb (min 4gb)
- Networking - a tope 10GB
- CPU no tiene tanta importancia - SSL



# SO

## Disk

- Ext4 - XFS (mucho mas seguro) - delayed block allocation

## Garbage Collector:

- G1GC -
  - MaxGCPauseMillis - tiempo de espera entre ciclos de garbage collection default 200ms se puede poner 20-50 ms
  - InitiatingHeapOccupancyPercent: por defecto es 45% - podemos reducirlos 35%
- Java Heap - Por lo general son unas 2000 páginas



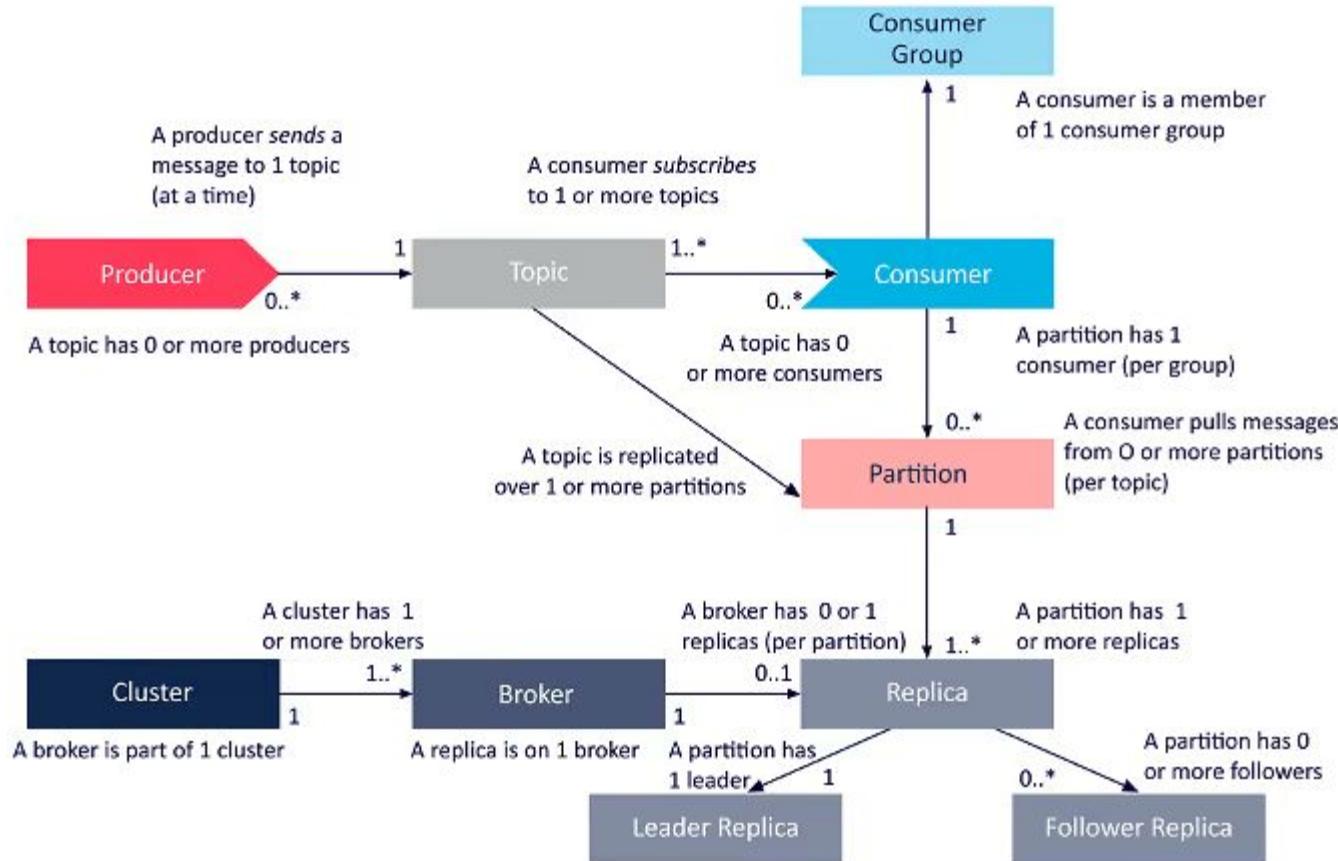
# /etc/sysctl.conf

“Desconectar la Virtual Memory” - Linux kernel > 3.5 vm.swappiness = 1

Vm.dirty\_background\_ratio = 5 (default 10)

Vm.dirty\_ratio = 60 / 80 (default 20)

# Resumen de arquitectura



# Más sobre Brokers. Zookeeper



# Kafka Broker Discovery

Cada broker Kafka tiene también la denominación de “Bootstrap Server”. Esto quiere decir que sólo te tienes que conectar a un broker para estar conectado al cluster entero.

Cada broker conoce los datos de todos los brokers, topics y particiones.

Al conectarse un cliente a un nodo del cluster, recibe estos metadatos.



# Zookeeper

Zookeeper gestiona los brokers, mantiene una lista de los mismos.

Asimismo ayuda en la elección de líder para cada partición.

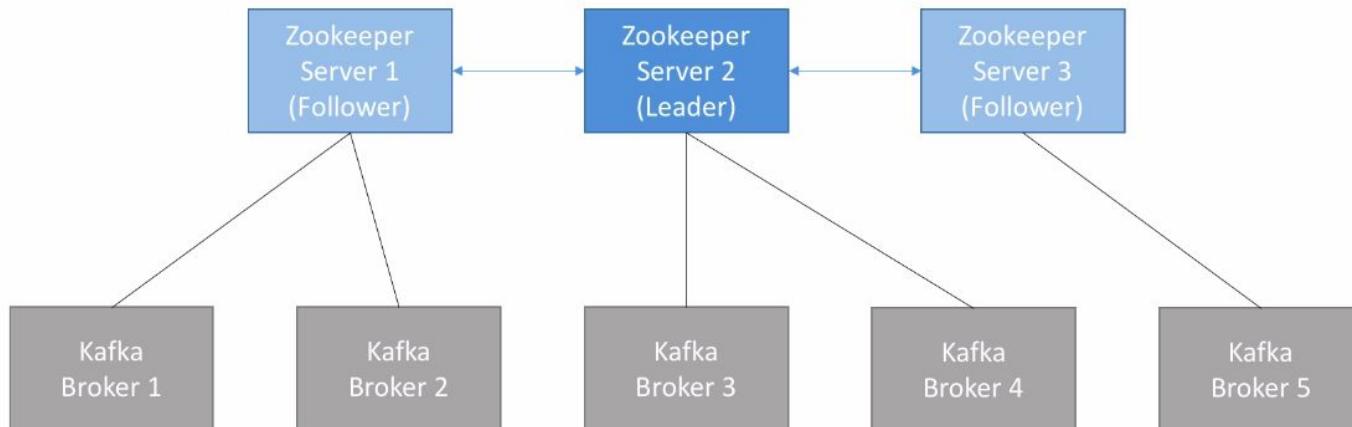
Envía notificaciones a Kafka en caso de haber cambios (nuevos topics, caídas o nuevos brokers, etc.)

Es necesario para el funcionamiento de Kafka. Zookeeper cumple la regla de  $2n+1$ .

Zookeeper tiene un líder (gestiona las escrituras) y el resto de servidores son followers (gestionan las lecturas)

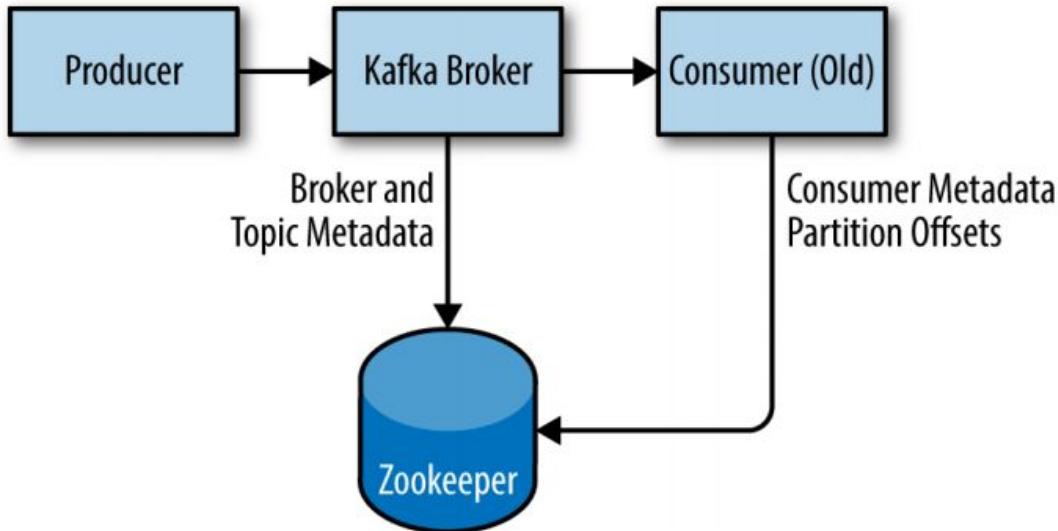


# Zookeeper





# Zookeeper



# A recordar

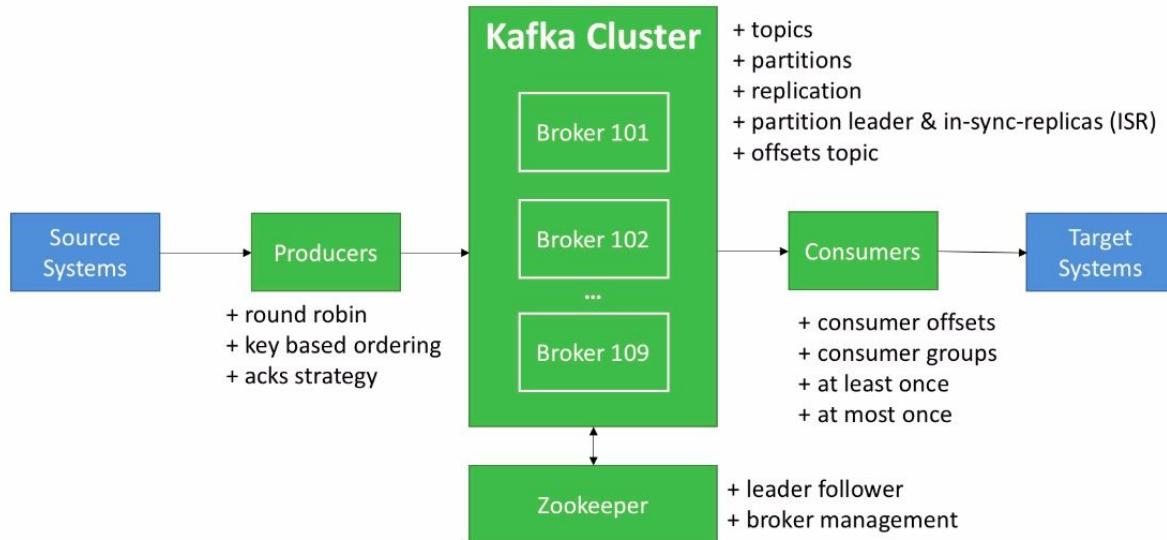


# Garantías

- Los mensajes se añaden a una partición de un topic en el orden que son enviados
- Los consumidores leen mensajes en el orden que se han guardado en una partición de un topic.
- Con un factor de réplica de N, los producers y consumers tolerarán N-1 brokers caídos.
- Un factor de 3 es común porque permite la caída de uno y el mantenimiento de otro.
- La misma key va a la misma partición.



# Kafka



# PRIMEROS PASOS

# Instalando Kafka

# Instalar Kafka en la máquina linux

Descargar e instalar java 8: sudo apt install openjdk-8-jdk

Descargar y extraer los binarios kafka: <https://kafka.apache.org/downloads>

Probar: bin/kafka-topics.sh (for example)

Editar PATH (opcional) (en ~/.bashrc) PATH="\$PATH:/your/path/to/your/kafka/bin"

Editar properties: (opcional)

zookeeper.properties: dataDir=/your/path/to/data/zookeeper

server.properties: log.dirs=/your/path/to/data/kafka

Arrancar: zookeeper-server-start.sh config/zookeeper.properties

bin/kafka-server-start.sh config/server.properties



# Configuraciones avanzadas del Broker Kafka

broker.id

Every Kafka broker must have an integer identifier, which is set using the broker.id configuration. By default, this integer is set to 0, but it can be any value. The most important thing is that the integer must be unique within a single Kafka cluster.

port

The example configuration file starts Kafka with a listener on TCP port 9092. This can be set to any available port by changing the port configuration parameter.



# Configuraciones avanzadas del Broker Kafka

## zookeeper.connect

The location of the Zookeeper used for storing the broker metadata is set using the `zookeeper.connect` configuration parameter. The example configuration uses a Zookeeper running on port 2181 on the local host, which is specified as `localhost:2181`. The format for this parameter is a semicolon-separated list of `hostname:port/path` strings, which include:

- `hostname`, the hostname or IP address of the Zookeeper server.
- `port`, the client port number for the server.
- `/path`, an optional Zookeeper path to use as a chroot environment for the Kafka cluster. If it is omitted, the root path is used.

# Kafka CLI



# Kafka Tools - CLI

Estas herramientas son geniales, (raro raro, pero lo son) , por lo que hay que usarlas ya que:

- Están bien documentadas
- Son fáciles de usar
- No contienen errores obvios.



# Creando un topic

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --topic first_topic --create
```

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --topic first_topic --create --partitions  
3
```

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --topic first_topic --create --partitions  
3 --replication-factor 2
```

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --topic first_topic --create --partitions  
3 --replication-factor 1
```



# Listing topics

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --list
```

```
first_topic
```



# Describing topics

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --topic first_topic --describe
```

```
Topic:first_topic    PartitionCount:3    ReplicationFactor:1  
Configs:segment.bytes=1073741824
```

```
Topic: first_topic    Partition: 0    Leader: 0    Replicas: 0    Isr: 0
```

```
Topic: first_topic    Partition: 1    Leader: 0    Replicas: 0    Isr: 0
```

```
Topic: first_topic    Partition: 2    Leader: 0    Replicas: 0    Isr: 0
```

Leader es el líder de cada partición. Isr las réplicas.



# EJERCICIO: Crear otro topic

Ejercicio: Crear otro topic (ej: topic\_segundo)



# Borrar un topic

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --topic topic2 --delete
```



# Producer CLI: kafka-console-producer

Campos requeridos:

--broker-list <String: broker-list> REQUIRED: The broker list string in the form HOST1:PORT1,HOST2:PORT2.

--topic <String: topic> REQUIRED: The topic id to produce messages to.



# Producer CLI: kafka-console-producer

```
bin/kafka-console-producer.sh --topic kkk --broker-list localhost:9092
```

```
>Hola Mundo
```

```
>Mi primer mensaje
```

```
>Hola de nuevo
```

```
>^C
```



# Producer CLI: Properties

```
bin/kafka-console-producer.sh --topic kkk --broker-list localhost:9092 --producer-property  
acks=all
```

```
>Hola con acks
```

```
>^C
```



# Producer CLI

```
bin/kafka-console-producer.sh --topic no-existente --broker-list localhost:9092
```

```
>holo, el topic no existía
```

```
[2019-10-28 19:38:26,891] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 3 :  
{no-existente=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
```

```
>^C
```

```
$ bin/kafka-topics.sh --bootstrap-server localhost:9092 --list
```

```
kkk
```

```
no-existente
```



# Consumer CLI

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic kkk
```

(Parece que no ocurre nada, mantenerlo y producir mensajes)

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic kkk
```

>Hola....

Aparecerán los mensajes en la terminal de consumer

Es un stream, es real-time



# Consumer CLI ¿Cómo leer todos los mensajes?

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic kkk --from-beginning
```

Hola Mundo

Mi primer mensaje

Hola....

Hola de nuevo

Hola con acks

Comprobar que el orden ES POR PARTICIÓN



# Consumer CLI: Consumer Groups

Los consumidores suelen ser parte de un grupo y un grupo es básicamente un ID.

Arrancar esto en dos terminales:

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic kkk --group grupo1
```

Comprobar que los mensajes sólo los capture uno del grupo.

Comprobar que hay tres particiones y si pongo tres cada uno lee de una partición. Luego “tiramos” un consumidor y veremos como balancea.



# Consumer CLI: Consumer Groups y offset

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic kkk --group grupo-nuevo --from-beginning
```

Veremos todos los mensajes. Lo paramos, lo volvemos a arrancar y veremos como no nos muestra ninguno, esto es porque el offset de grupo-nuevo ya está en “todos los mensajes anteriores leídos”



# Groups CLI: kafka-consumer-groups

This tool helps to list all consumer groups, describe a consumer group, delete consumer group info, or reset consumer group offsets

```
bin/kafka-consumer-groups.sh --list --bootstrap-server localhost:9092
```

```
grupo-nuevo
```

```
grupo1
```

```
bin/kafka-consumer-groups.sh --describe --group grupo1 --bootstrap-server localhost:9092
```

Vemos los offsets... el lag indica lo que le falta por leer, long-end offset es lo que le falta por leer...



# Groups CLI: kafka-consumer-groups

--reset-offsets

Reset offsets of consumer group.

Supports one consumer group at the time, and instances should be inactive

Has 2 execution options: --dry-run (the default) to plan which offsets to reset, and --execute to update the offsets. Additionally, the --export option is used to export the results to a CSV format.

You must choose one of the following reset specifications: --to-datetime, --by-period, --to-earliest, --tolatest, --shift-by, --from-file, --to-current.

To define the scope use --all-topics or --topic. One scope must be specified unless you use '--from-file'.

```
bin/kafka-consumer-groups.sh --reset-offsets --to-earliest --group grupo-nuevo --bootstrap-server localhost:9092 --all-topics --execute
```

Jugar leyendo un consumidor from-beggining, describiendo los grupos (ver lag, etc)... probar shift-by (2, -2...)



# UI: Kafka Tools

<http://www.kafkatool.com/download.html>

# JAVA API



# Generate a maven project

```
mvn archetype:generate \  
  -DgroupId=com.iprocuratio.kafka \  
  -DartifactId=java-api \  
  -DarchetypeArtifactId=maven-archetype-quickstart \  
  -DinteractiveMode=false
```



# Project java - api

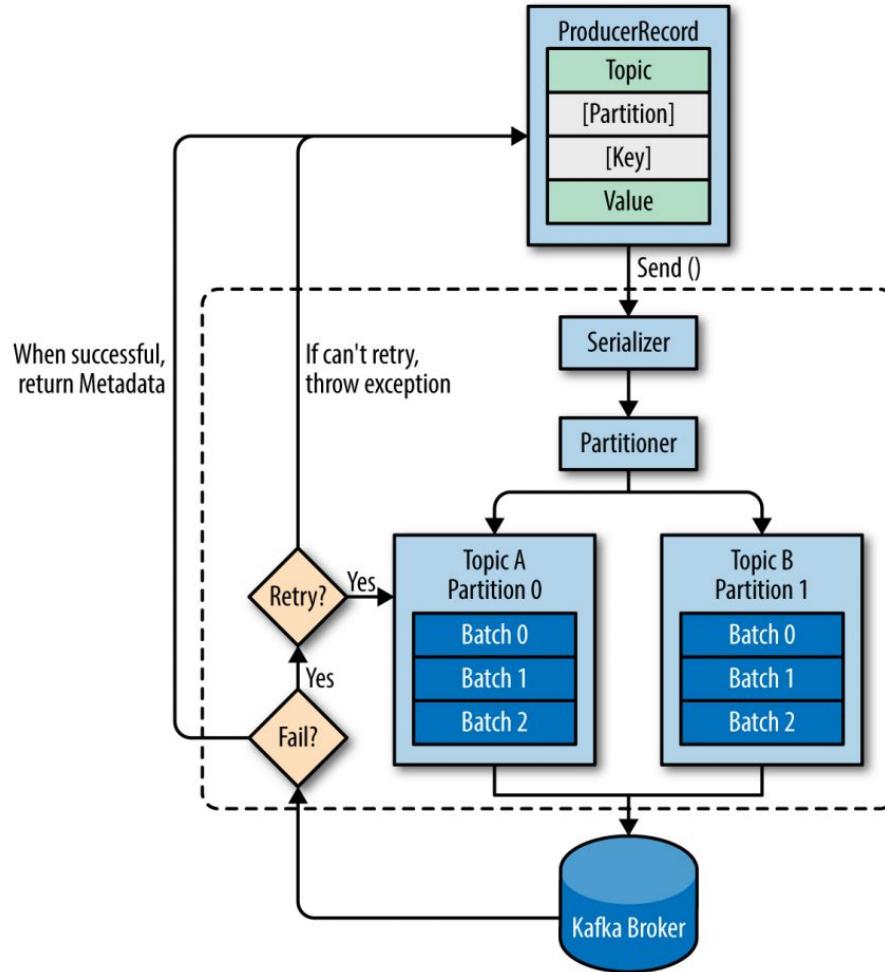
# Compatibilidad en KAFKA

# Compatibilidad bidireccional

- Desde Kafka 0.10.2 (Jul -17) los mensajes son retrocompatibles de forma bidireccional, mediante un sistema de versionado
- Esto supone:
  - Que un cliente Kafka 1.1 (más antiguo) puede comunicarse con un Broker 2.1 (más nuevo)
  - Que un cliente Kafka 2.0 (más nuevo) puede comunicarse con un broker 1.0 (más antiguo)
- La nueva compatibilidad bidireccional del cliente de Kafka desacopla las versiones de los broker de las versiones del cliente. Esto en la práctica permite que usemos las últimas versiones de clientes o brokers independientemente del resto de versiones.
- Confluent Platform proporciona una utilidad de línea de comandos para leer las versiones API de los correderos, el script kafka-broker-api-versions.sh.

```
1 cmccabe@aurora:~/confluent-3.2.0/bin> ./kafka-broker-api-versions.sh
2 Missing required argument "[bootstrap-server]"
3 Option                      Description
4 -----                      -----
5 --bootstrap-server <String: server(s)> REQUIRED: The server to connect to.
6   to use for bootstrapping>
7 --command-config <String: command      A property file containing configs to
8   config property file>                 be passed to Admin Client.
```

