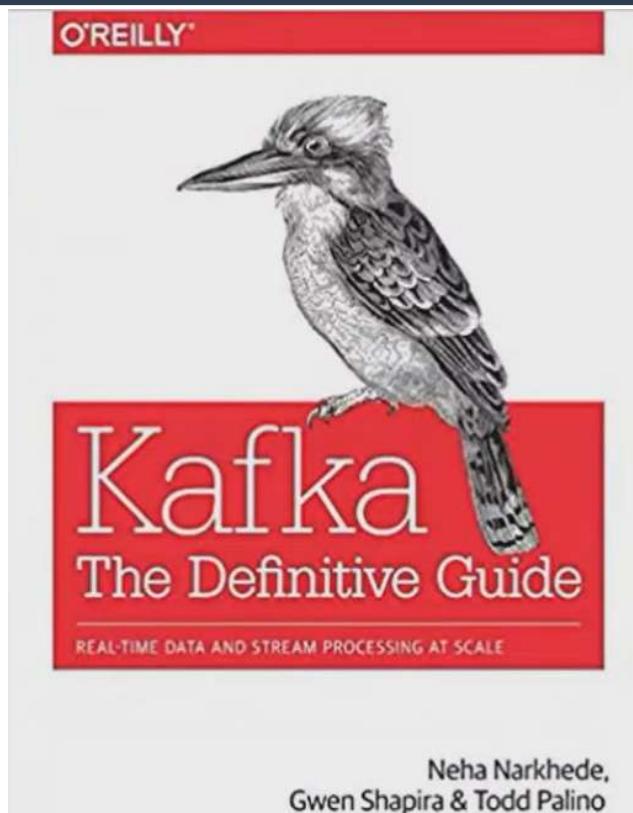




Módulo Profesional: Big Data Aplicado

Kafka

ÍNDICE



<https://www.confluent.io/resources/ebook/kafka-the-definitive-guide/>

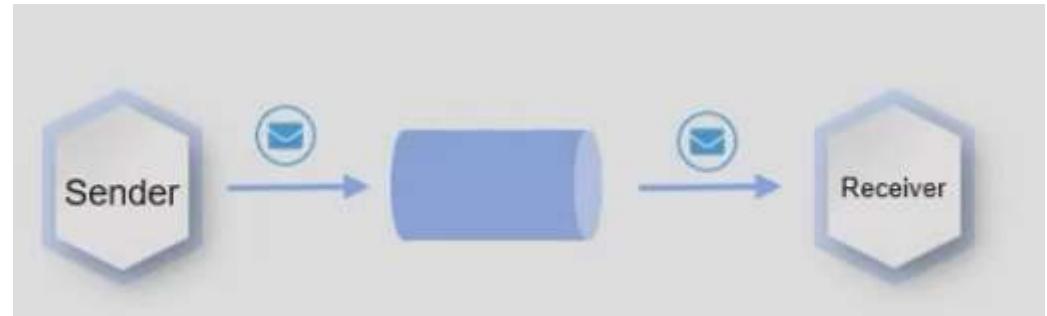
ÍNDICE

1. Introducción
2. Arquitectura Central
3. Características
4. Kafka Streams
5. Ecosistema

Introducción

Sistema de mensajería punto a punto

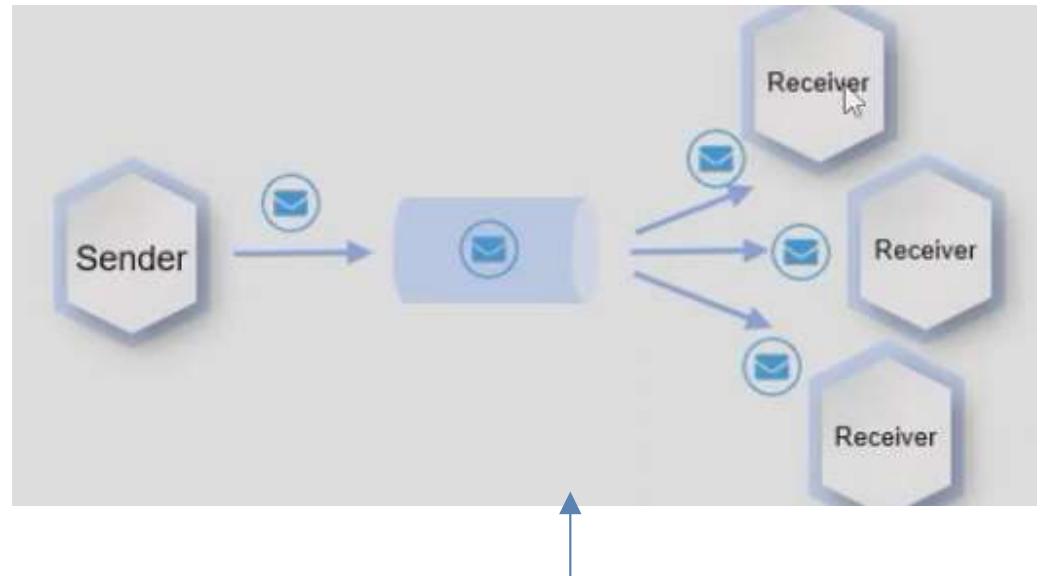
En un sistema punto a punto, los mensajes se almacenan en una cola. Uno o más consumidores pueden consumir los mensajes en la cola, pero un mensaje en particular solo puede ser consumido por un máximo de un consumidor.



Introducción

Sistema de mensajería Publish-Subscribe