# Producers





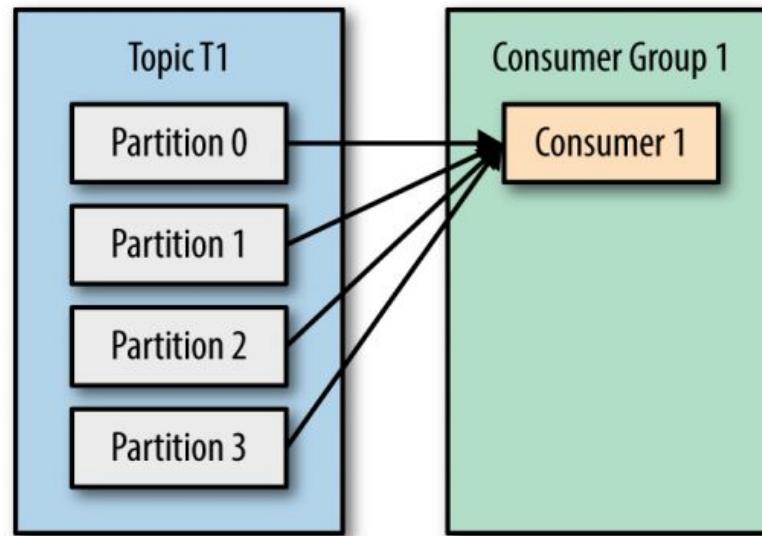
# Revisión de propiedades

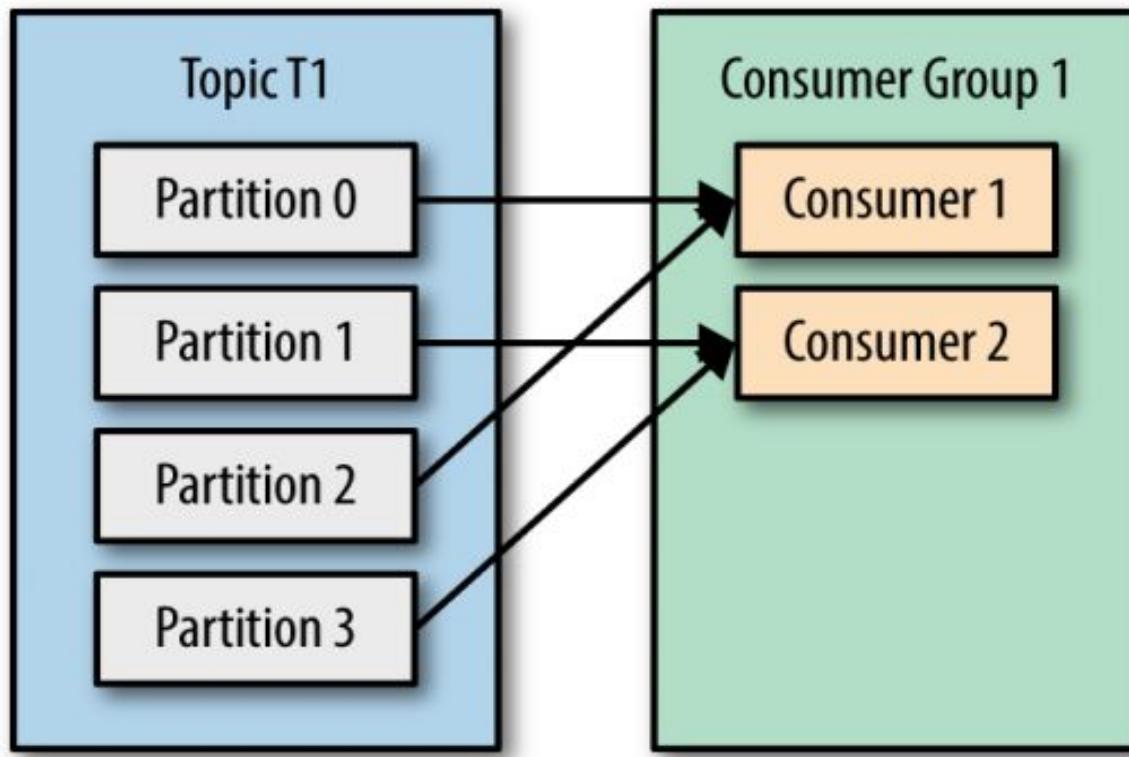
<https://kafka.apache.org/documentation/#producerconfigs>

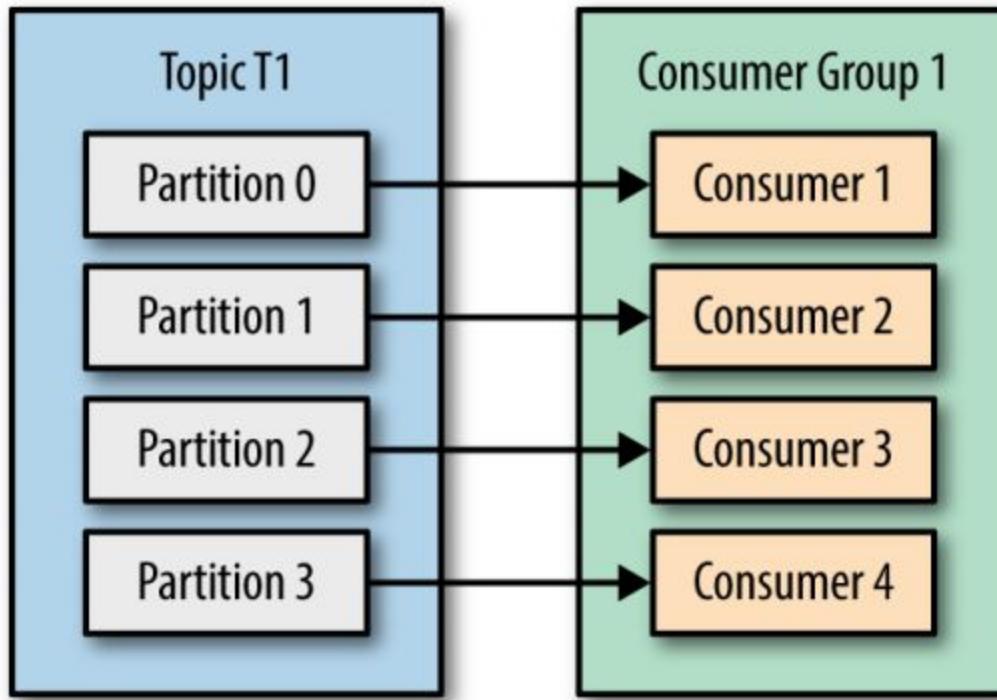
# Consumers: Revisión

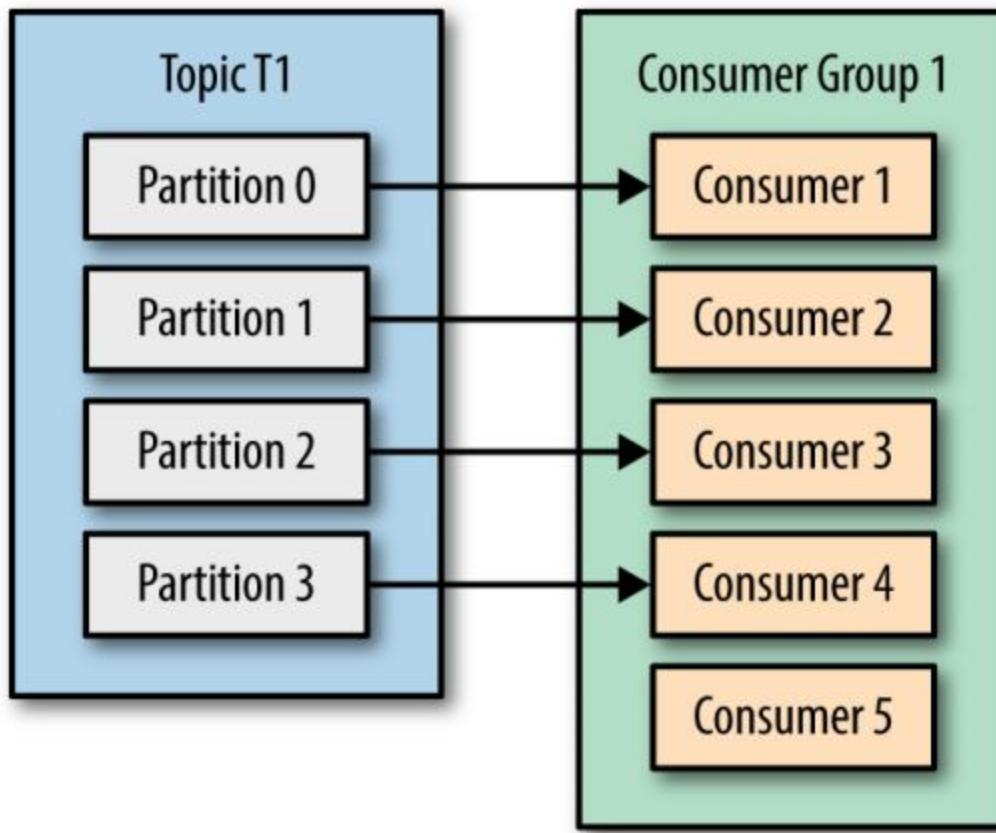


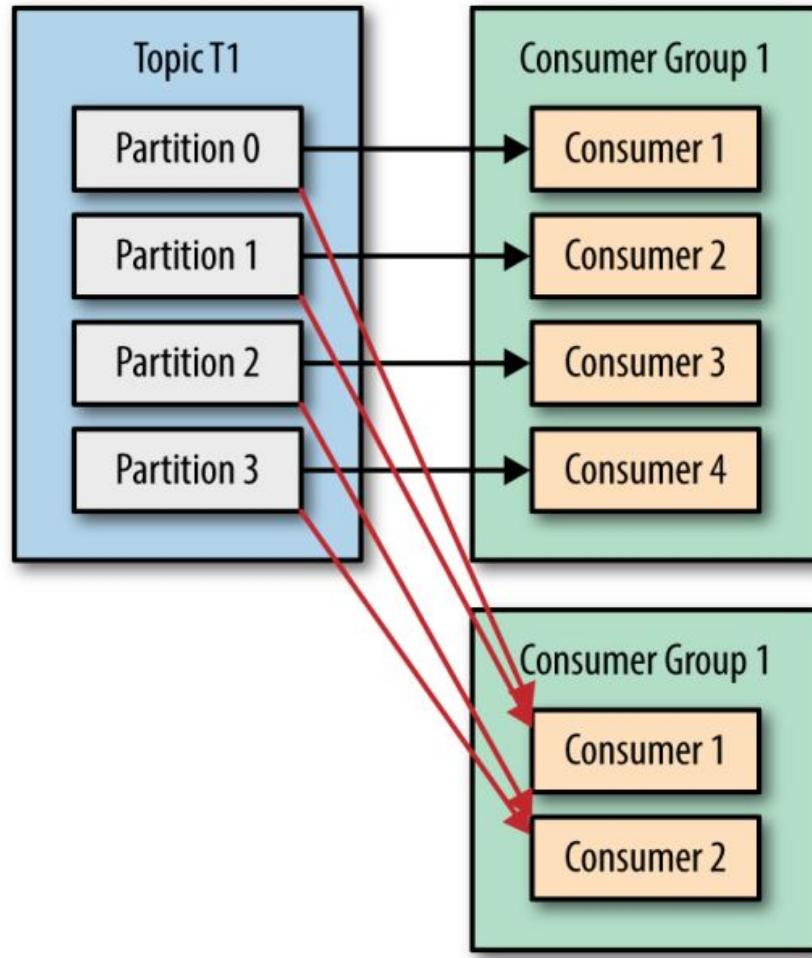
# Consumer Groups













# Segmentos

La partición puede ser la unidad estándar de almacenamiento en Kafka, pero no es el nivel más bajo de abstracción proporcionado. Cada partición se subdivide en segmentos.

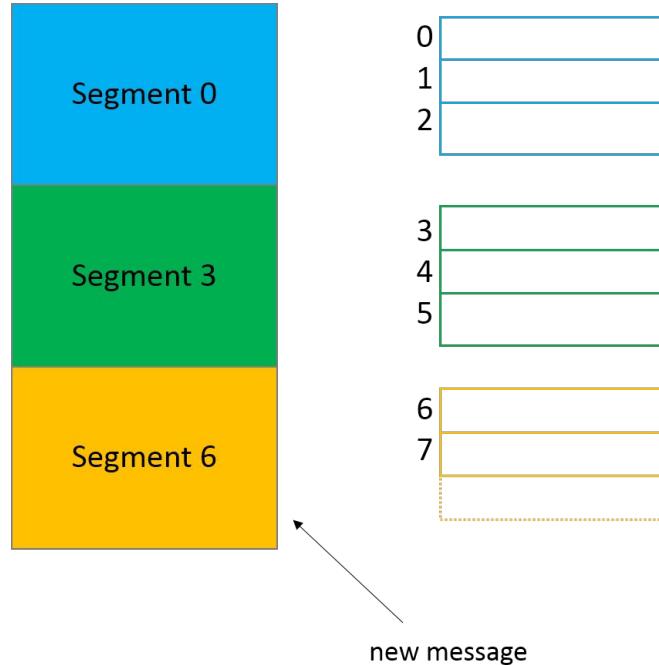
Un segmento es simplemente una colección de mensajes de una partición. En lugar de almacenar todos los mensajes de una partición en un solo archivo (piense nuevamente en la analogía del archivo de registro), Kafka los divide en fragmentos llamados segmentos. Hacer esto ofrece varias ventajas.

Lo más importante, **facilita la depuración de datos**. Como se introdujo anteriormente, la partición es inmutable desde la perspectiva del consumidor. **Pero Kafka aún puede eliminar los mensajes en función de la "Política de retención" del topic**. Eliminar segmentos es mucho más simple que eliminar cosas de un solo archivo, especialmente cuando un productor podría estar introduciendo datos en él.



# Segmentos

Kafka siempre escribe los mensajes en estos archivos de segmento bajo una partición. Siempre hay un segmento activo en el que Kafka escribe. Una vez que se alcanza el límite de tamaño del segmento, se crea un nuevo archivo de segmento y se convierte en el nuevo segmento activo.





# Política de retención

Apache Kafka proporciona retención a nivel de segmento en lugar de a nivel de mensaje. Por lo tanto, Kafka continúa eliminando segmentos de su final ya que violan las políticas de retención.

Apache Kafka nos proporciona las siguientes políticas de retención:

- Retención basada en el tiempo
- Retención basada en el tamaño



# Retención basada en el tiempo

Configuramos el tiempo máximo que un segmento (por lo tanto, los mensajes) puede vivir. Una vez que un segmento ha abarcado el tiempo de retención configurado, se marca para su eliminación o compactación según la política de limpieza configurada. El tiempo de retención predeterminado para los segmentos es de 7 días.

Parámetros de configuración:

- Configura el tiempo de retención en milisegundos.
  - log.retention.ms=1680000 (default null)
- Se usa si log.retention.ms no está configurado
  - log.retention.minutes=1680 (default null)
- Se utiliza si log.retention.minutes no está establecido
  - log.retention.hours=168 (default 168)



# Retención basada en el tamaño

Configuramos el tamaño máximo de una estructura de datos de registro para una partición de tema. Una vez que el tamaño del registro alcanza este tamaño, comienza a eliminar segmentos de su extremo. Esta política no es popular, ya que no proporciona una buena visibilidad sobre la caducidad de los mensajes. Sin embargo, puede ser útil en un escenario en el que necesitamos controlar el tamaño de un registro debido al espacio de disco limitado.

Parámetros de configuración:

- Configura el tamaño máximo de un registro, en bytes
  - `log.retention.bytes=104857600`



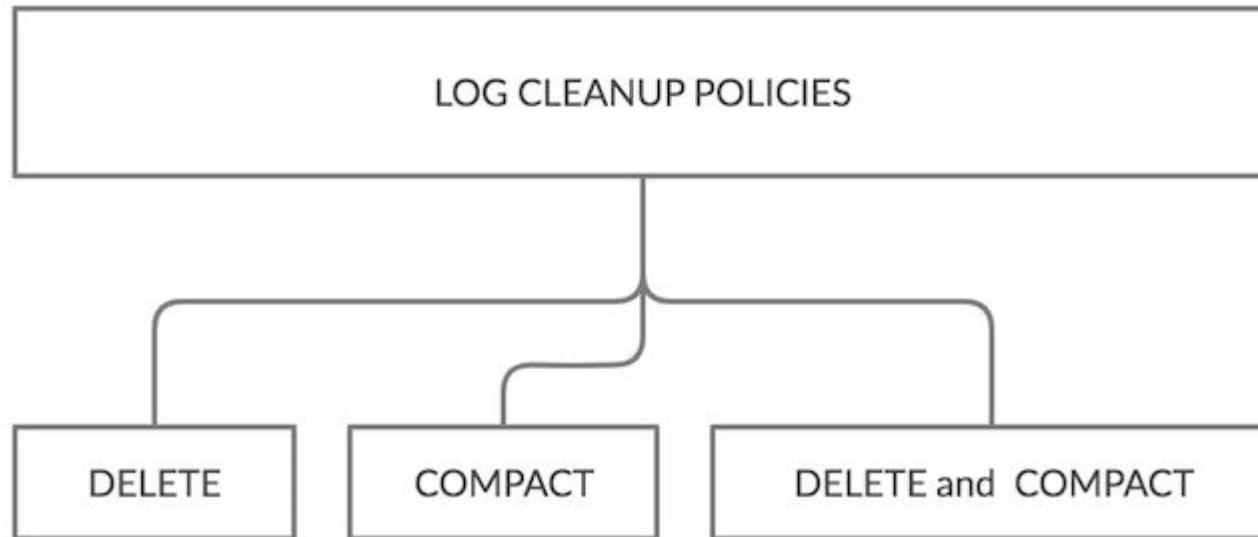
# Políticas de limpieza de mensajes

En Kafka, a diferencia de otros sistemas de mensajería, los mensajes sobre un topic no se eliminan inmediatamente después de ser consumidos. En cambio, la configuración de cada topic determina cuánto espacio está permitido y cómo se administra.

El concepto de hacer que los datos caduquen se llama Clean Up. Es una configuración de nivel de topic.



# Políticas de limpieza de mensajes





# Delete Policy

Esta es la política de limpieza predeterminada. Esto descartará los segmentos antiguos cuando se haya alcanzado su tiempo de retención o límite de tamaño.



# Compactación

La compactación de registros es un mecanismo para proporcionar una retención por registro de grano más fino, en lugar de la retención basada en el tiempo de grano más grueso y sirve para los mensajes de clave-valor. La idea es eliminar selectivamente los registros donde tenemos una actualización más reciente con la misma clave. De esta forma, se garantiza que el registro tendrá al menos el último estado para cada clave.

Para simplificar esta descripción, Kafka elimina los registros antiguos cuando hay una versión más nueva con la misma clave en el registro de partición.

Offset	1	2	3	4
Key	p3	p5	p3	p6
Value	10\$	7\$	11\$	25\$

latest-product-price-0 partition

Record with the key p3 is duplicated. So  
remove the old one with offset 1

Offset	2	3	4
Key	p5	p3	p6
Value	7\$	11\$	25\$

latest-product-price-0 partition



# Ejemplo

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --topic latest-product-price --replication-factor 1 --partitions 1 --config "cleanup.policy=compact" --config "delete.retention.ms=100" --config "segment.ms=100" --config "min.cleanable.dirty.ratio=0.01"

bin/kafka-console-producer.sh --broker-list localhost:9092 --topic latest-product-price --property parse.key=true --property key.separator=:

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic latest-product-price --property key.separator=: --property print.key=true
```



# Ejemplo

Producir cosas con la misma key...

Se reproducirán en el consumidor...

Parar el consumidor...

Producir cosas con la misma keyf

Ver cómo se ha compactado... a nivel de segmento



# Revisión de propiedades

<https://kafka.apache.org/documentation/#consumerconfigs>

# Rebalanceo de particiones (asignación a consumidores)



# Rebalanceo de particiones

Dentro de un grupo de consumidores, Kafka cambia la propiedad de la partición de un consumidor a otro en determinadas ocasiones. El proceso de cambiar la propiedad de la partición entre los consumidores se denomina rebalanceo.



# Rebalanceo de particiones

Los rebalanceos son importantes porque brindan al grupo de consumidores una alta disponibilidad y escalabilidad (lo que nos permite agregar y eliminar consumidores de manera fácil y segura), pero en el curso normal de los eventos son bastante indeseables.

La forma en que los consumidores mantienen la membresía en un grupo de consumidores y la asignación de las particiones a ellos es enviando heartbeats a un broker de Kafka designado como coordinador del grupo (este broker puede ser diferente para diferentes grupos de consumidores). Mientras el consumidor envíe latidos a intervalos regulares, se supone que está vivo y procesa los mensajes de sus particiones.

.



# Rebalanceo de particiones

El rebalanceo ocurre en los siguientes eventos:

1. Un nuevo consumidor se une a un grupo de consumidores: Se activa un rebalanceo de partición cuando agregamos un nuevo consumidor al grupo. El nuevo consumidor comienza a consumir mensajes de particiones previamente consumidas por otro consumidor.
2. Un consumidor existente se apaga: Se activa un reequilibrio de partición cuando un consumidor se apaga o falla. Las particiones que solía consumir serán consumidas por uno de los consumidores restantes
3. Se modifica el topic: También se desencadena un rebalanceo de partición cuando se modifican los topics que consume el consumer group. Por ejemplo, se agregan nuevas particiones al topic.

# Rebalanceo de particiones (dentro de brokers, tamaño)



# Rebalancear las particiones de kafka

Tras algún tiempo, las particiones desiguales y los brokers desequilibrados pueden afectar al cluster Kafka. No te deja más remedio que reasignar tus particiones.

Antes de comenzar, citemos la documentación de Kafka sobre las particiones:

Más particiones permiten un mayor paralelismo para el consumo, pero esto también dará como resultado más archivos en los brokers.

Un clúster desequilibrado puede generar un consumo de disco innecesario, problemas de CPU o incluso la necesidad de agregar otro broker para manejar tráfico inesperado



# Con Kafka Tools - CLI

kafka-reassign-partitions

Este comando mueve particiones de topic entre réplicas.

Sin embargo, kafka-reassign-partitions tiene 2 fallos. No es consciente del tamaño de las particiones, y tampoco puede proporcionar un plan para reducir el número de particiones para migrar de broker a broker. Confiar ciegamente en ésta aplicación estresará el cluster para nada. Es inteligente calcular nuestro propio plan para grandes topics y aplicarlo. Moveremos por lo tanto las particiones entre brokers ‘a mano’



# Configuracion kafka tools

La herramienta utiliza dos archivos JSON para la entrada. Ambos son creados por el usuario. Los dos archivos son los siguientes:

- Topics-to-Move JSON
- Reassignment Configuration JSON



# Configuracion kafka tools

## Topics-to-Move JSON

Especifica los topics a revisar.

```
{"topics": [ {"topic": "mytopic1"},  
            {"topic": "mytopic2"}],  
    "version": 1  
}
```



# Reassignment configuration

Este archivo JSON es un archivo de configuración que contiene los parámetros utilizados en el proceso de reasignación. Este archivo es creado por el usuario, sin embargo, la herramienta genera una propuesta de su contenido. Cuando la herramienta `kafka-reassign-partitions` se ejecuta con la opción `--generate`, genera una configuración propuesta que se puede ajustar y guardar como un archivo JSON. El archivo creado de esta manera es el JSON de configuración de reasignación.

# Reassignment configuration

```
{"version":1,  
  
  "Partitions":  
  
    [{"topic":"mytopic1","partition":3,"replicas":[4,5],"log_dirs":  
      ["any","any"]},  
     {"topic":"mytopic1","partition":1,"replicas":[5,4],"log_dirs":  
      ["any","any"]},  
  
     {"topic":"mytopic2","partition":2,"replicas":[6,5],"log_dirs":  
      ["any","any"]}]  
  }
```

Property	Description
topic	Specifies the topic.
partition	Specifies the partition.
replicas	Specifies the brokers that the selected partition is assigned to. The brokers are listed in order, which means that the first broker in the list is always the leader for that partition. Change the order of brokers to resolve any leader balancing issues among brokers. Change the broker IDs to reassign partitions to different brokers.
log_dirs	Specifies the log directory of the brokers. The log directories are listed in the same order as the brokers. By default <code>any</code> is specified as the log directory, which means that the broker is free to choose where it places the replica. By default, the current broker implementation selects the log directory using a round-robin algorithm. An absolute path beginning with a / can be used to explicitly set where to store the partition replica.



# Consideraciones

Cloudera recomienda que minimice el volumen de cambios de réplica por ejecución. En lugar de mover 10 réplicas con un solo comando, mueva dos a la vez para ahorrar recursos del clúster.

Esta herramienta no se puede utilizar para crear una réplica no sincronizada en la partición principal.

Utilice esta herramienta solo cuando todos los brokers y topics estén en buen estado.

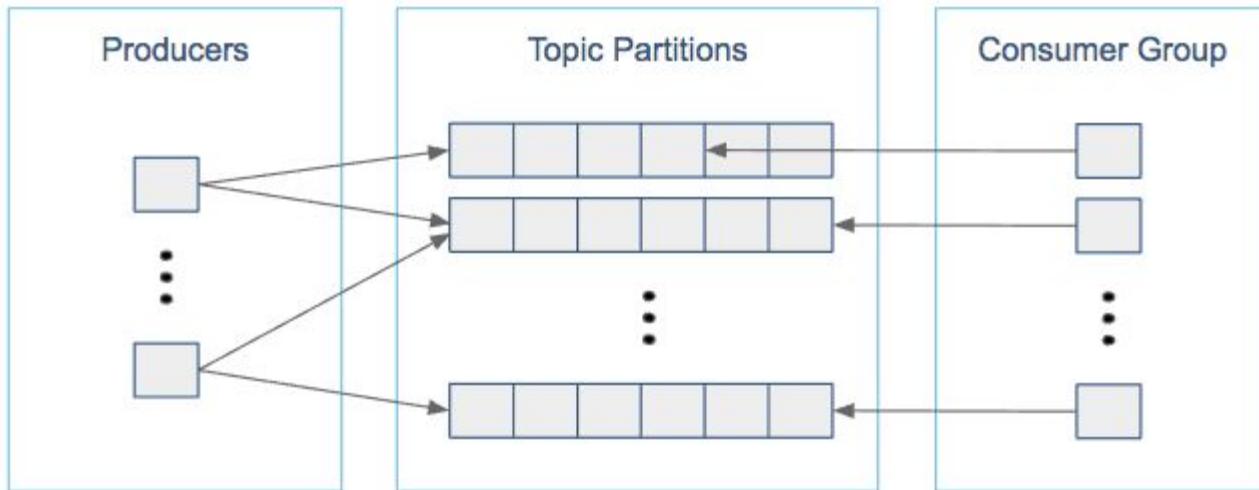
Anticipar el crecimiento del sistema. Redistribuya la carga cuando el sistema esté al 70 % de su capacidad. Esperar hasta que sea necesaria la redistribución debido a que se alcanzaron los límites de recursos puede hacer que el proceso de redistribución consuma mucho tiempo.

# **El número correcto de particiones para un topic**



# Por qué elegir bien

Elegir el número adecuado de particiones para un tema es la clave para lograr un alto grado de paralelismo con respecto a las escrituras y lecturas y la distribución de la carga. La carga distribuida uniformemente sobre las particiones es un factor clave para tener un buen rendimiento. Tomar una buena decisión requiere una estimación basada en el rendimiento deseado de productores y consumidores por partición.





## En qué fijarse

Por ejemplo, si desea poder leer 1 GB/seg, pero su consumidor solo puede procesar 50 MB/seg, necesita al menos 20 particiones y 20 consumidores en el grupo de consumidores. Del mismo modo, si desea lograr lo mismo para los productores y 1 productor sólo puede escribir a 100 MB/seg, necesita 10 particiones. En este caso, si tiene 20 particiones, puede mantener 1 GB/seg para producir y consumir mensajes. Debe ajustar la cantidad exacta de particiones a la cantidad de consumidores o productores, de modo que cada consumidor y productor logre su rendimiento objetivo.



# Formula aproximada

Productores requeridos:

Producción esperada (mb/sec)

---

Consumidores requeridos:

Producción esperada (mb/sec)

---

Capacidad productor (mb/sec)

Capacidad consumidor (mb/sec)

Elegiríamos el mayor de los dos



# Más consideraciones

- El número de particiones se puede especificar en el momento de la creación del tema o más tarde.
- La reasignación de particiones puede ser muy costosa y, por lo tanto, es mejor aprovisionar en exceso que por defecto.
- Cambiar la cantidad de particiones que se basan en claves es un desafío e implica la copia manual.
- Actualmente no se admite la reducción del número de particiones. En su lugar, cree un nuevo tema con un número menor de particiones y copie los datos existentes.
- Los metadatos sobre las particiones se almacenan en ZooKeeper en forma de znodes. Tener una gran cantidad de particiones tiene efectos en ZooKeeper y en los recursos del cliente: Las particiones innecesarias ejercen una presión adicional sobre ZooKeeper..
- Los clientes productores y consumidores necesitan más memoria, porque necesitan realizar un seguimiento de más particiones y también almacenar datos en búfer para todas las particiones.
- Como pauta para un rendimiento óptimo, no debe tener más de 4000 particiones por broker y no más de 200 000 particiones en un clúster.



# Como saber si necesitamos mas consumidores

Asegúrese de que los consumidores no se queden atrás de los productores al monitorizar el retraso del consumidor. Para verificar la posición de los consumidores en un grupo de consumidores (es decir, qué tan atrás están del final del registro), use el siguiente comando:

```
kafka-consumer-groups --bootstrap-server localhost:9092 --describe  
--group CONSUMER_GROUP
```

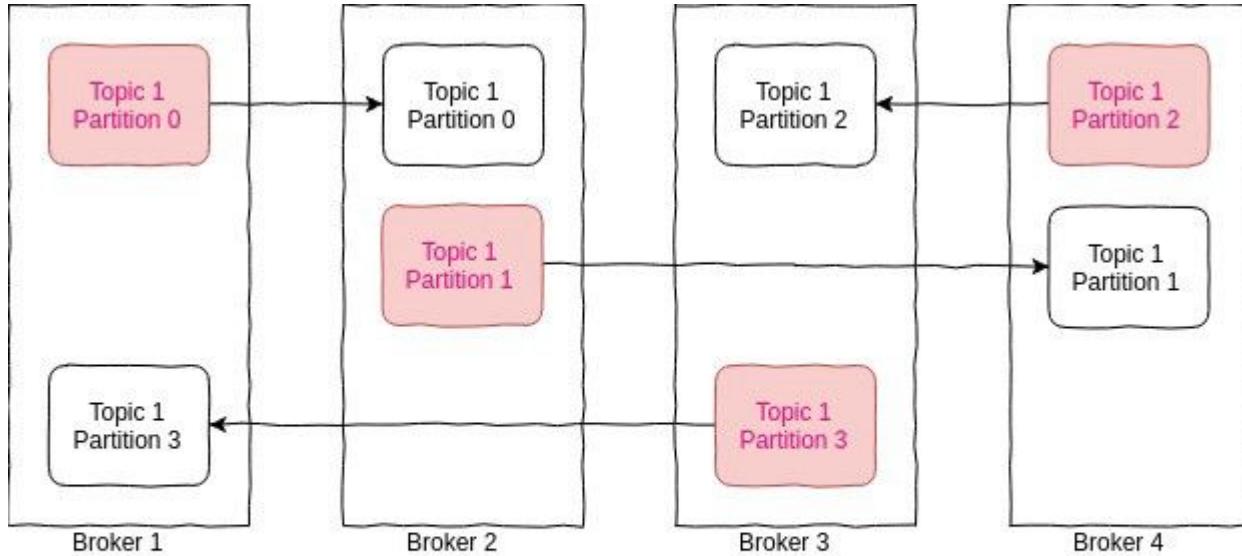
# Leader election



# Elección de leader

Cada vez que se crea un nuevo topic, Kafka ejecuta su algoritmo de elección de líder para determinar el líder preferido de una partición. La primera réplica de la lista es siempre la líder de la partición. Esto es cierto sin importar quién es el líder actual e incluso si las réplicas se reasignaron a diferentes brokers utilizando la herramienta de reasignación de réplicas. De hecho, si reasigna manualmente las réplicas, es importante recordar que la réplica que especifique primero será la réplica líder.

# Election leader



Leader

In sync replica



Todas las réplicas de los líderes se distribuyen entre los brokers y existen réplicas de las mismas en otro broker. En Kafka, todo lo que se lee y se escribe sucede a través de un líder. Por lo tanto, es importante que los líderes se distribuyan de manera uniforme entre los brokers. El algoritmo de elección de líder de Kafka se encargará de distribuir el líder de partición de manera uniforme.

A lo largo del tiempo, esto puede cambiar. Si el broker 4 está muerto, el broker 3 se convertirá en el líder de la partición 2 del topic. En ese caso, el liderazgo está sesgado.

Conduce a una distribución desigual de los líderes.



¿Qué pasa si el corredor que se prefirió para esta partición regresa? Intentará obtener la carga que se le asignó originalmente.

Pero solo puede hacerlo si todas las réplicas están sincronizadas en ese momento cuando intenta recuperar el liderazgo. En caso de que las réplicas no estén sincronizadas, la partición líder se dejará con el broker actual, el motivo será evitar la incoherencia y la pérdida de datos. Después de eso, activará el reequilibrio solo si **auto.leader.rebalance.enable** se establece en verdadero. Luego, periódicamente verificará si se prefiere o no al líder de la partición. Un punto importante a tener en cuenta aquí es que incluso el reequilibrio de líder automático se establece en verdadero, es una operación asíncrona. No garantiza que el líder se moverá de inmediato (el motivo es que las réplicas no están sincronizadas). Pero eventualmente, lo hará.



Si los líderes no se distribuyen por igual, cómo superar este estado.

Ahora somos conscientes de que los eventos pueden activarse periódicamente para reequilibrar el liderazgo y el control de la propiedad, lo mismo es `auto.leader.rebalance.enable`. Pero esta configuración no se recomienda, porque tiene implicaciones de rendimiento.

Habría que usar la herramienta de `kafka tools cli ...` con mucho cuidado

# Capacity planning and sizing



# Múltiples clusters



# Múltiples clusters

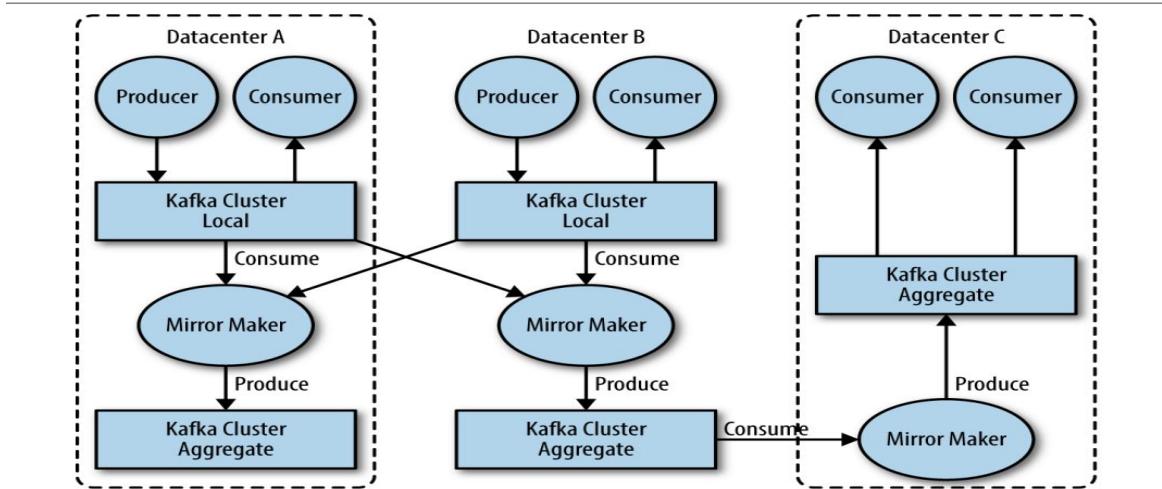
Hay varias razones para tener múltiples clusters :

- Segregar tipos de datos
- Aislamiento por requisitos de seguridad
- Múltiple datacenters (disaster recovery)



# Mirror maker

Utilidad de Kafka para sincronizar clusters a través de una cola



# Zookeeper 2



# Zookeper en kafka

Apache ZooKeeper desempeña un papel muy importante en la arquitectura de sistemas. Apache Zookeeper es un servicio de sincronización de configuración de código abierto distribuido junto con un registro de nombres para aplicaciones distribuidas.

Originalmente, el marco ZooKeeper se creó en "Yahoo!". Porque ayuda a acceder a sus aplicaciones de manera fácil. Además, para el servicio organizado utilizado por Hadoop, HBase, se convirtió en un estándar y en otros frameworks distribuidos.



# Zookeeper + Kafka

ZooKeeper almacena mucha información compartida sobre Kafka Consumers y Kafka Brokers.

- Kafka Brokers:
  - Estado (si está vivo o muerto)
  - Cuotas
  - Réplicas, selección de leader
  - Registro de nodos
- Consumers:
  - Registro

# PROYECTO

# Producer



# Generate a maven project