En el sistema de publicación-suscripción, los mensajes se almacenan en un **topic**. A diferencia del sistema punto a punto, los consumidores pueden suscribirse a uno o más topics y consumir todos los mensajes de ese topic. En el sistema de publicación-suscripción, los productores de mensajes se llaman **publicadores** y los consumidores de mensajes se llaman **suscriptores**.



Kafka es de este tipo

Apache Kafka

Kafka como un sistema de mensajería Publicación/Suscripción

Kafka como un sistema de almacenamiento

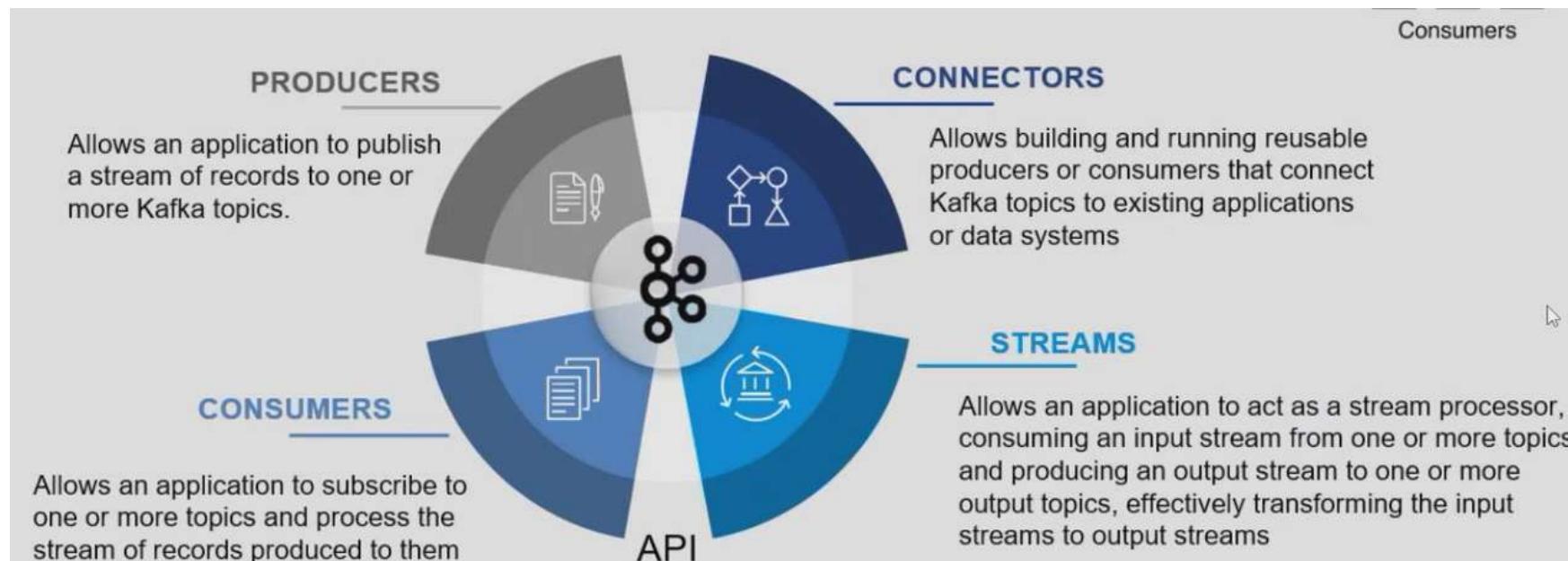
Sistema de archivos distribuido para el registro de confirmaciones (commit log)

Características

- Rápido, Baja latencia
- Escalable
- Duradero
- Tolerante a fallos
- Confiable
- Replicación

Apache Kafka

Apache Kafka es una cola de mensajes pub/sub en tiempo real, tolerante a fallos y altamente escalable, diseñada como un registro de transacciones distribuido.

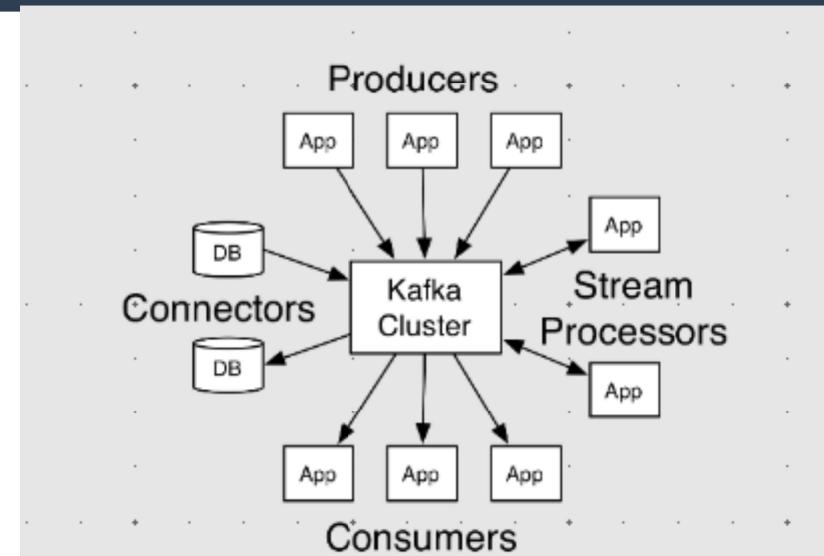


Kafka. Connectors

Los conectores de Kafka son componentes que permiten la ingestión y exportación de datos entre Kafka y otros sistemas. Existen dos tipos principales de conectores:

Source Connectors (Conectores de origen): se utilizan para extraer datos de fuentes externas y publicarlos en temas de Kafka. *Ejemplo:* Un conector que lee datos de una base de datos MySQL y los envía a Kafka.

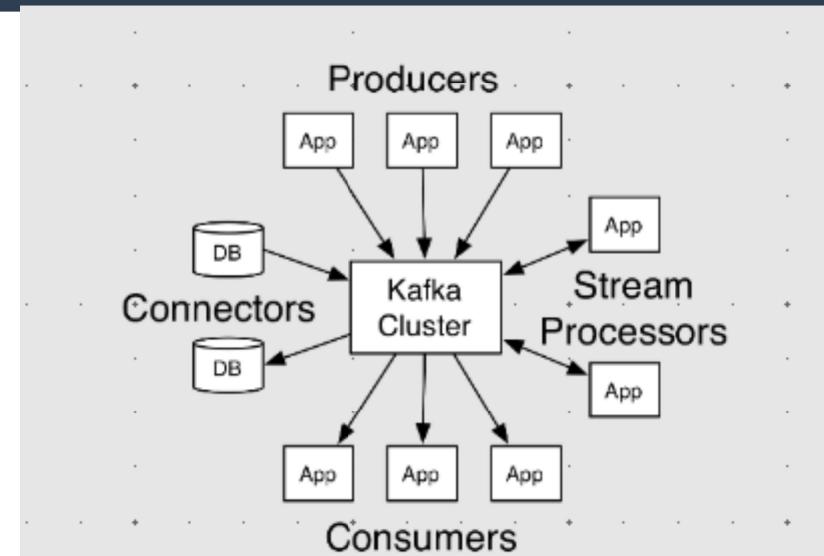
Sink Connectors (Conectores de destino): se utilizan para leer datos de Kafka y escribirlos en un sistema externo. *Ejemplo:* Un conector que toma datos de un topic de Kafka y los almacena en Amazon S3.