```
mvn archetype:generate \
```

```
  -DgroupId=com.iprocuratio.kafka \
```

```
  -DartifactId=stock \
```

```
  -DarchetypeArtifactId=maven-archetype-quickstart \
```

```
  -DinteractiveMode=false
```

Alpha Vantage Support

[Claim your API key](#)

## Support

# Alpha Vantage Support

 Claim your API Key

Claim your free API key with lifetime access. We highly recommend that you use a legitimate email address - this is the primary way we will contact you for feature announcements and troubleshooting purposes (e.g. if you lose your API key). We never send promotional or marketing materials to our users.

Which of the following best describes you?

Investor

Organization (e.g. company, university, etc.):

**ANSWER** *See page 10.*

Email:

[View Details](#)

No soy un robot 

Which of the following best describes you?

Investor

Organization (e.g. company, university, etc.):

Email:



No soy un robot



reCAPTCHA

Privacidad - Condiciones

[GET FREE API KEY](#)

Welcome to Alpha Vantage! Your dedicated access key is:  Please record this API key for future access to Alpha Vantage.

## Support

---

### Frequently Asked Questions

**I have got my API key, now what?**

Welcome to Alpha Vantage! Simply check out our [Get Started Guide](#) and you should be good to go. And you can always drop us a note at [support@alphavantage.co](mailto:support@alphavantage.co) anytime!

**Are there usage/frequency limits for the API service?**



Upgrade



# Get started with Alpha Vantage Data



Patrick Collins

Follow

Sep 13 · 3 min read



<https://www.alphavantage.co/>

I need some data...

But I'm unsure where to start,  
where do I go?

Well, thanks for asking.

## Getting started with Alpha Vantage

Alpha Vantage API is a powerful tool to get real-time stock quotes, historical data, cryptocurrencies, technical indicators, FX rates, and more. For those unfamiliar with APIs, it's a way to directly get the raw data over the internet. [Link for API information here.](#)

## Sign up

Here is the [link to sign up](#), which takes ~1 minute, and has cool fireworks!





# Particiones

# ACK's : Advanced



# ACK 0

No se solicita respuesta

Si ocurre algo, no nos enteramos

Es útil cuando no importa demasiado una pérdida potencial de mensaje:

- Recuperación de métricas
- Colección de logs

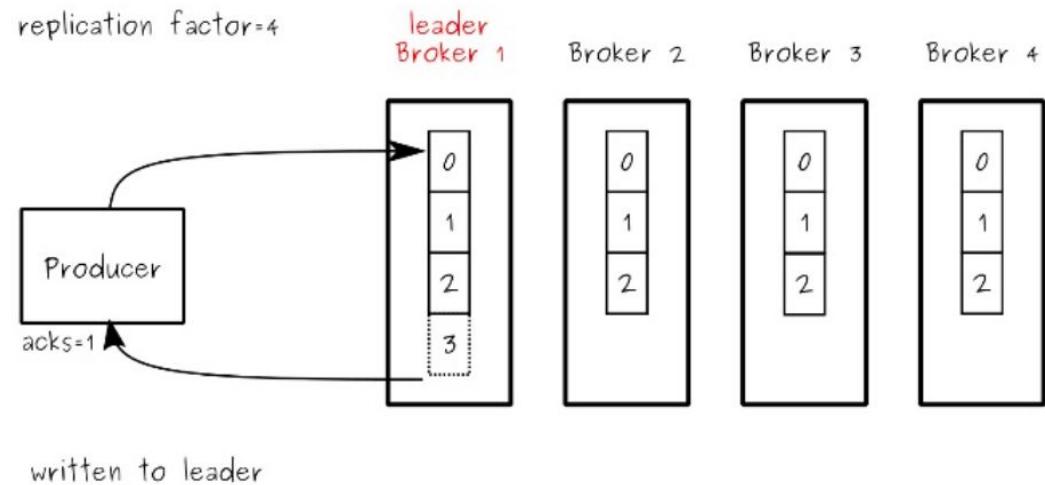


# ACK 1 (leader ack)

La replicación sucede en background

Si el ack no se recibe se puede reintentar

Si el leader cae antes de replicar  
podríamos sufrir pérdidas de datos.



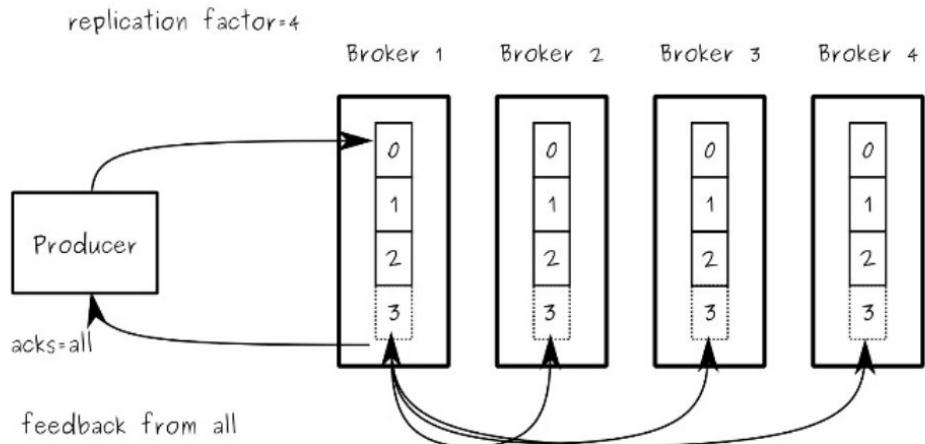
# ACK all (Replica ack)

Leader y replicas contestan

Añade seguridad y latencia

Sin pérdida de datos (si hay suficientes réplicas)

En mensajes críticos





# ACK all (Replica ack)

Ha de usarse junto con min.insync.replicas.

Puede establecerse a nivel de broker o sobreescribirlo a nivel de topic

Representa el número de réplicas que dará por bueno un acks all (incluyendo al leader)

En otro caso dará error. Esto es porque no todas las réplicas pueden estar subidas en un momento dado (mantenimiento o error, pero para eso están) y el acks all debería seguir...

# Producer: Reintentos



# Reintentos del producer

En caso de fallos temporales...

Podríamos tener:

- NotEnoughReplicasException - cuando el acks está a all y no hay min.insync.replicas respondiendo.

Hay un retries setting:

- Por defecto 0 en Kafka  $\leq 2.0$
- Por defecto max int en Kafka  $\geq 2.1$

Reintenta cada:

- retry.backoff.ms = 100ms



# Reintentos del producer

No reintenta para siempre:

`delivery.timeout.ms = 120 000ms (2min)`

Producirá un error si no hay ack en 2 minutos.

<https://cwiki.apache.org/confluence/display/KAFKA/KIP-91+Provide+Intuitive+User+Timeouts+in+The+Producer>



# Orden en los reintentos

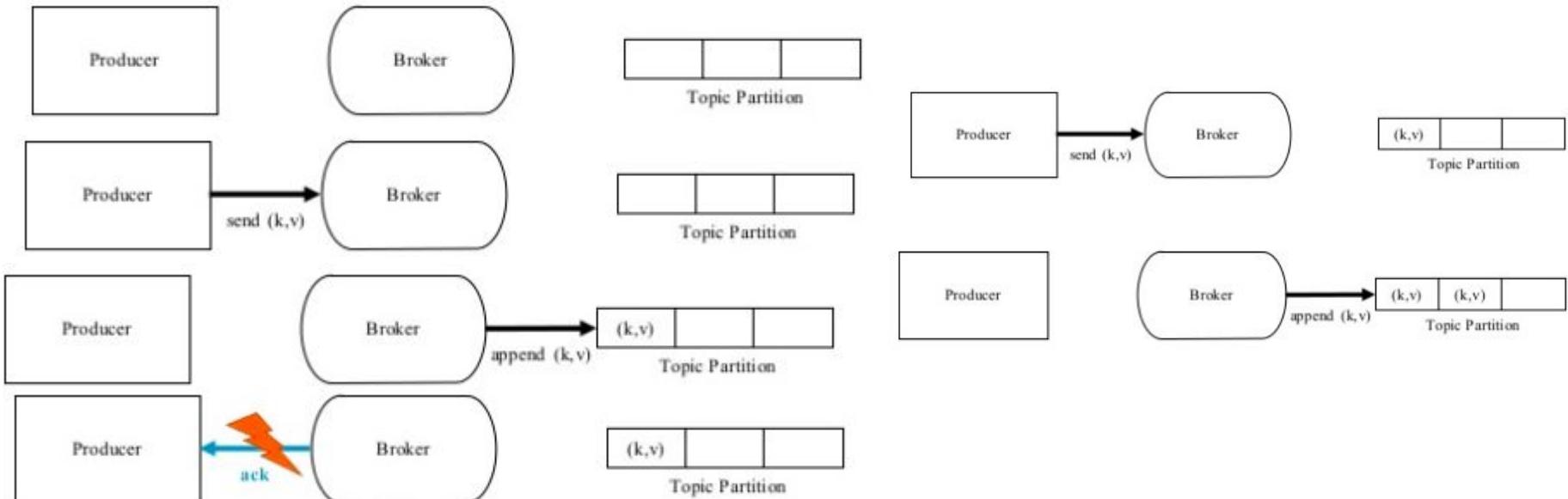
Hay una configuración: `max.in.flight.request.per.connection` es el número de envíos en paralelo a un broker. Por defecto 5, si hay retry se vuelve enviar el que ha fallado, no los demás. El siguiente batch espera.

Los reintentos pueden provocar que el orden se rompa, AUNQUE VAYAN A LA MISMA PARTICIÓN, eso sí con `max.inflight request = 5` en kafka > 1 ya no (KAFKA 5494)

Ojo: si lo dejamos a 1 dejamos a toda la cola esperando.

# Productores idempotentes

Cuando el productor envía mensajes a Kafka puedo introducir algún duplicado.





# Productores idempotentes

Hay que poner:

Enable.idempotence true

Lo que hace es enviar un version id por mensaje en los reintentos. El broker sabrá que no lo tiene que repetir.

# Message Compression



# Compresión

La compresión es muy importante a nivel de producer.

Compression.type

None, gzip, lz4, snappy, zstd

La compresión es más efectiva cuanto más grande el batch.size

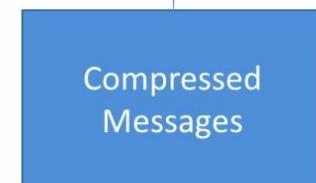
<https://blog.cloudflare.com/squeezing-the-firehose/>



Producer Batch



Compressed  
Producer Batch



Big decrease in size!

Send to Kafka





# Compresión

Ventajas:

- Mucho menor tamaño de la request
- Menor latencia
- Menor uso de disco en kafka

Desventajas:

- Tiempo de compresión en consumidores y productores
- Lz4 y snappy más rápidos (menor compresión pero no siempre) - testearlo en tus propios datos

Usar siempre en producción

Tunear linger.ms y batch.size junto con la compresión

# Producer Batching



# Batching

Kafka intenta enviar los mensajes tan pronto como puede:

- 5 flight request quiere decir 5 en paralelo
- Mientras se envía, si hay más en la cola (otro batch preparado) también lo enviará.
- Es un tema importante en rendimiento



# linger.ms y batch.size

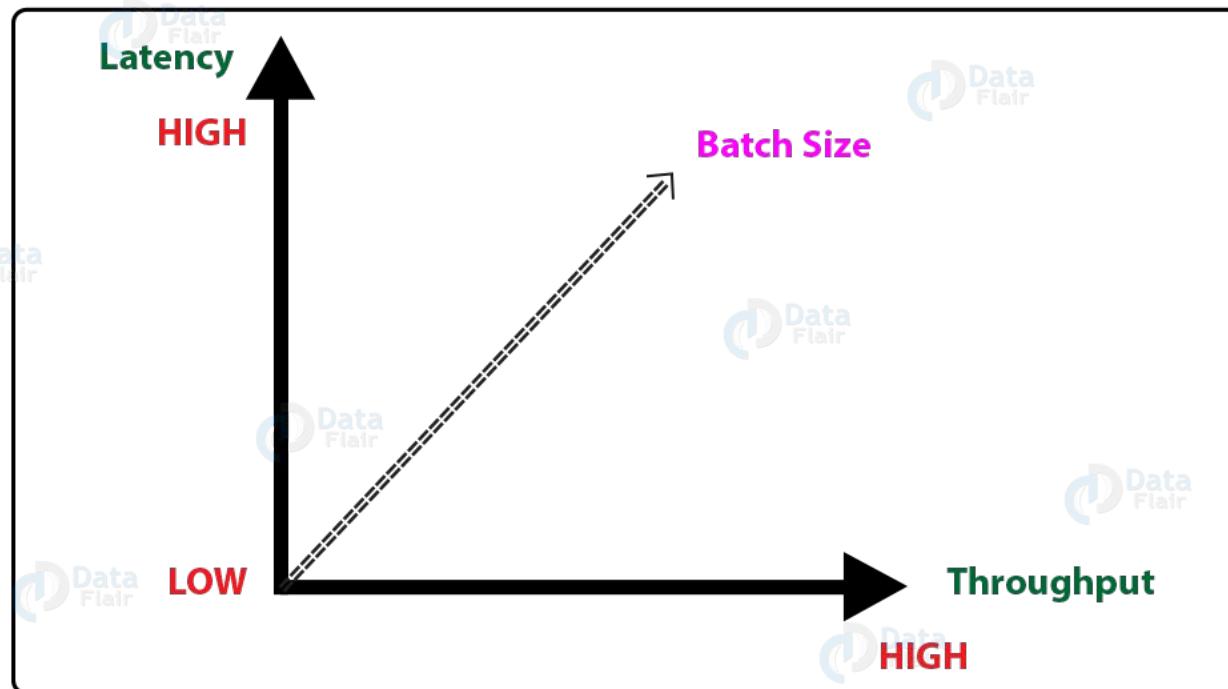
Linger.ms es el número de ms que espera para enviar un batch. Por defecto es 0, pero deberíamos incrementarlo para dar la oportunidad de enviar mensajes en paralelo.

Unos pocos milisegundos nos pueden dar mucho rendimiento.

Si un batch se llena antes de los ms, lo envía ya (batch.size)



# Kafka Performance Tuning





# batch.size

Es por defecto 16kb. Incrementarlo a 64kb por ejemplo casi siempre es buena idea.

Un batch va a la misma partición, por lo que tampoco exageremos para no desperdiciar memoria.

El batch.size siempre debería ser varias veces superior al tamaño del mensaje. Si fuese inferior no realiza el batch.



# **Max.block.ms buffer.memory**

Cuando un producer produce más rápido que lo que kafka puede absorber se guarda en la memoria de buffer.

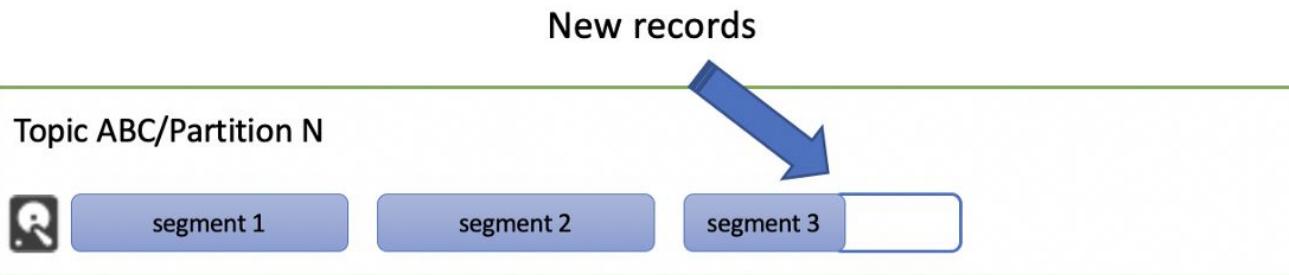
Por defecto buffer.memory = 33554432 (32mb)

Si el buffer se llena el send() empezará a bloquearse.

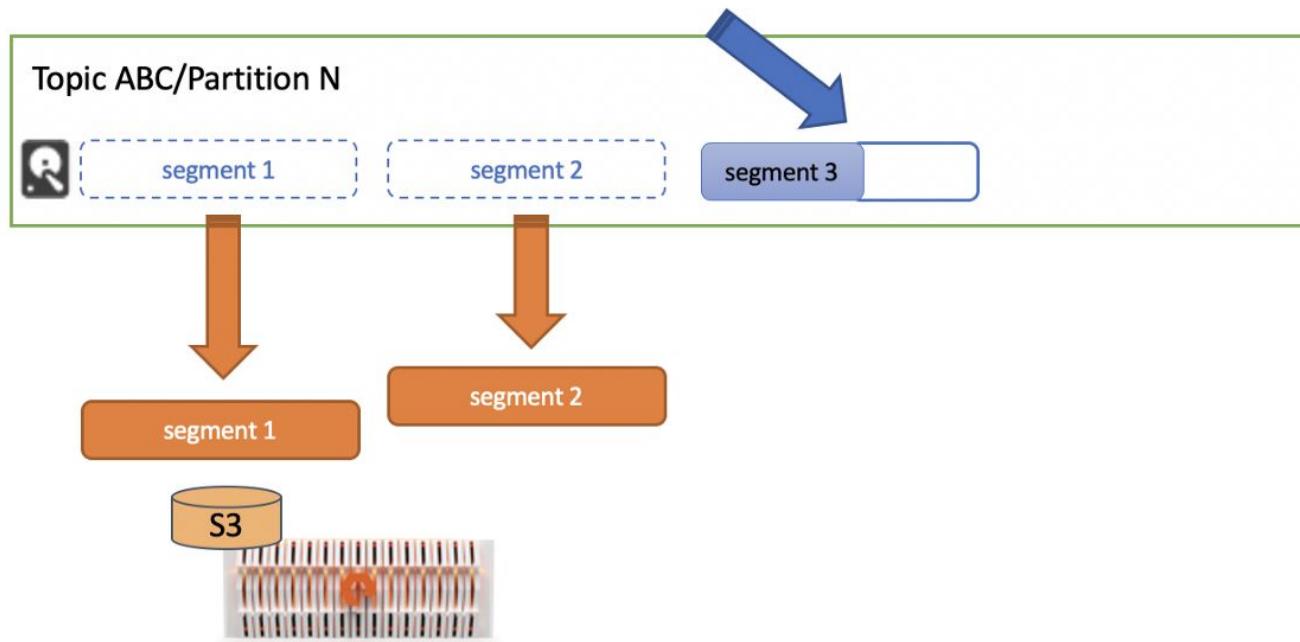
max.block.ms = 60.000 es el tiempo máximo que se bloqueará el send() antes de enviar una excepción.(estará sobrecargado)

# Tiered Storage

Classic  
Kafka



w/ Tiered  
Storage



# Consumidor Elastic Search



# Requerimientos

- Docker
- Postman



# Elastic Search

Elasticsearch es un motor de búsqueda y analítica distribuido con base en JSON.



# Instalación de Elastic Search

<https://www.elastic.co/es/>

<https://bonsai.io/>

Docker:

<https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html>

```
docker run -p 9200:9200 -p 9300:9300  
-e "discovery.type=single-node"  
--name elastic -d  
docker.elastic.co/elasticsearch/elasticsearch
```



# Kibana

“Tu ventana al Elastic Stack”

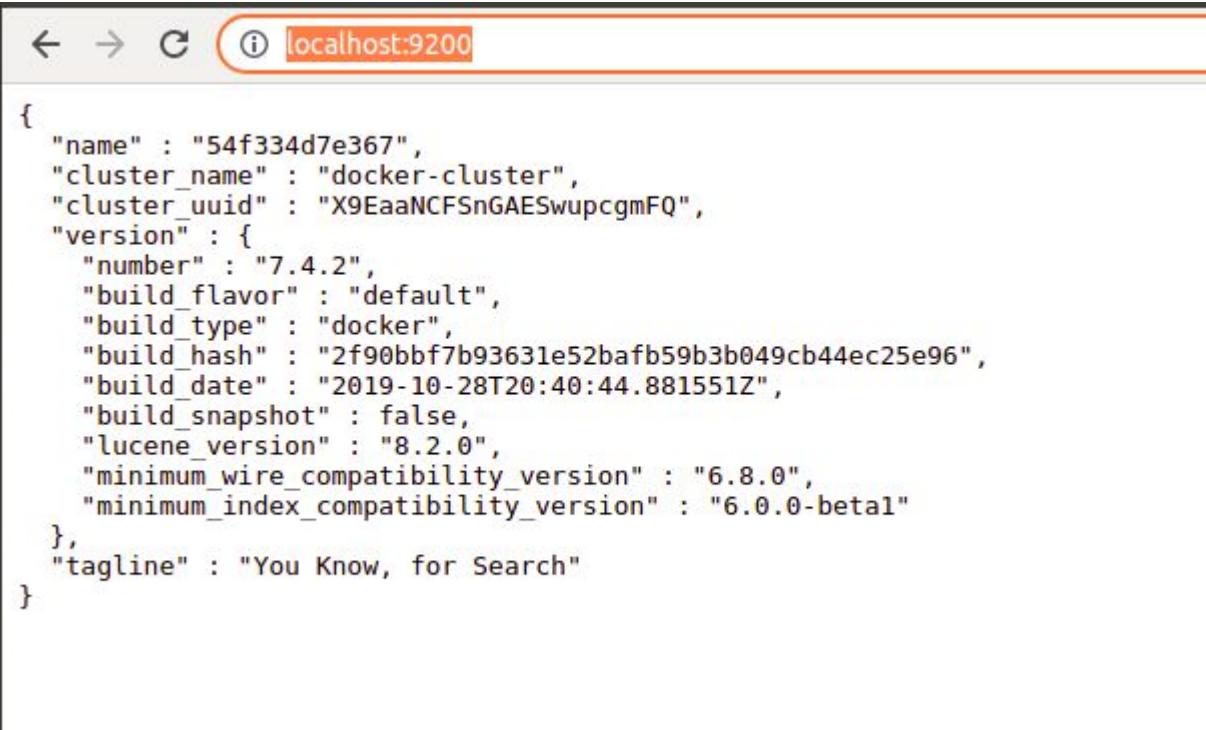
Kibana te permite visualizar tus datos de Elasticsearch y navegar por el Elastic Stack para que puedas hacer cualquier cosa, desde rastrear la carga de búsqueda hasta comprender cómo fluyen las solicitudes a través de tus aplicaciones.

# Instalación de Elastic + Kibana

```
version: '2'
services:
  elastic:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.4.2
    environment:
      - "discovery.type=single-node"
    ports:
      - 9200:9200
      - 9300:9300

  kibana:
    image: docker.elastic.co/kibana/kibana:7.4.2
    environment:
      SERVER_NAME: localhost
      ELASTICSEARCH_HOSTS: http://elastic:9200
    depends_on:
      - elastic
    ports:
      - 5601:5601
```

# Probando Elastic



A screenshot of a web browser window displaying the Elasticsearch cluster state at `localhost:9200`. The page shows a JSON response with the following data:

```
{  
  "name" : "54f334d7e367",  
  "cluster_name" : "docker-cluster",  
  "cluster_uuid" : "X9EaaNCFSnGAESwupcgmFQ",  
  "version" : {  
    "number" : "7.4.2",  
    "build_flavor" : "default",  
    "build_type" : "docker",  
    "build_hash" : "2f90bbf7b93631e52bafb59b3b049cb44ec25e96",  
    "build_date" : "2019-10-28T20:40:44.881551Z",  
    "build_snapshot" : false,  
    "lucene_version" : "8.2.0",  
    "minimum_wire_compatibility_version" : "6.8.0",  
    "minimum_index_compatibility_version" : "6.0.0-beta1"  
  },  
  "tagline" : "You Know, for Search"  
}
```



# Probando Kibana

Navegador web Firefox → localhost:5601/app/kibana#/home?\_g=()

K D Home

The sidebar features a vertical list of 15 icons, each with a corresponding color-coded square above it: a clock (grey), a magnifying glass (blue), a bar chart (green), a document (orange), a gear (yellow), a location pin (light blue), a gear with dots (teal), a folder (purple), a document with lines (pink), a gear with a checkmark (light green), a circular arrow (light orange), a lock (brown), a wrench (dark blue), a heart (red), and a gear (light grey).

Add Data to Kibana

Use these solutions to quickly turn your data into useful visualizations.

**APM**

APM automatically collects in-depth performance metrics and errors from inside your applications.

[Add APM](#)

---

**Add sample data**

Load a data set and a Kibana visualization will be generated for you.

[Add sample data](#)

---

**Visualize and Explore Data**

Explore your data with Kibana's built-in visualizations or import your own.

[Visualize and Explore Data](#)



# Elastic Health

A screenshot of a terminal window displaying the output of the Elasticsearch health endpoint at `localhost:9200/_cat/health?v`. The output shows a single node named `docker-cluster` with a status of `green`. The table includes columns for epoch, timestamp, cluster name, status, node total, node data shards, pri (primary), relo (relocation), init, unassign, pending tasks, max task wait time, active shards, and percent.

epoch	timestamp	cluster	status	node.total	node.data	shards	pri	relo	init	unassign	pending_tasks	max_task_wait_time	active_shards	percent
1574156092	09:34:52	docker-cluster	green	1	1	4	4	0	0	0	0	-	100.0%	

Vemos cosas como:

- Total de nodos
- Porcentaje de fragmentos (shards) activos. Debido a que Elasticsearch es un motor de búsqueda distribuido, un índice generalmente se divide en elementos conocidos como fragmentos que se distribuyen en varios nodos. ... réplica: de manera predeterminada, Elasticsearch crea cinco fragmentos primarios y una réplica para cada índice.



# Elastic Nodes