Kafka. Connectors

Conectores más conocidos:

- JDBC Source Connector → Extrae datos de bases de datos como MySQL, PostgreSQL, Oracle.
- Elasticsearch Sink Connector → Envía datos de Kafka a Elasticsearch para búsquedas avanzadas.
- S3 Sink Connector → Guarda datos en Amazon S3.
- MongoDB Connector → Integra Kafka con MongoDB.
- Debezium Connector → Permite Change Data Capture (CDC) desde bases de datos como MySQL, PostgreSQL y MongoDB.



Confluent Kafka

Confluent es la empresa detrás de Apache Kafka, fundada por los creadores de Kafka.

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

Apache Kafka

Licencia Apache v2.0
Kafka Brokers (es decir, los demonios)
Producer y Consumer API
Streams API
Connect API

Confluent Open

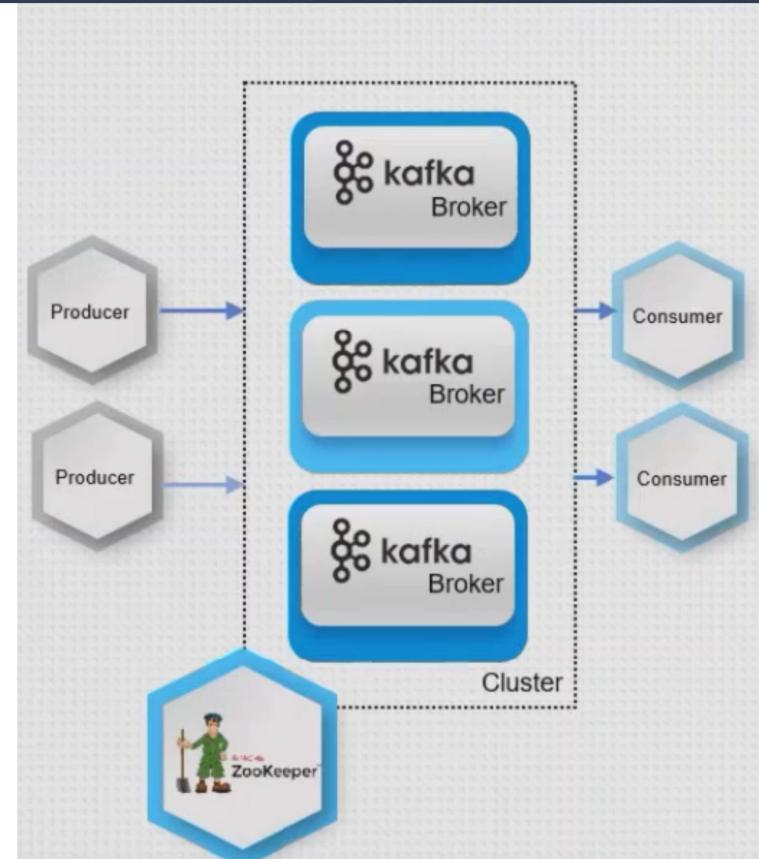
Clientes (Java, C, C++, Python, .)
Conectores (JDBC, Elasticsearch, HDFS, etc.)
Schema Registry
REST Proxy
KSQL

Confluent Enterprise

Control Center para monitoreo
Replicación en múltiples centros de datos
Confluent Cloud:
AWS
GCP
Trainings & certificaciones
Soporte 24/7