```
← → C ⓘ localhost:9200/_cat/nodes?v
ip      heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
172.22.0.3      34        98   3    2.26   1.56   0.98  dilm    *  6b02ca00b6f0
```



# Elastic Indices

← → ⌂ ⓘ localhost:9200/\_cat/indices?v

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
green	open	.kibana_task_manager_1	5PurSOM_T-AVrJaXPL89g	1	0	2	0	46.1kb	46.1kb
green	open	.apm-agent-configuration	tT0Y2_ZoT9ytqMPBmdMcmw	1	0	0	0	283b	283b
green	open	kibana_sample_data_logs	rDi_OnJdTUiIha5MHVowmw	1	0	14074	0	11.4mb	11.4mb
green	open	.kibana_1	bGUuVAoXR_atScFn0a36jQ	1	0	51	0	94.9kb	94.9kb



# Crear un índice

## Create an Index



Now let's create an index named "customer" and then list all the indexes again:

```
PUT /customer?pretty  
GET /_cat/indices?v
```

[COPY AS CURL](#) [VIEW IN CONSOLE](#)

The first command creates the index named "customer" using the PUT verb. We simply append pretty to the end of the call to tell it to pretty-print the JSON response (if any).

And the response:

```
health status index      uuid                                pri rep docs.count docs.delete  
yellow open   customer  95SQ4TSUT7mWBT7VNHH67A    5   1          0
```

# Creando Twitter Index

The screenshot shows the Postman application interface. At the top, there is a header bar with a logo on the left and a search bar on the right. Below the header, the main workspace displays a request configuration.

**Request Details:**

- Method:** PUT
- URL:** <http://localhost:9200/twitter>

**Params Tab (Active):**

Under the Params tab, there is a section for "Query Params". A table is shown with two rows:

KEY	VALUE
Key	Value

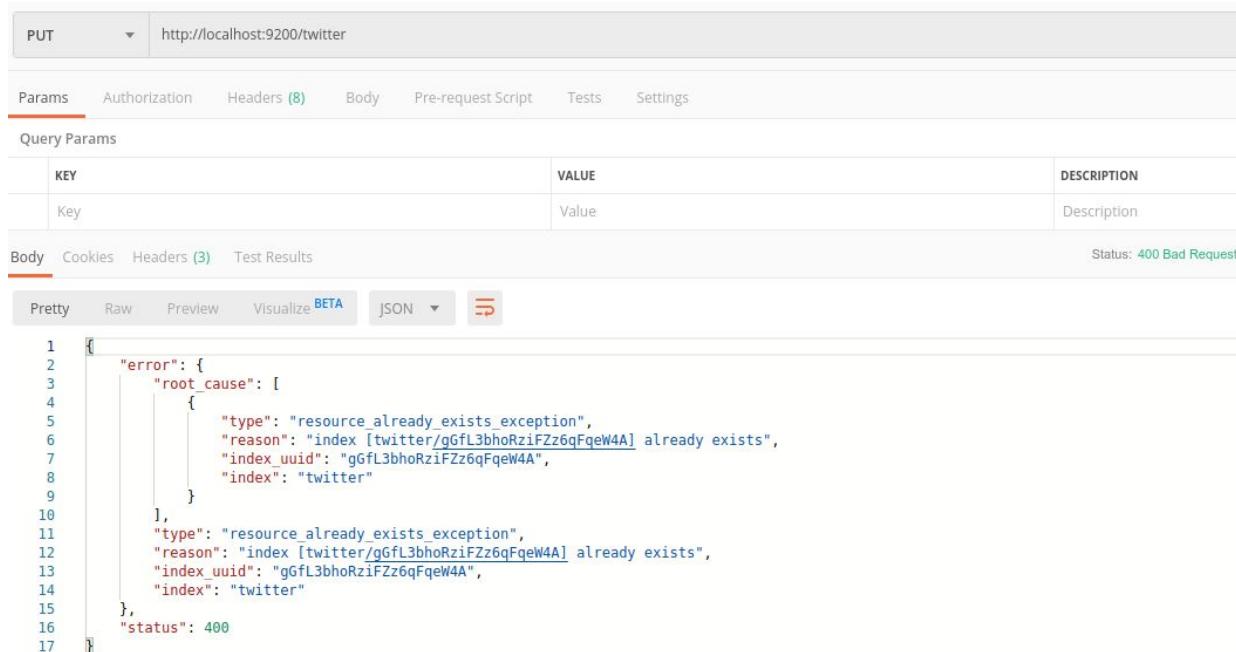
**Body Tab (Active):**

Under the Body tab, the "Pretty" option is selected. The JSON body is displayed as follows:

```
1 {  
2   "acknowledged": true,  
3   "shards_acknowledged": true,  
4   "index": "twitter"  
5 }
```

Below the JSON preview, there are buttons for "Raw", "Preview", "Visualize BETA", "JSON", and a copy icon.

# Si le doy de nuevo...



The screenshot shows a POST request in Postman to `http://localhost:9200/twitter`. The **Params** tab is selected, showing a single query parameter `Key` with value `Value`. The **Body** tab is selected, showing a JSON response with line numbers:

```
1 {  
2   "error": {  
3     "root_cause": [  
4       {  
5         "type": "resource_already_exists_exception",  
6         "reason": "index [twitter/_gGfL3bhoRziFZz6qFqeW4A] already exists",  
7         "index_uuid": "gGfL3bhoRziFZz6qFqeW4A",  
8         "index": "twitter"  
9       }  
10      ],  
11      "type": "resource_already_exists_exception",  
12      "reason": "index [twitter/_gGfL3bhoRziFZz6qFqeW4A] already exists",  
13      "index_uuid": "gGfL3bhoRziFZz6qFqeW4A",  
14      "index": "twitter"  
15    },  
16    "status": 400  
17 }
```

The status bar indicates `Status: 400 Bad Request`.

# Comprobamos el índice

```
← → ⌂ ⓘ localhost:9200/_cat/indices?v

health status index          uuid                               pri  rep docs.count docs.deleted store.size pri.store.size
yellow open  twitter         gGfL3bhoRziFZz6qFqeW4A   1    1      0           0        230b       230b
green  open  .kibana_task_manager_1  5PurSOM_T_-AVrJaXPL89g  1    0      2           0        46.1kb     46.1kb
green  open  .apm-agent-configuration tT0Y2_ZoT9ytqMPBmdMcmw  1    0      0           0        283b       283b
green  open  kibana_sample_data_logs rDi_OnJdTUiIha5MHVowmw  1    0  14074           0        11.4mb    11.4mb
green  open  .kibana_1          bGUUnVAoXR_atScFn0a36jQ   1    0      51          0        94.9kb    94.9kb
```



# Insertando datos

## Index and Query a Document



Let's now put something into our customer index. We'll index a simple customer document into the customer index, with an ID of 1 as follows:

```
PUT /customer/_doc/1?pretty
{
  "name": "John Doe"
}
```

[COPY AS CURL](#) [VIEW IN CONSOLE](#)

And the response:



# Insertando datos

PUT http://localhost:9200/twitter/tweets/1

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL BETA JSON

```
1 {  
2   "curso": "Kafka 2019",  
3   "profesor": "Alfonso Tienda",  
4   "modulo": "ElasticSearch"  
5 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize BETA JSON

```
1 {  
2   "_index": "twitter",  
3   "_type": "tweets",  
4   "_id": "1",  
5   "_version": 1,  
6   "result": "created",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 0,  
13  "_primary_term": 1  
14 }
```



# Si lo volvemos a ejecutar lo actualiza

PUT http://localhost:9200/twitter/tweets/1

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL BETA JSON

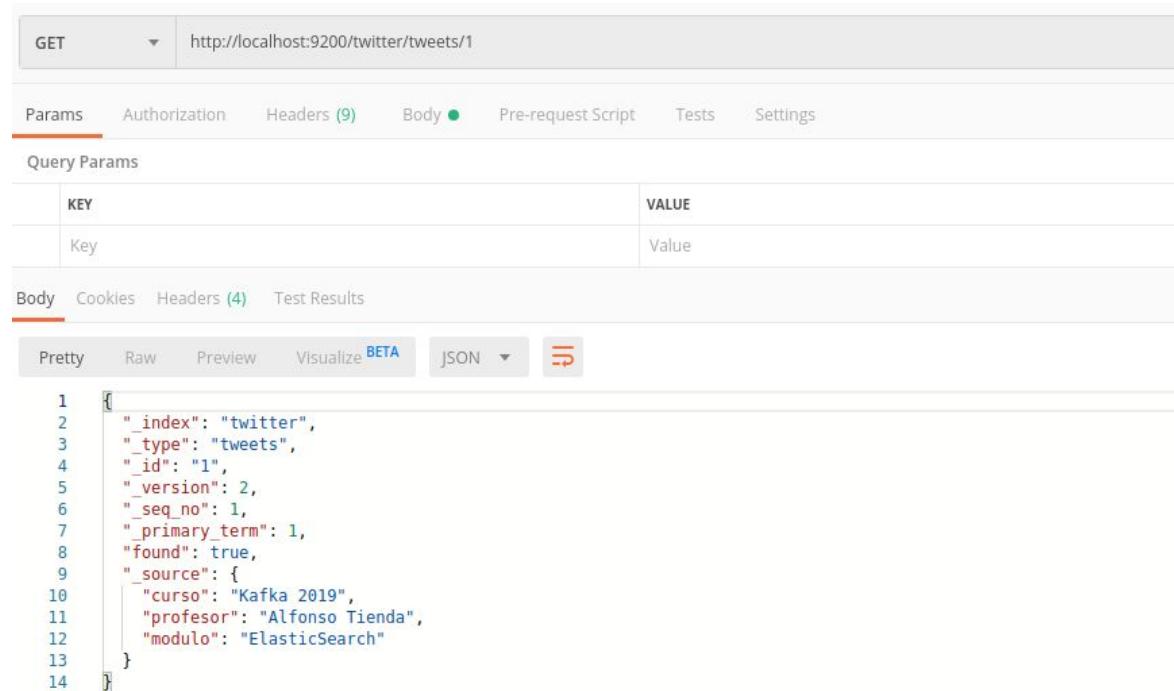
```
1 {  
2   "curso": "Kafka 2019",  
3   "profesor": "Alfonso Tienda",  
4   "modulo": "ElasticSearch"  
5 }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize BETA JSON

```
1 {  
2   "_index": "twitter",  
3   "_type": "tweets",  
4   "_id": "1",  
5   "_version": 2,  
6   "result": "updated",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 1,  
13  "_primary_term": 1  
14 }
```

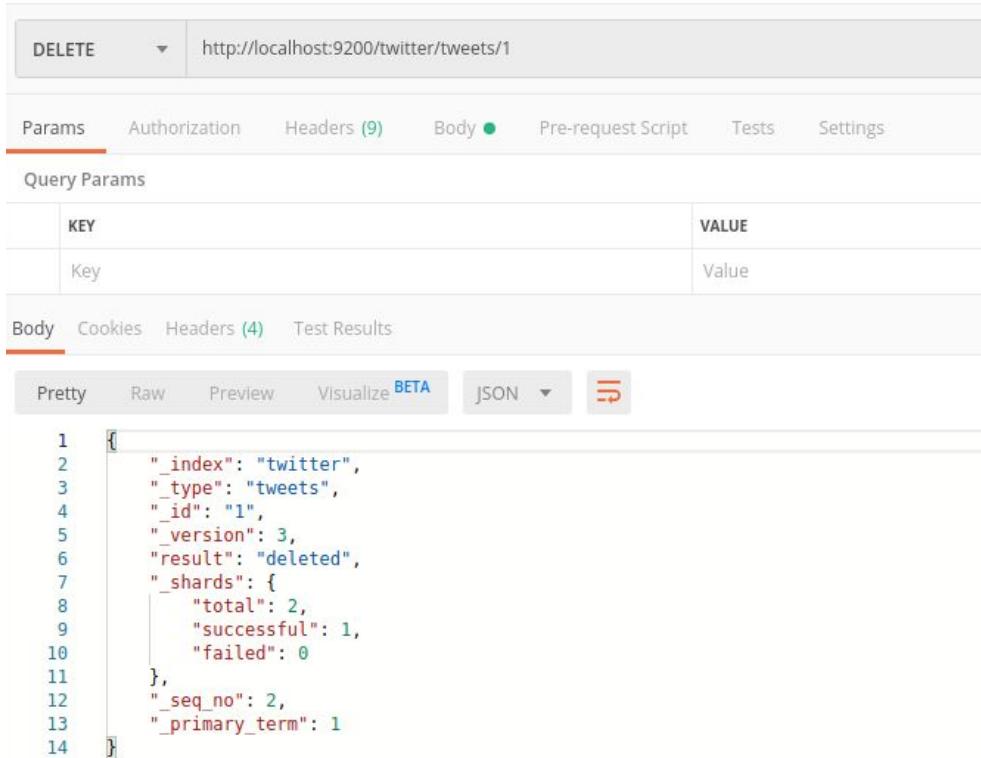
# Ver un elemento



The screenshot shows the Postman application interface. At the top, there is a header bar with a 'GET' method selected and the URL `http://localhost:9200/twitter/tweets/1`. Below the header, there are tabs for 'Params', 'Authorization', 'Headers (9)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is currently active. Under 'Query Params', there is a table with columns 'KEY' and 'VALUE', which is currently empty. Below the table, there are tabs for 'Body', 'Cookies', 'Headers (4)', and 'Test Results'. The 'Body' tab is active. Under 'Body', there are four options: 'Pretty' (selected), 'Raw', 'Preview', and 'Visualize BETA'. To the right of these options is a 'JSON' dropdown menu with a red icon. The main content area displays the JSON response in 'Pretty' format, numbered from 1 to 14. The JSON structure is as follows:

```
1 {  
2   "_index": "twitter",  
3   "_type": "tweets",  
4   "_id": "1",  
5   "_version": 2,  
6   "_seq_no": 1,  
7   "_primary_term": 1,  
8   "found": true,  
9   "_source": {  
10     "curso": "Kafka 2019",  
11     "profesor": "Alfonso Tienda",  
12     "modulo": "ElasticSearch"  
13   }  
14 }
```

# Borrar un elemento



The screenshot shows the Postman application interface. At the top, there is a header bar with a 'DELETE' button and a URL field containing 'http://localhost:9200/twitter/tweets/1'. Below the header, there are tabs for 'Params', 'Authorization', 'Headers (9)', 'Body' (which is currently selected), 'Pre-request Script', 'Tests', and 'Settings'. Under the 'Params' tab, there is a section for 'Query Params' with a table having 'KEY' and 'VALUE' columns, both of which are currently empty. Under the 'Body' tab, there are sections for 'Pretty', 'Raw', 'Preview', 'Visualize BETA', 'JSON' (with a dropdown arrow), and a copy icon. The JSON response is displayed in a code editor-like area with line numbers from 1 to 14. The JSON content is as follows:

```
1 {  
2   "_index": "twitter",  
3   "_type": "tweets",  
4   "_id": "1",  
5   "_version": 3,  
6   "result": "deleted",  
7   "_shards": {  
8     "total": 2,  
9     "successful": 1,  
10    "failed": 0  
11  },  
12  "_seq_no": 2,  
13  "_primary_term": 1  
14 }
```



# Borrar un índice

## Delete an Index



Now let's delete the index that we just created and then list all the indexes again:

```
DELETE /customer?pretty  
GET /_cat/indices?v
```

[COPY AS CURL](#) [VIEW IN CONSOLE](#)

And the response:

# Borrar un índice

The screenshot shows the Postman interface with a DELETE request to `http://localhost:9200/twitter/`. The 'Body' tab is selected, containing a JSON response with three lines of code:

```
1 {  
2   "acknowledged": true  
3 }
```

← → ⌂ ⓘ localhost:9200/\_cat/indices?v

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
green	open	.kibana_task_manager_1	5PurS0M_T_-AVrJaXPL89g	1	0	2	0	46.1kb	46.1kb
green	open	.apm-agent-configuration	tT0Y2_ZoT9ytqMPBmdMcmw	1	0	0	0	283b	283b
green	open	kibana_sample_data_logs	rDi_OnJdTUiIha5MHVowmw	1	0	14074	0	11.4mb	11.4mb
green	open	.kibana_1	bGUvVAoXR_atScFn0a36jQ	1	0	51	0	94.9kb	94.9kb



# Creando el cliente JAVA



## Docs

[Java REST Client \[7.4\]](#) » [Java High Level REST Client](#) » [Getting started](#) » [Maven Repository](#)

« [Javadoc](#)

[Dependencies](#) »

## Maven Repository

The high-level Java REST client is hosted on [Maven Central](#). The minimum Java version required is 1.8.

The High Level REST Client is subject to the same release cycle as Elasticsearch. Replace the version with the desired client version.

If you are looking for a SNAPSHOT version, the Elastic Maven Snapshot repository is available at <https://snapshots.elastic.co/maven/>.

### Maven configuration

Here is how you can configure the dependency using maven as a dependency manager. Add the following to your pom.xml file:

```
<dependency>
  <groupId>org.elasticsearch.client</groupId>
  <artifactId>elasticsearch-rest-high-level-client</artifactId>
  <version>7.4.2</version>
</dependency>
```

### Gradle configuration

Here is how you can configure the dependency using gradle as a dependency manager. Add the following to your build.gradle file:

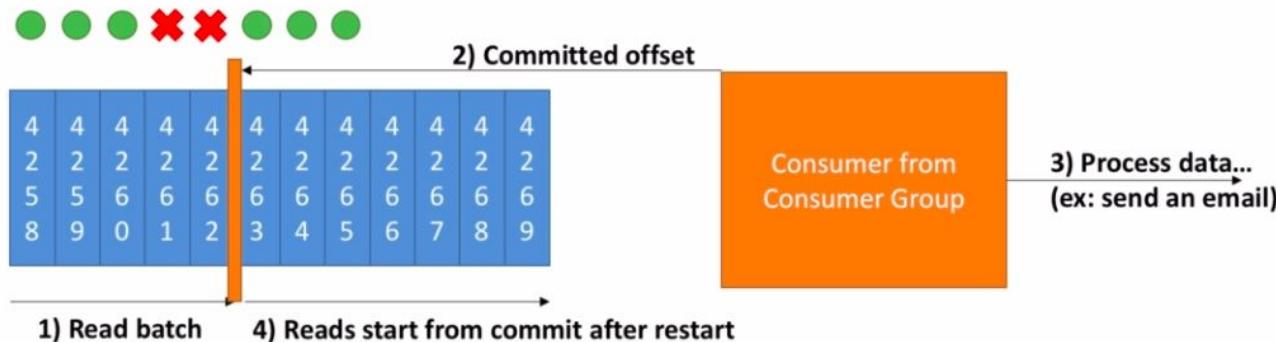
```
dependencies {
  compile 'org.elasticsearch.client:elasticsearch-rest-high-level-client'
}
```

# Delivery Semantics



# At most once

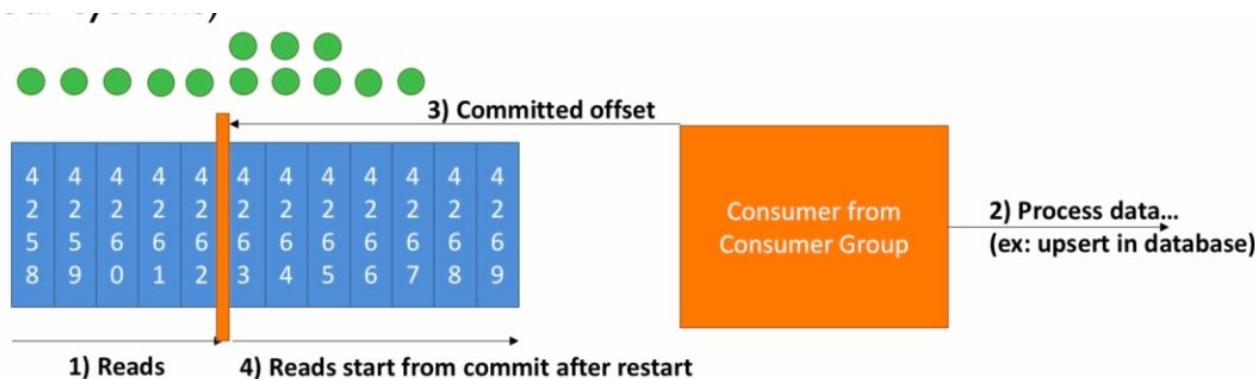
Los offsets se comitean tan pronto como el batch de mensajes se recibe. Si el procesamiento falla el mensaje no se repite.





# At least once

Los offsets se comitean al final, tras realizar la acción. Si no son idempotentes se podría dar el caso de repetir el proceso.





# Exactly once

Exactamente una vez.

- AT LEAST ONCE
- Operación idempotente

Según dice kafka al juntar dos streams en diferentes instalaciones, también



## En resumen

Para la mayoría de aplicaciones es mejor la estrategia de at least once, con procesos idempotentes.

# Consumer offset commit strategies



# Estrategias

enable.auto.commit a true y procesamiento síncrono de mensajes