Kafka. Brokers

- Operan como parte de un clúster.
- Almacenan particiones y réplicas de topics: Los datos escritos en Kafka se guardan en disco y se replican para tolerancia a fallos.
- Reciben mensajes de los productores:
 - Les asignan offsets.
 - Confirman los mensajes y los almacenan en disco.
- Sirven a los consumidores:
 - Responden a solicitudes de recuperación de particiones.
 - Responden con los mensajes: Kafka utiliza un método de copia cero (zero-copy) para enviar los mensajes directamente desde el archivo al canal de red, sin buffers intermedios, lo que mejora el rendimiento.



Kafka. Brokers

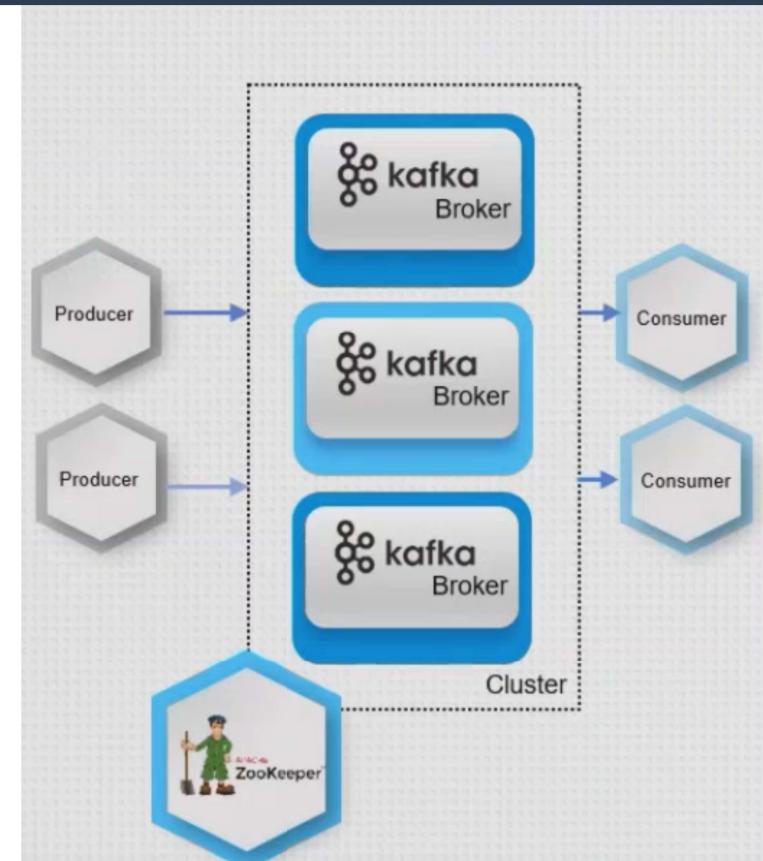
TOPIC equivalente a una tabla de Base de Datos (no es una tabla es un equivalente).

Ejemplo: Queremos guardar un topic que nos hemos creado que se llama “Topic Ventas”.

Pregunta: Imaginad que el productor quiere enviar un mensaje que sea una Venta donde envía ¿Qué es lo que he vendido?; ¿A qué precio lo he vendido?; ¿A quién y qué día se lo he vendido? Lo metemos en un JSON.

Si el productor lo envía : ¿se guarda en el primer, segundo y tercer Broker o sólo en alguno de ellos?

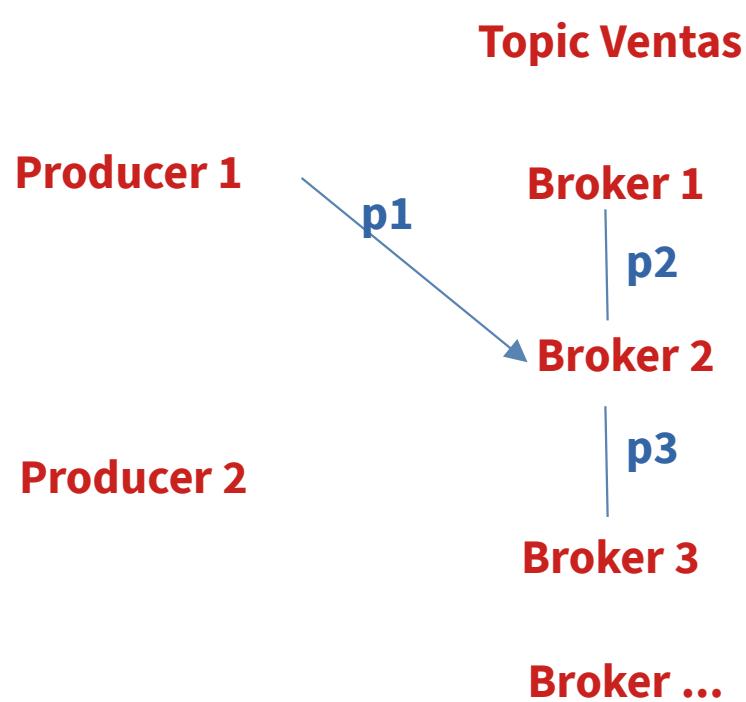
¿Qué sería lo más lógico y lo mejor?



Kafka. Brokers



Kafka. Brokers



Pasos:

p1. Envío mensaje y lo guarda en el servidor Broker 2 (zooKeeper se lo dice). Obtiene respuesta Ok.

P2 Guarda un Backup en el Broker 1

P3 Guarda también un Backup en el Broker 3.

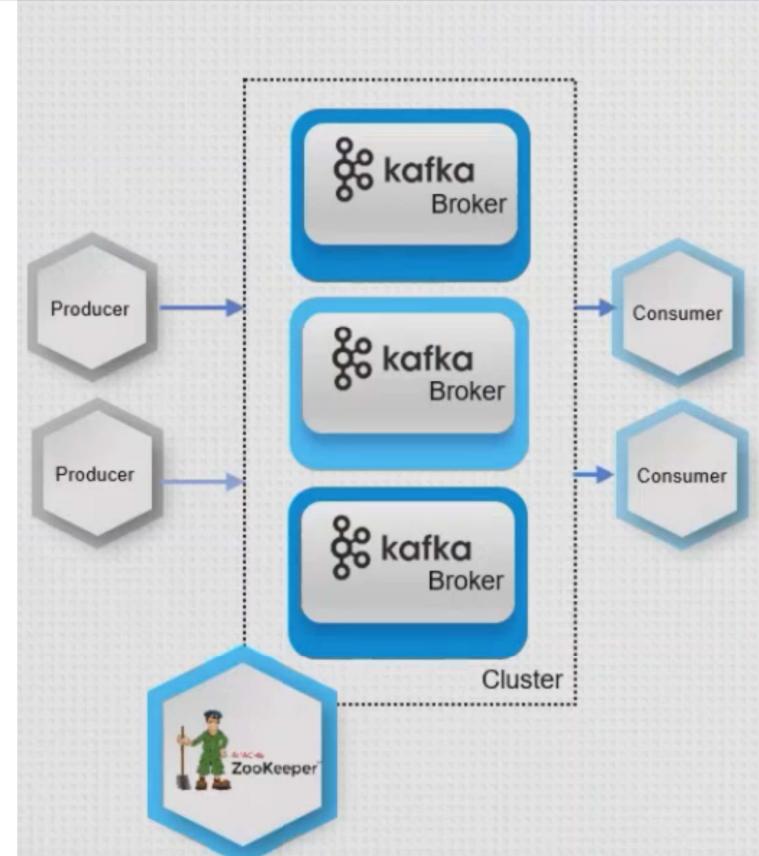
¿Hay algún problema en este comportamiento?

Brokers se conocen entre sí de forma limitada cada uno tiene una IP que se la asigna ZooKeeper y quién asigna quién debe ser Backup.

Kafka. Zookeeper

Lleva el registro del estado de los nodos del clúster de Kafka.