```
while(true){  
    List<Records> batch = consumer.poll(Duration.ofMillis(100))  
    doSomethingSynchronous(batch)  
}
```

Con el auto-commit los offsets se comitean en auto.commit.interval.ms = 5000 ms por defecto.

Si el procesamiento no es síncrono estaremos en at-most-once



# Estrategias

Enable.auto.commit en false y procesamiento síncrono

Almacenamos el batch

```
while(true){  
    batch += consumer.poll(Duration.ofMillis(100))  
    if isReady(batch) {  
        doSomethingSynchronous(batch)  
        consumer.commitSync();  
    }  
}
```

# Instalacion del Kafka Cluster



# Kafka setup

Necesitarás múltiples brokers ( $2n+1$ ) en diferentes racks (o localizaciones físicas diferentes, incluidos discos) y un cluster (idealmente separado) de por lo menos 3 zookeepers. Incluso en Amazon idealmente estará en regiones diferentes.

Tanto zookeeper como kafka deberían estar aislados.

Alternativa: Kafka as a service



# Kafka Metrics

Las métricas de kafka están expuestas mediante JMX

[https://es.wikipedia.org/wiki/Java\\_Management\\_Extensions](https://es.wikipedia.org/wiki/Java_Management_Extensions)

Donde almacenar estas métricas:

- Elastic Search + Kibana
- Datadog
- New Relic
- Confluent Control Center
- Prometheus , Grafana



# Kafka Metrics

Métricas fundamentales:

- Particiones a las que les faltan réplicas. Todas aquellas que les faltan ISR. Pueden suponer sobrecarga del sistema
- Request Handlers: Utilización de hilos de E/S, red... utilización completa del Kafka Broker
- Latencia de respuesta

<https://kafka.apache.org/documentation/#monitoring>



# Operaciones usuales

- Rolling restart de los brokers
- Actualización de las configuraciones
- Rebalanceo de particiones
- Incrementar el factor de réplica
- Añadir o eliminar un broker
- Actualizar kafka

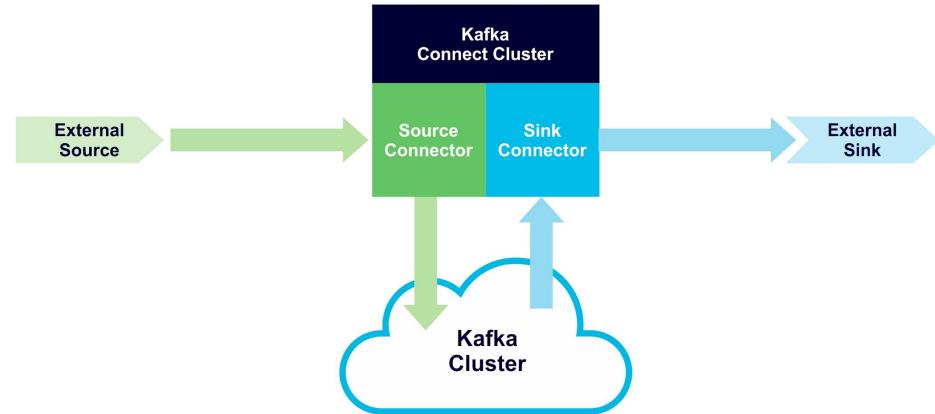
# Kafka Connect



# Kafka Connect

Es para el uso de conectores  
de entrada y salida

<https://www.confluent.io/hub/>



# Schema Registry



# Schema Registry

Kafka recoge bytes de la entrada y los publica

No valida los datos

¿Qué ocurre si...

- Un productor emite datos incorrectos?
- Renombran un campo?
- El formato de datos cambia?

**¡El consumidor no funciona!**



# Schema Registry

Necesitamos datos que sean auto-descriptivos

Debemos evolucionar los consumidores de forma que los consumidores sigan siendo compatibles.

Para esto usamos esquemas y el Schema Registry, ya que Kafka es TAN bueno porque no parsea datos y ni siquiera lee los datos. En eso se basa su velocidad. No debemos ocupar a los brokers con esto. De hecho, utiliza el patrón copia cero (zero-copy), mediante el cual el rendimiento se ve mejorado porque permite a la CPU realizar otras tareas mientras la copia de datos se realiza en paralelo en otra parte de la máquina

Emite los byte array exactamente como los recibe.



# Schema Registry

Kafka por lo tanto no es consciente del contenido del mensaje ni siquiera de su tipo (String, Integer...)

Por las razones anteriores el schema registry ha de ser un componente separado y se ha de permitir a productores y consumidores interactuar con él.

El Confluent Schema Registry (Con los serializadores de Avro) hace esta función.



# Nuestra pipeline sin Schema Registry

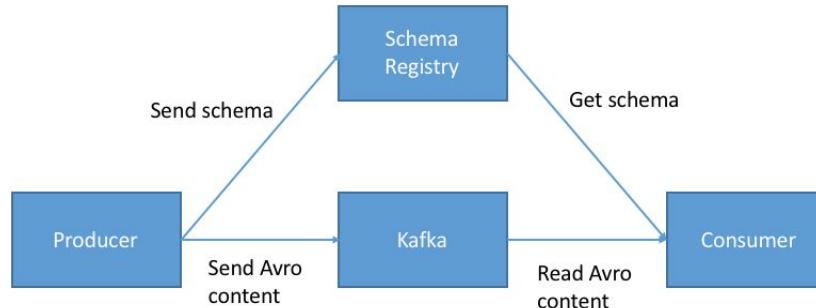




# Schema Registry

El schema registry:

- Almacena y recupera esquemas para los productores y consumidores.
- Asegurar la compatibilidad forward, backward en los topics
- Decrementar el payload enviado a kafka.



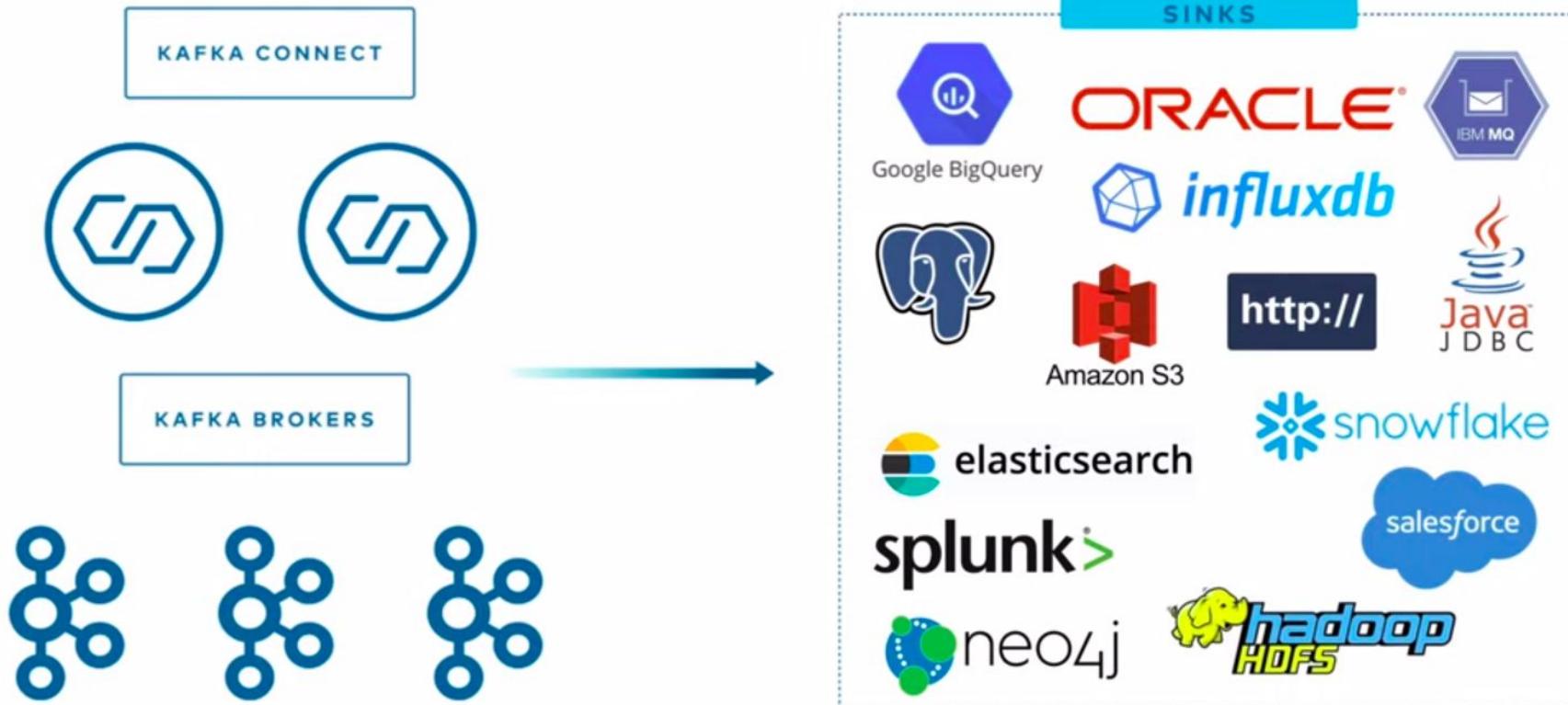


# Contras

- Es complejo de instalar correctamente
- Necesita estar en alta disponibilidad, ya que ahora dependemos también de esta pieza
- Hay que modificar parcialmente el código del productor y el consumidor
- Apache Avro... no es sencillo

# Kafka Connect







# Sistema declarativo

No es necesario escribir código para  
Kafka Connect

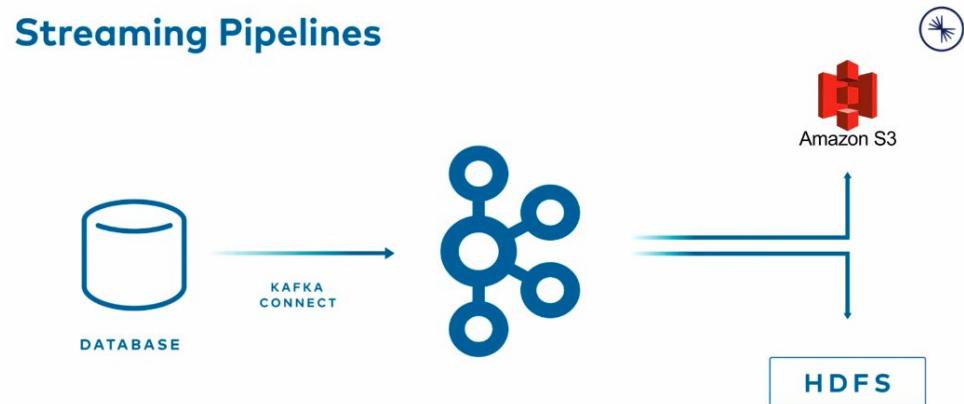
```
{  
  "connector.class":  
    "io.confluent.connect.jdbc.JdbcSourceConnector",  
  
  "connection.url":  
    "jdbc:mysql://asgard:3306/demo",  
  
  "table.whitelist":  
    "sales,orders,customers"  
}
```



# Casos de uso

- Data Buffering
- Desacoplamiento
- Tolerancia a fallos (en los dos puntos)

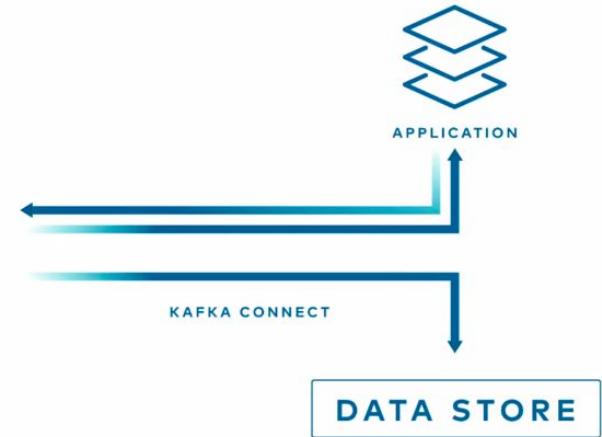
## Streaming Pipelines





# Casos de uso

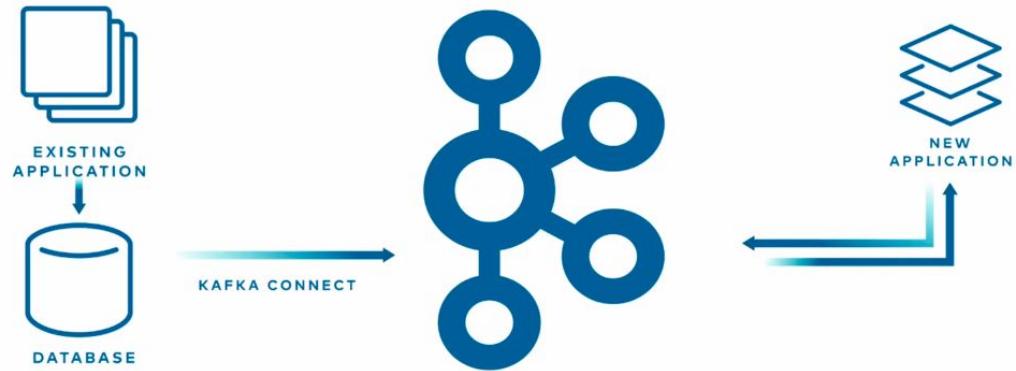
- Data storing (se pueden seguir guardando datos en la base de datos sea cual sea la velocidad real de escritura del store)





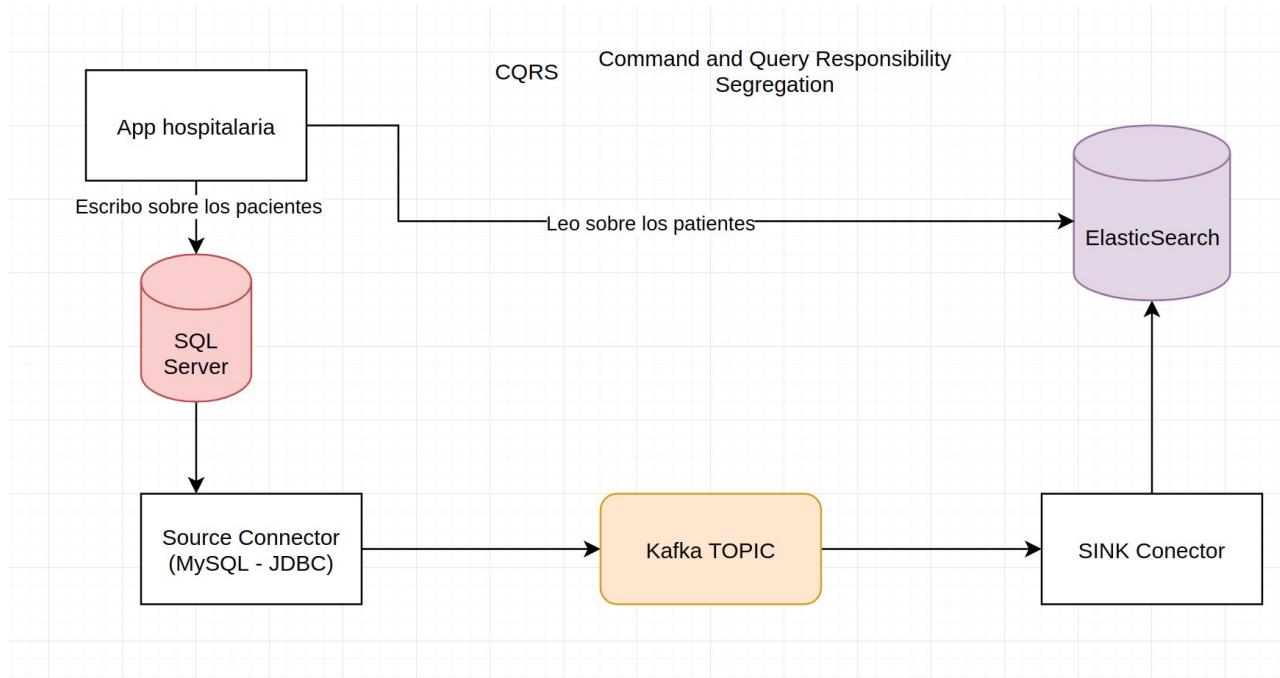
# Casos de uso

- Conexión con aplicaciones legadas





# Ejemplo: CQRS en Elastic desde BD Relacional

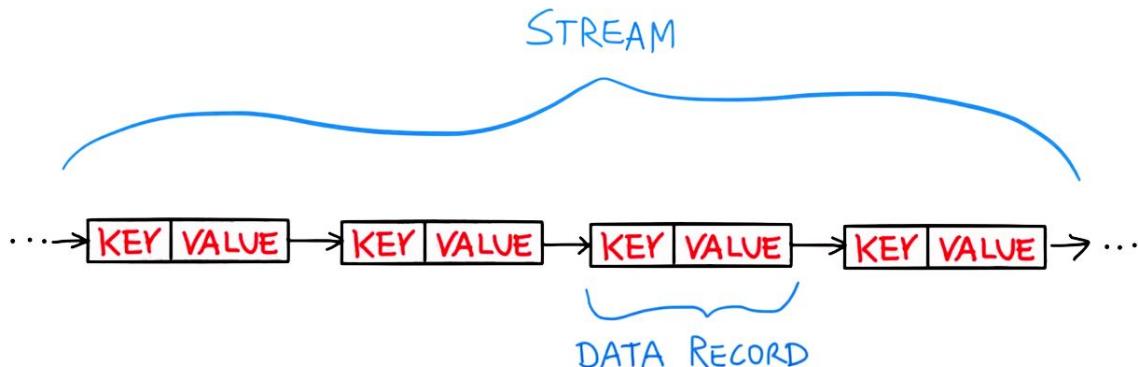


# Kafka Streams



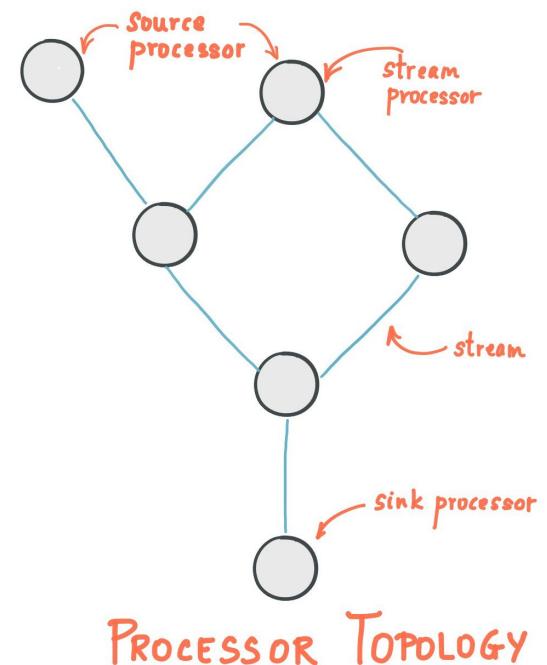
# ¿Qué es un stream?

- Un stream es un flujo de registros continuo e infinito
  - No hay que solicitar datos nuevos, simplemente los recibes
- En el caso de Kafka, los registros son pares de clave-valor



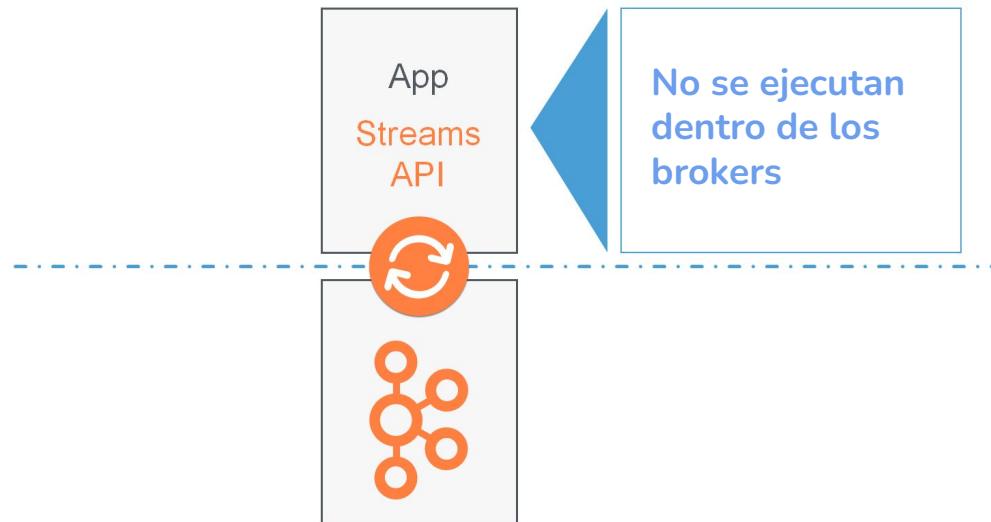
# Kafka Streams API

- Kafka Streams API transforma y enriquece los datos
  - Soporta procesamiento por registro con una latencia de microsegundos
  - Soporta procesamiento stateless, stateful y operaciones en intervalos de tiempo (windowing)
- Se escriben aplicaciones Java estándar para procesar los datos en tiempo real
  - No se requiere un procesamiento separado
  - Tolerante a fallos
  - Cuando creas una aplicación de streams estas de hecho creando un sistema distribuido, sin embargo la API de kafka resuelve los problemas asociados a este tipo de sistemas.
- Soporta la semántica exactly once
- El trabajo de los streams se realiza en la aplicación, conectada al broker, no en los brokers en si.



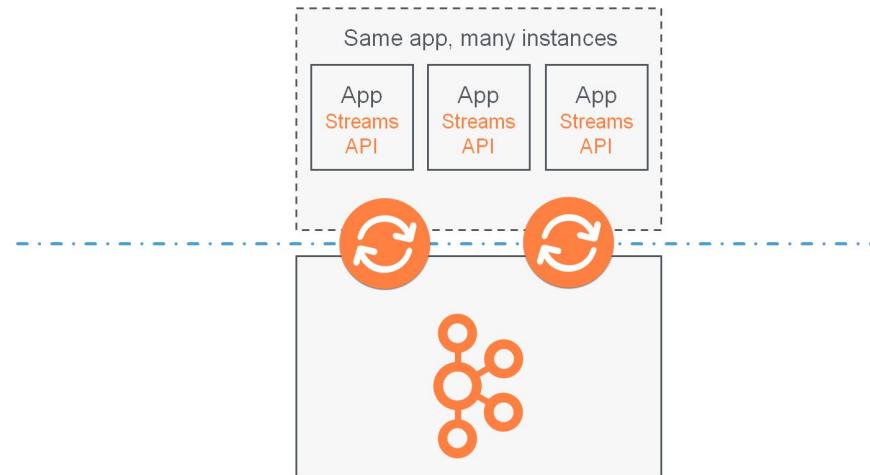


# Streams Apps





# Streams Apps





# SERDES

- En streams se usar SERDES (Serializers - Deserializers) para serializar y deserializar mensajes en ByteArray
- Los de clave pueden ser diferentes que los de valor



# Crear Streams

- Creamos los streams desde topics de Kafka:

```
StreamsBuilder builder = new StreamsBuilder();
// KStream es la abstracción de un stream , esto es del flujo infinito de datos estructurados
KStream<String, String> source = builder.stream( topic: "names");
```

- Si sólo quiero un log de cambios y el último valor de una key, usamos KTable, serían “hechos” que evolucionan



# Abstracciones en la API

- **KStream:** es un flujo de pares clave-valor, un modelo similar al utilizado para un topic de Kafka. Los registros en un KStream provienen directamente de un topic o han pasado por algún tipo de transformación; por ejemplo, hay un método de filtro que toma un predicado y devuelve otro KStream que solo contiene aquellos elementos que satisfacen el predicado.
- **KTable:** Los KStreams no tienen estado, pero permiten la agregación convirtiéndolos en la otra abstracción principal: una KTable, que a menudo se describe como un "flujo de registro de cambios". Una KTable contiene el último valor para una clave de mensaje determinada y reacciona automáticamente a los nuevos mensajes entrantes. Un caso especial es la GlobalKTable, que guarda todos los datos en todas las particiones





# KStreams KTable

**Streams**  
record history

- 1. e4 e5
- 2. Nf3 Nc6
- 3. Bc4 Bc5
- 4. d3 Nf6
- 5. Nbd2



“The sequence of moves”

**Tables**  
represent state

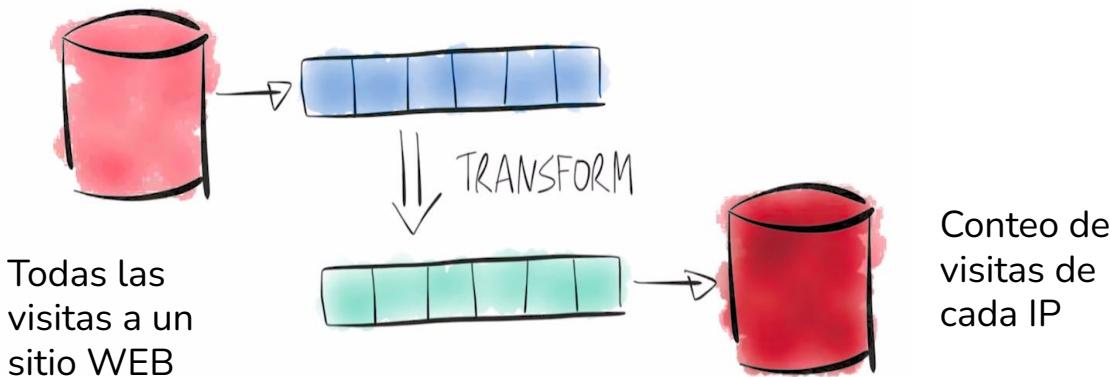


“The state of the board”



# Transformaciones

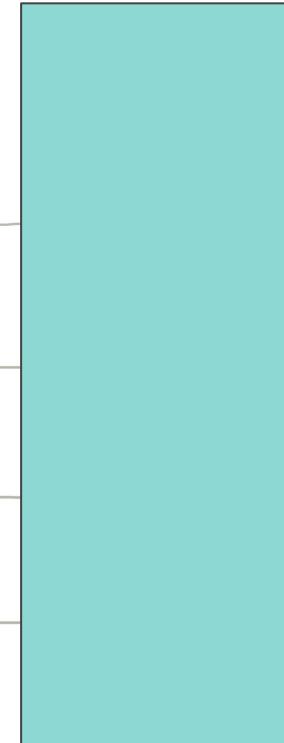
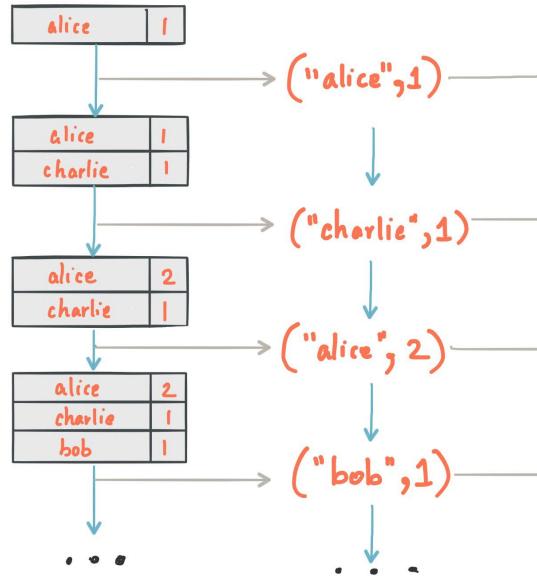
STREAM + TABLE



# Dualidad KStreams - KTables

Topic “ventas”

TABLE                    STREAM (as changelog)





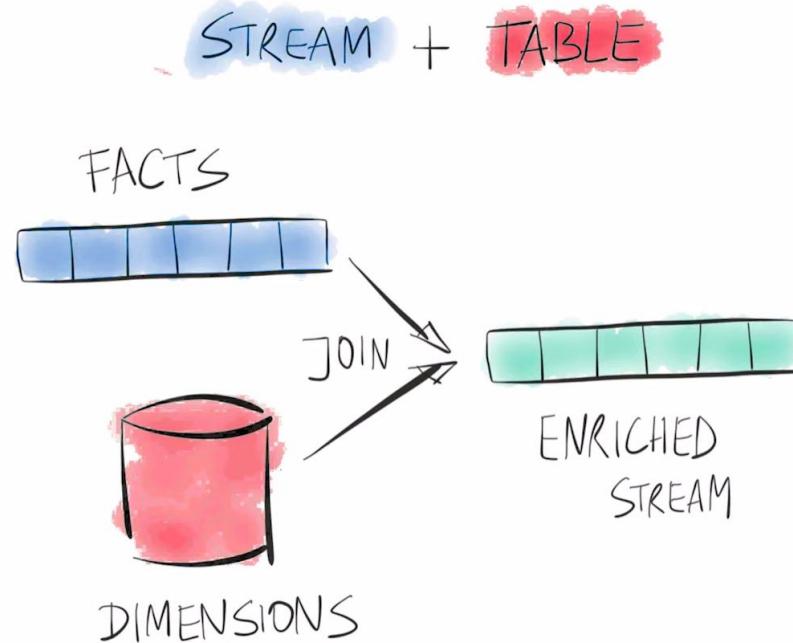
# Filter and map

- Es una transformación stateless - Creamos un nuevo stream
- Filter elimina elementos (filtra)
- Map transforma elementos

```
// KStream es la abstracción de un stream , esto es del flujo infinito de datos estructurados
KStream<String, String> source = builder.stream( topic: "names");
source.filter((key, name) -> name.toString().toLowerCase(Locale.ROOT).startsWith("a"))
    .mapValues(value -> "holo "+value).to( s: "hello");
```

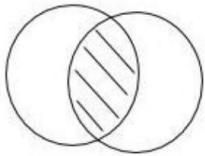


# Merging Data

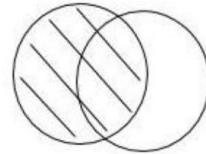




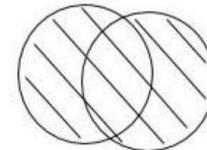
# Joins



inner:  
*KStream-KStream*  
*KTable-KTable*  
*KStream-KTable*  
*KStream-GlobalKTable*



left:  
*KStream-KStream*  
*KTable-KTable*  
*KStream-KTable*  
*KStream-GlobalKTable*



outer:  
*KStream-KStream*  
*KTable-KTable*

<https://www.confluent.io/blog/crossing-streams-joins-apache-kafka/>



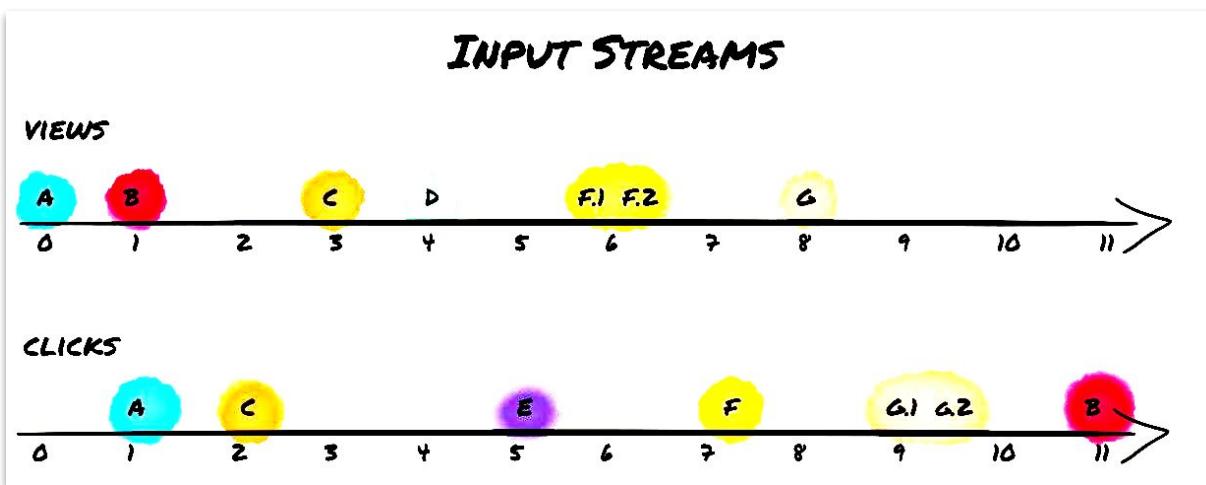
# Join: Stream - Stream

Todas las uniones KStream-KStream tienen una ventana temporal, por lo que el desarrollador tiene que especificar cuánto tiempo debe ser esa ventana y si el orden relativo de los elementos de ambos flujos es importante

Un KStream no tiene estado. Para ejecutar una combinación con un rendimiento aceptable, es necesario mantener algún estado interno. Ese estado contiene todos los elementos de la secuencia dentro de la ventana de tiempo. En segundo lugar, semánticamente hablando, unir dos flujos produce resultados "interesantes" si ambos registros de cada flujo se sincronizan cerca el uno del otro (es decir, ambos eventos ocurrieron dentro de un cierto período de tiempo). Por ejemplo, se mostró un anuncio en una página web y un usuario hace clic en él en 10 segundos.

Una combinación interna en dos streams produce un resultado si aparece una clave en ambos streams dentro de la ventana. Aplicado al ejemplo, esto produce los siguientes resultados:

# Joins: ejemplo





# Inner Join Stream-Stream

## INNER STREAM-STREAM JOIN

VIEWS



CLICKS

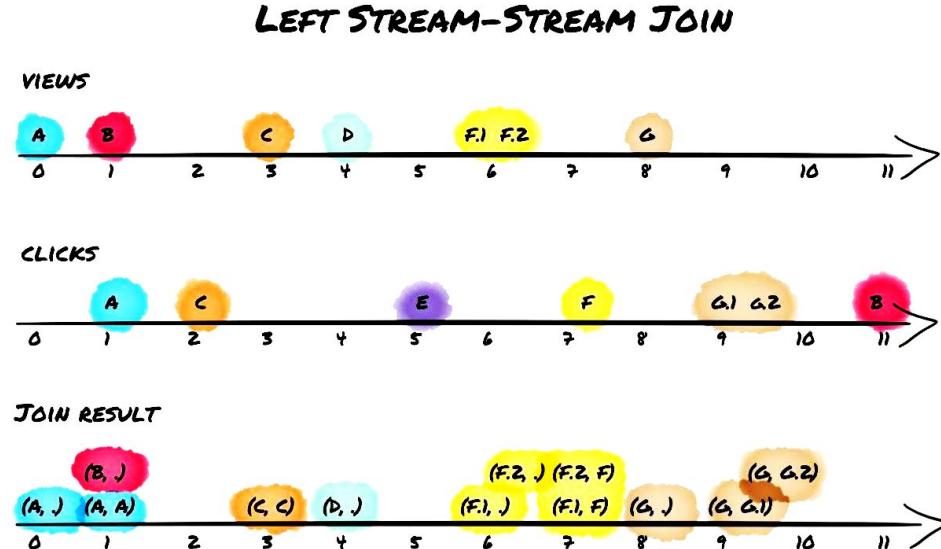


JOIN RESULT



Nuestra ventana es de 10 segundos

# Left Join Stream-Stream



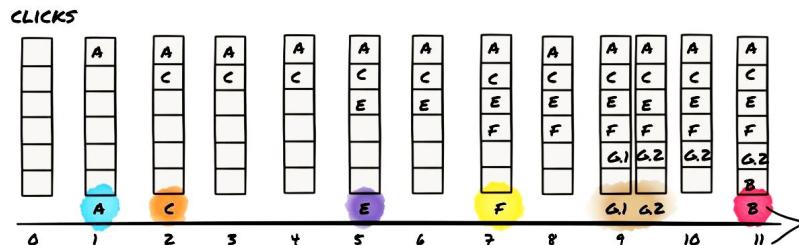
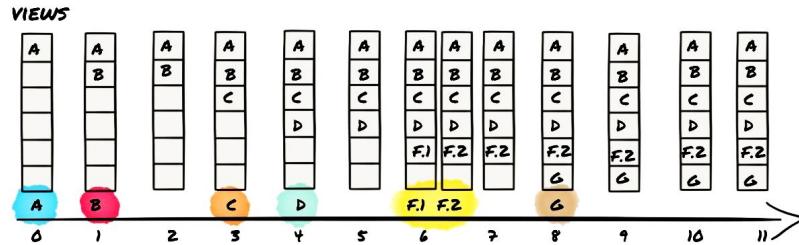
Nuestra ventana es de 10 segundos

# Join KTable - Ktable

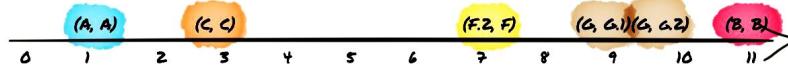
No tenemos ventana temporal

Hay que recordar que las tablas  
almacenan sólo el último  
elemento de cada key

INNER TABLE-TABLE JOIN



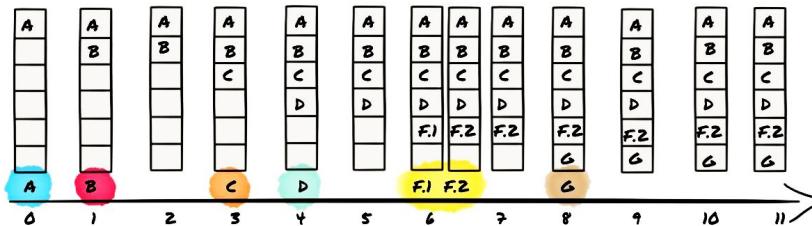
JOIN RESULT



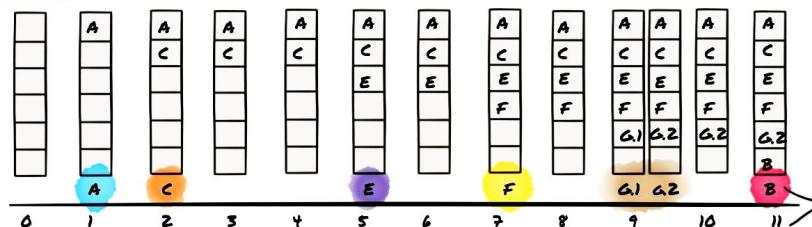
# Join KTable - Ktable

LEFT TABLE-TABLE JOIN

IEWS



CLICKS



JOIN RESULT





# KStream - KTable

Los eventos entrantes en un stream se pueden unir contra una table. Similar a una unión tabla-tabla, esta unión no requiere ventana; sin embargo, el resultado de esta operación es otro flujo y no una tabla.

Para las uniones asimétricas (stream vs table), sólo la entrada de stream desencadena un cálculo de unión, mientras que los registros de entrada de tabla solo actualizan la tabla materializada.



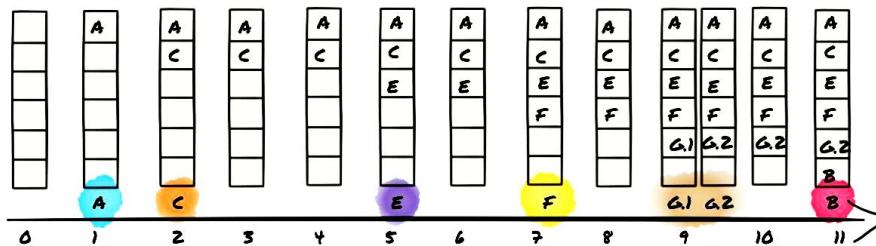
# KStream - KTable

## INNER STREAM-TABLE JOIN

IEWS



CLICKS



JOIN RESULT





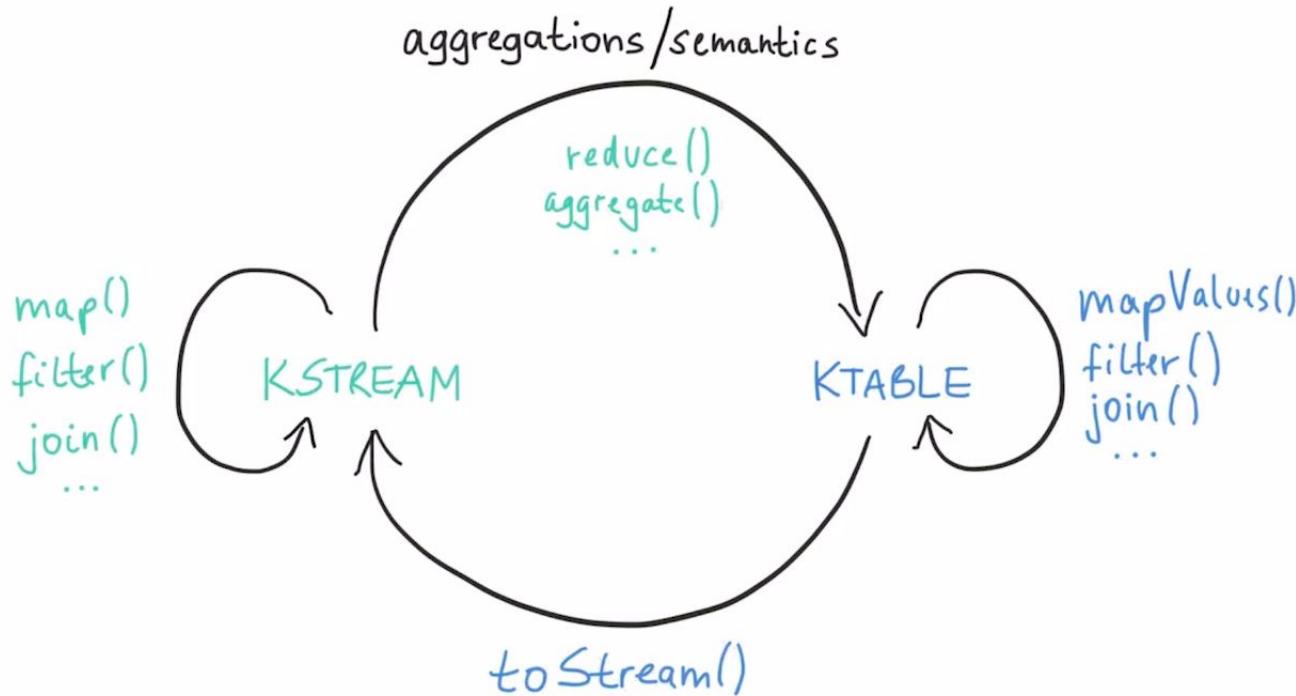
# Inner KTable - KTable

Una KTable es un flujo de registro de cambios de actualizaciones; por lo tanto, una KTable "simple" es una secuencia sin estado con una semántica diferente a la de un KStream. Sin embargo, a menudo las KTables también se materializan en una "estado local", creando una tabla que siempre contiene el último valor de una clave. Si se unen dos KTables, siempre se materializan. Esto permite buscar registros de unión coincidentes.

Las uniones en KTables no tienen ventanas y su resultado es una vista en constante actualización del resultado de la combinación de ambas tablas de entrada. Si se actualiza una tabla de entrada, la KTable resultante también se actualiza en consecuencia; tenga en cuenta que esta actualización de la tabla de resultados es solo un nuevo registro de salida, porque la KTable resultante no se materializa de forma predeterminada.



# Stateful processing



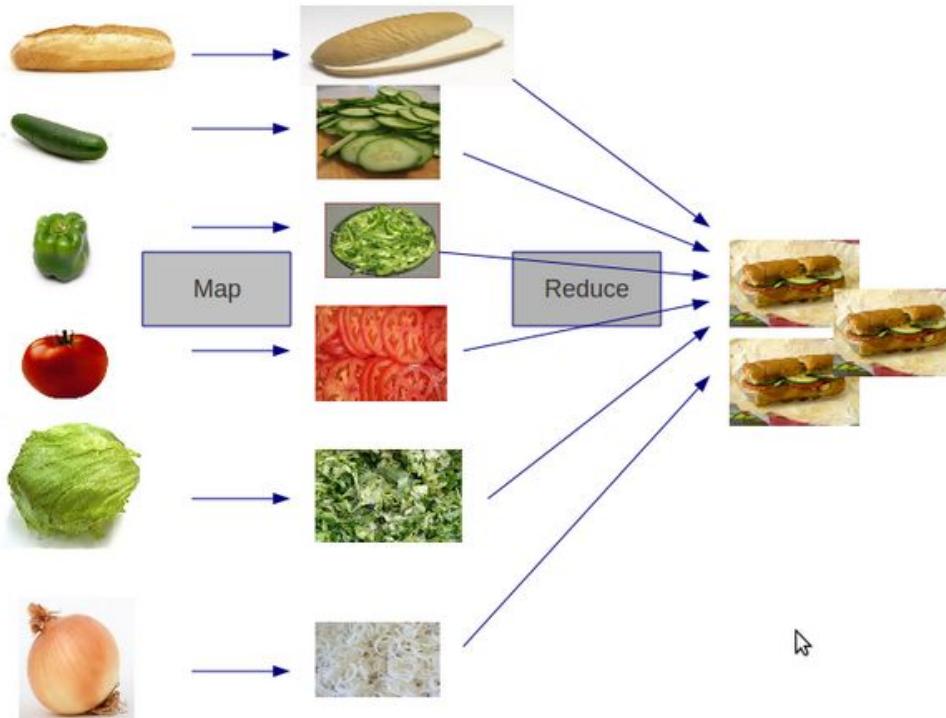


# Aggregate

- Es el reduce
- (1) es un substractor

```
1 final KTable<Song, Long> songPlayCounts =
2     songPlaysKGroupedTable.aggregate(TopFiveSongs::new,
3         (aggKey, value, aggregate) -> { aggregate.add(value); return aggregate; },
4         (aggKey, value, aggregate) -> { aggregate.remove(value); return aggregate; }, ①
5         topFiveSerde,
6         "top-five-songs-by-genre"
7     );
```

# Map- Reduce

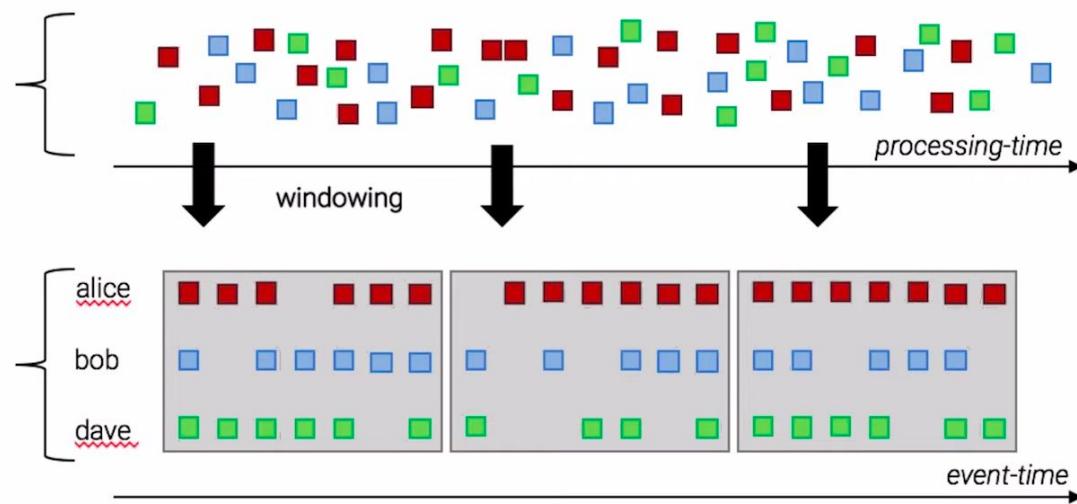




# Windowing

```
1 groupedBySongId.count("song-play-count");
2 groupedBySongId.count(TimeWindows.of(TimeUnit.MINUTES.toMillis(5)), "song-play-count-windowed");
```

El windowin es segun el timestamp del evento



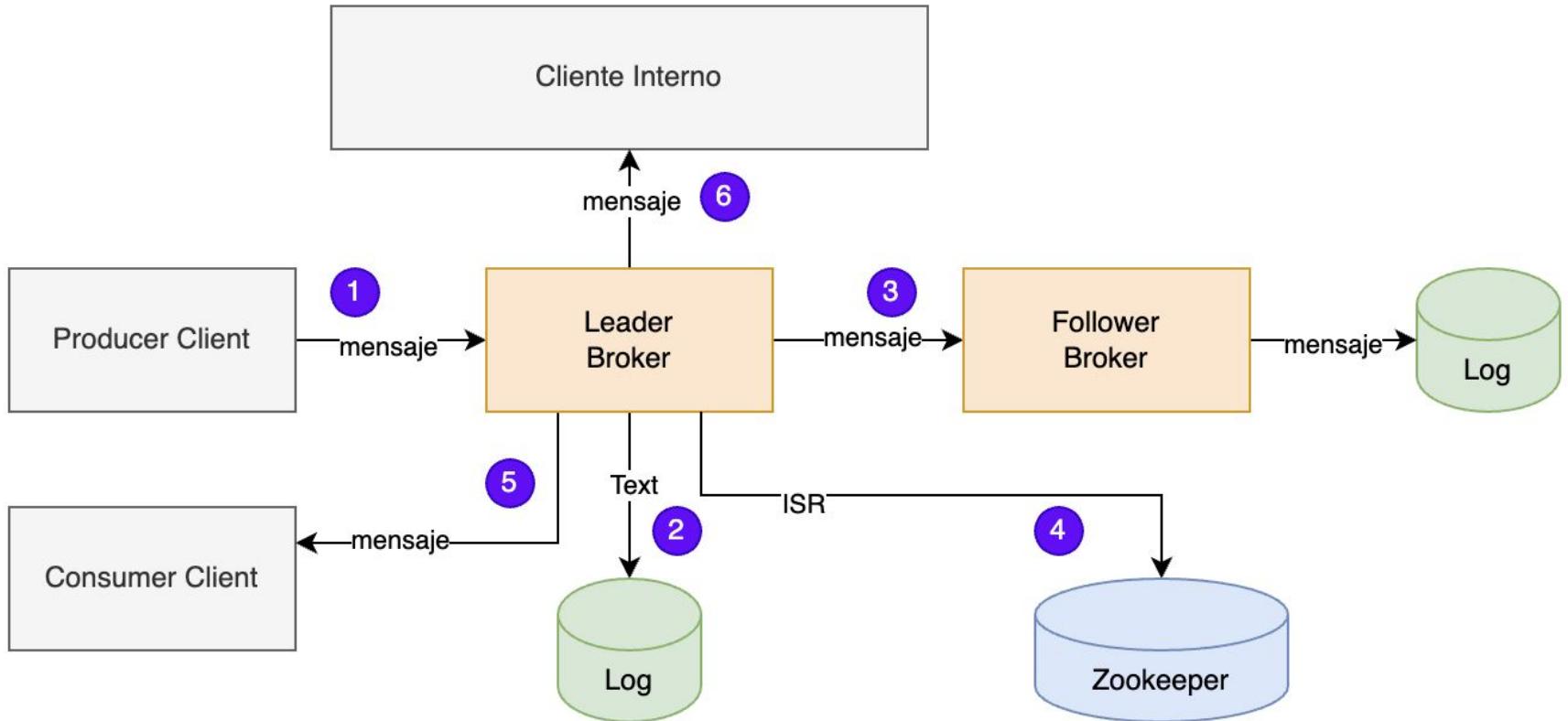
# Security



# Security

Kafka usa una serie de procedimientos de seguridad para establecer y mantener confidencialidad, integridad y disponibilidad de los datos:

- Autenticación establece tu identidad y determina QUIEN eres
- Autorización determina QUÉ te está permitido hacer
- Encriptación protege tus datos
- Auditoría que recoge qué has hecho y qué has intentado hacer
- Control de cuotas que determina cuántos recursos puedes usar





# **Una instalación segura garantiza**

- Autenticidad del cliente
- Autenticidad del servicor
- Privacidad de datos
- Integridad de datos
- Control de acceso
- Auditabilidad
- Disponibilidad



# Protocolos de seguridad

- Los brokers de kafka están configurados con listeners. Cada listener puede tener sus propiedades de seguridad.
- Kafka soporta
  - TLS (SSL) tanto en encriptación como en client/server authentication
  - SASL en autenticación.
  - Cada protocolo combina una forma de transporte (PLAINTEXT ó SSL) con una forma opcional de autenticación: (SSL ó SASL)
    - PLAINTEXT
    - SSL
    - SASL\_PLAINTEXT
    - SASL\_SSL



# Configuracion

- El listener usado para comunicación interbroker está en inter.broker.listener.name o security.inter.broker.protocol.

Ejemplo server (SSL interno y SASL\_SSL externo):

- listeners=EXTERNAL://:9092 INTERNAL://10.0.0.2:9093, BROKER://10.0.0.2:9094
- advertised.listeners=EXTERNAL://broker1.example.com:9092,  
INTERNAL://broker1.local:9093, BROKER://broker1.local:9094
- listener.security.protocol.map=EXTERNAL:SASL\_SSL, INTERNAL:SSL, BROKER: SSL
- Inter.broker.listener.name = BROKER

Cliente:

- security.protocol=SASL\_SSL
- Bootstrap.servers: broker1.example.com:9092: broker2.example.com:9092



# Autenticación: SSL

El certificado digital del servidor se verifica por el cliente para establecer la identidad del servidor. Si la autenticación cliente usando SSL está ‘enabled’ el servidor asimismo verifica el certificado digital del cliente para asegurar su identidad.

El tráfico está encriptado, lo que impide el patrón de transferencia de copia cero con SSL. Esto supone un overhead, según el patrón de tráfico, del 20% al 30%

Para configurarlo los brokers han de configurar una key store conteniendo la clave privada del broker y el certificado, y los clientes han de ser configurados con una trust store con el certificado del broker o un CA correspondiente. Los certificados pueden contener el SAN o el Common Name para comprobar el hostname. El uso de wildcards es válido.

Para que los broker autentiquen a los clientes se ha de incluir client.auth=required en su configuración.



# Práctica

Creamos un key-pair para el CA y lo guardamos

```
keytool -genkeypair -keyalg RSA -keysize 2048 -keystore  
server.ca.p12 -storetype PKCS12 -storepass server-ca-password  
-keypass server-ca-password -alias ca -dname "CN=BrokerCA" -ext  
bc=ca:true -validity 365
```

Exportamos el certificado publico de la CA a server.ca.crt. Se incluirá en las trust stores y en las cadenas de certificados.

```
keytool -export -file server.ca.crt -keystore server.ca.p12 -storetype  
PKCS12 -storepass server-ca-password -alias ca -rfc
```

Si usamos wildcard hostnames la misma key store podrá ser usada para todos los brokers

```
keytool -genkey -keyalg RSA -keysize 2048 -keystore  
server.ks.p12 -storetype PKCS12 -storepass server-ks-password  
-keypass server-ks-password -alias server -dname "CN=Kafka,  
O=Iprocuratio, C=ES" -ext bc=ca:true -validity 365  
  
keytool -certreq -file server.csr -keystore server.ks.p12  
-storetype PKCS12 -storepass server-ks-password -keypass  
server-ks-password -alias server  
  
keytool -gencert -infile server.csr -outfile server.crt  
-keystore server.ca.p12 -storetype PKCS12 -storepass  
server-ca-password -alias ca -ext SAN=DNS:localhost -validity  
365  
  
cat server.crt server.ca.crt > serverchain.crt  
  
keytool -importcert -file serverchain.crt -keystore  
server.ks.p12 -storepass server-ks-password -keypass  
server-ks-password -alias server -storetype PKCS12 -noprompt
```



Si se usa TLS para la comunicación inter-broker, creamos una trust-store para los brokers con el CA para permitir a los brokers autenticarse uno contra otro

```
keytool -import -file server.ca.crt -keystore server.ts.p12  
-storepass server-ts-password -alias server -storetype PKCS12  
-noprompt
```

Generamos una trust store para los clients con los CA certificates del broker

```
keytool -import -file server.ca.crt -keystore client.ts.p12  
-storepass client-ts-password -alias ca -storetype PKCS12  
-noprompt
```

Si la autenticación TLS está establecida, a partir de ahora, los clientes necesitarán ser configurados con una key store, pasos similares a los anteriores. En la prueba sólo haremos pruebas basados en comunicación inter-broker



Para la comunicación inter-broker necesitamos configurar TLS para los brokers. Los brokers necesitan una trust-store sólo si TLS es usada para la comunicación inter-broker o un la autenticación de cliente está establecida

```
ssl.keystore.location= /Users/atienda/desarrollo/cursos/kafka-admin/server.ks.p12
```

```
ssl.keystore.password=server-ks-password
```

```
ssl.key.password=server-ks-password
```

```
ssl.keystore.type=PKCS12
```

```
ssl.truststore.location=/Users/atienda/desarrollo/cursos/kafka-admin/server.ts.p12
```

```
ssl.truststore.password=server-ts-password
```

```
ssl.truststore.type=PKCS12
```

```
ssl.client.auth=required
```



# SASL

SASL puede ser combinado con TLS como la capa de transporte para proveer un canal seguro con autenticación y encriptación. Kafka tiene soporte nativo para algunos mecanismos SASL de uso común:

- GSSAPI (kerberos - Active Directory, OpenLDAP)
- PLAIN: Usuario y password que se usa normalmente con un server-side callback para verificar los passwords desde una fuente externa
- SCRAM-SHA-256 y SCRAM-SHA-512: Autenticación por usuario y password sin necesidad de fuente externa
- OAUTHBEARER. Con OAuth2 bearer tokens

Kafka usa la Java Authentication & Authorization Service (jaas) para la configuración de SASL



# Práctica - SASL/SCRAM

```
bin/kafka-configs.sh --zookeeper localhost:2181 --alter --add-config  
'SCRAM-SHA-512=[password=password]' --entity-type users --entity-name  
user
```

```
bin/kafka-configs.sh --zookeeper localhost:2181 --alter --add-config  
'SCRAM-SHA-256=[password=admin-secret],SCRAM-SHA-512=[password=admin-  
secret]' --entity-type users --entity-name admin
```



# Práctica - SASL/SCRAM

Se pueden activar uno o varios mecanismos en un listener. El usuario y el password será requerido sólo si el listener se usa para la comunicación inter-broker

```
sasl.enabled.mechanisms=SCRAM-SHA-512
```

```
sasl.mechanism.inter.broker.protocol=SCRAM-SHA-512
```

```
listener.name.external.scram-sha-512.sasl.jass.config=org.apache.kafka.common.security.scram.ScramLoginModule required username="kafka" password="kafka-password"
```



Los clientes se tendrán que configurar con SASL y la configuración JAAS del cliente debe incluir el username y el password:

```
sasl.mechanism=SCRAM-SHA-512
```

```
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required  
username="user" password="password"
```



# ACLAUTHORIZER

ACLAUTHORIZER soporta control de acceso de grano fino a los recursos de Kafka usando Listas de control de acceso (ACL). Se guardan en Zookeeper y se cachean en la memoria de cada broker para disponer de búsqueda con más altas prestaciones.

# Monitoring



# ¿Dónde están las métricas?

Todas las métricas expuestas por Kafka pueden ser accedidas vía la interfaz de Java Management Extensions (JMX)

Suele haber un proceso separado que se ejecuta en el sistema conectando con esta interfaz JMX como con NagiosXI y el plugin jmxtrans. También se puede incluir un agente JMX que se ejecuta directamente en el proceso de Kafka como jolokia o mx4j



# Fuentes de métricas

- Aplicación: las de jmx
- Logs: los logs de kafka en si mismos
- Infraestructura
- Clientes



# Health Checks

Hay que asegurarse de que tienes alguna forma de monitorizar la salud completa de los proceso de aplicación. Hay dos formas:

- Un proceso externo que reporta si el broker está up o down
- Alertas en la carencia de métricas reportadas por un broker (stale metrics)

Si bien el segundo método funciona, es difícil saber si ha caído un broker o el sistema de monitorización.

Para el broker kafka, puede ser simplemente conectar al puerto externo para comprobar si responde.



# Service Level Objectives

- Un SLI (Service Level Indicator) es una métrica que describe un aspecto de la confiabilidad del servicio. Debe estar alineado con la experiencia de los clientes, y cuanto más objetivos son, mejor son.
- Un SLO, también llamado service level threshold (slt) combina un sli con un valor objetivo. Una forma común de expresarlo es en número de nueves (99,9%, tres nueves). Incluye también un intervalo de tiempo para calcular las métricas.
- Un SLA es un contrato entre un proveedor de servicios y un cliente que contiene varios SLO.



# Tipos de SLI comunes en kafka

- Latencia
- Calidad
- Seguridad
- Throughput



# Diagnosticando problemas de cluster

- Single broker problems
- Clusters sobrecargados
- Problemas con el controlador



# CLUSTER LEVEL

- Carga no balanceada. Cuando entre máquinas estos números no son los mismos:
  - Partition count
  - Leader partition count
  - All topics messages in rate
  - All topics bytes in rate
  - All topics bytes out rate
- Quedarse sin recursos
  - Uso CPU
  - Entrada de datos de red
  - Salida de datos de red
  - Media de tiempo de espera de disco
  - % de uso de discos



# MÉTRICAS A TENER EN CUENTA

`kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions`

Number of under-replicated partitions ( $| \text{ISR} | < | \text{current replicas} |$ ). Replicas that are added as part of a reassignment will not count toward this value. Alert if value is greater than 0.

`kafka.server:type=ReplicaManager,name=UnderMinIsrPartitionCount`

Number of partitions whose in-sync replicas count is less than minIsr.



# Broker Metrics

`kafka.controller:type=KafkaController, name=ActiveControllerCount`

Number of active controllers in the cluster. Alert if the aggregated sum across all brokers in the cluster is anything other than 1 because there should be exactly one controller per cluster.

`kafka.server:type=KafkaRequestHandlerPool, name=RequestHandlerAvgIdlePercent`

Average fraction of time the request handler threads are idle. Values are between 0 (all resources are used) and 1 (all resources are available).

Por debajo de 0,2 -> tenemos un potencial problema -> rebalanceo



# Broker Metrics

`kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec`

Byte-in rate from producers (writing).

Uno de ellos recibe más que otro —

`kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec`

Byte-out rate from clients (reading).

`kafka.server:type=BrokerTopicMetrics,name=ReplicationBytesOutPerSec`

Byte-out rate to other brokers.



`kafka.server:type=ReplicaManager,name=LeaderCount`

Number of leaders on this broker. This should be mostly even across all brokers. If not, set

`auto.leader.rebalance.enable` to `true` on all brokers in the cluster

`kafka.controller:type=KafkaController,name=OfflinePartitionsCount`

Number of partitions that don't have an active leader and are hence not writable or readable. Alert if value is greater than 0.

- Todos los brokers con replicas de esta particion estan ko
- Si la votación de leader election no está clara -> unclear leader election -> enabled



# Problemas de kafka habituales

- Falta de memoria:
  - RAM -> balanceo o nuevos brokers
  - DISCO -> falta de espacio -> falta de espacio en inode
- Lags -> lags por topic -> si crecen continuamente -> no procesamos a la velocidad que estamos produciendo
- Levantar consumidores sin tener control de las particiones que hay en el sistema -> si no hay suficientes habrá que crear y balancear
- El coste de las tareas administrativas -> es alto
- Mensajes duplicados -> hay un tema del productor que hay que mirar
- Lecturas - leer dos veces el mismo o saltarme uno. Intentar comitear al final. Ante cualquier fallo ese mensaje se va a dead-letter-queue (sobre cualquier excepción) -> mensajes que no a podido procesar
- Procesamiento duplicado -> groupid.
-