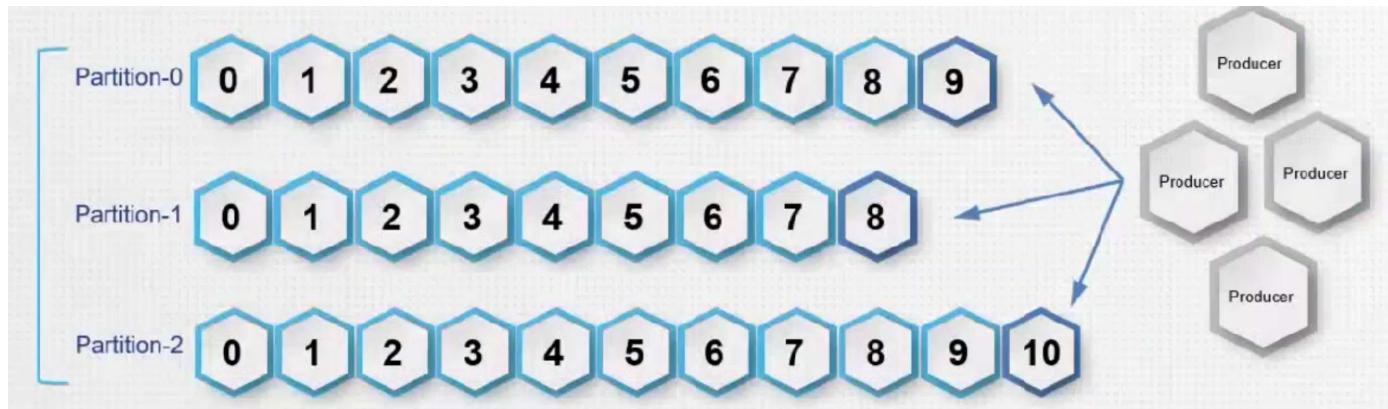
- Sistema distribuido
- Elección del controlador.
- Configuración de topics.
- Listas de control de acceso.



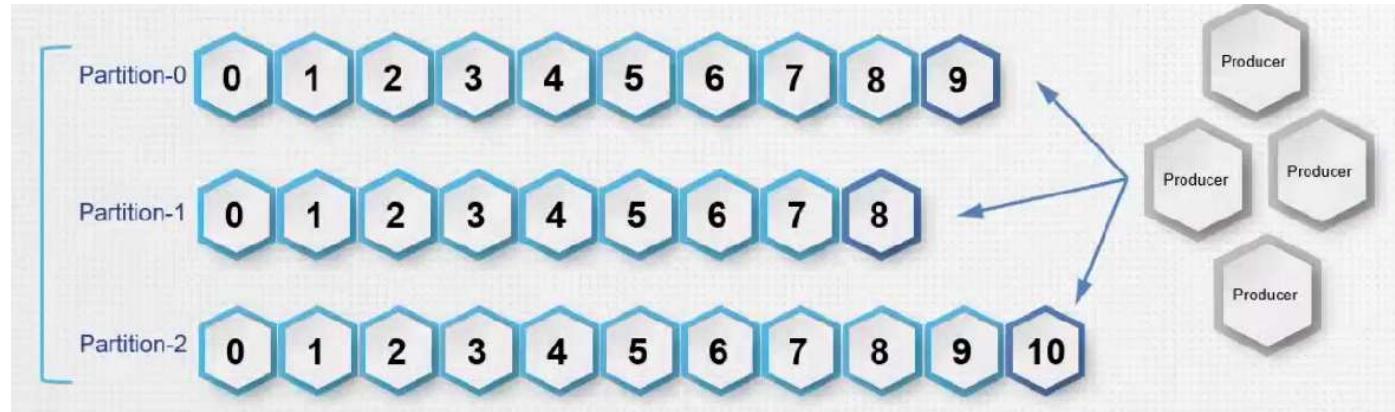
Kafka. Topics



- Un flujo de mensajes que pertenece a una categoría en particular se llama **topic**.
- Un Topic de Kafka está dividido en particiones.
- Cada partición es una secuencia totalmente ordenada de mensajes con un identificador único llamado **offset**.
- **Leader:** cada partición tiene un servidor que actúa como Leader.



Kafka. Topics



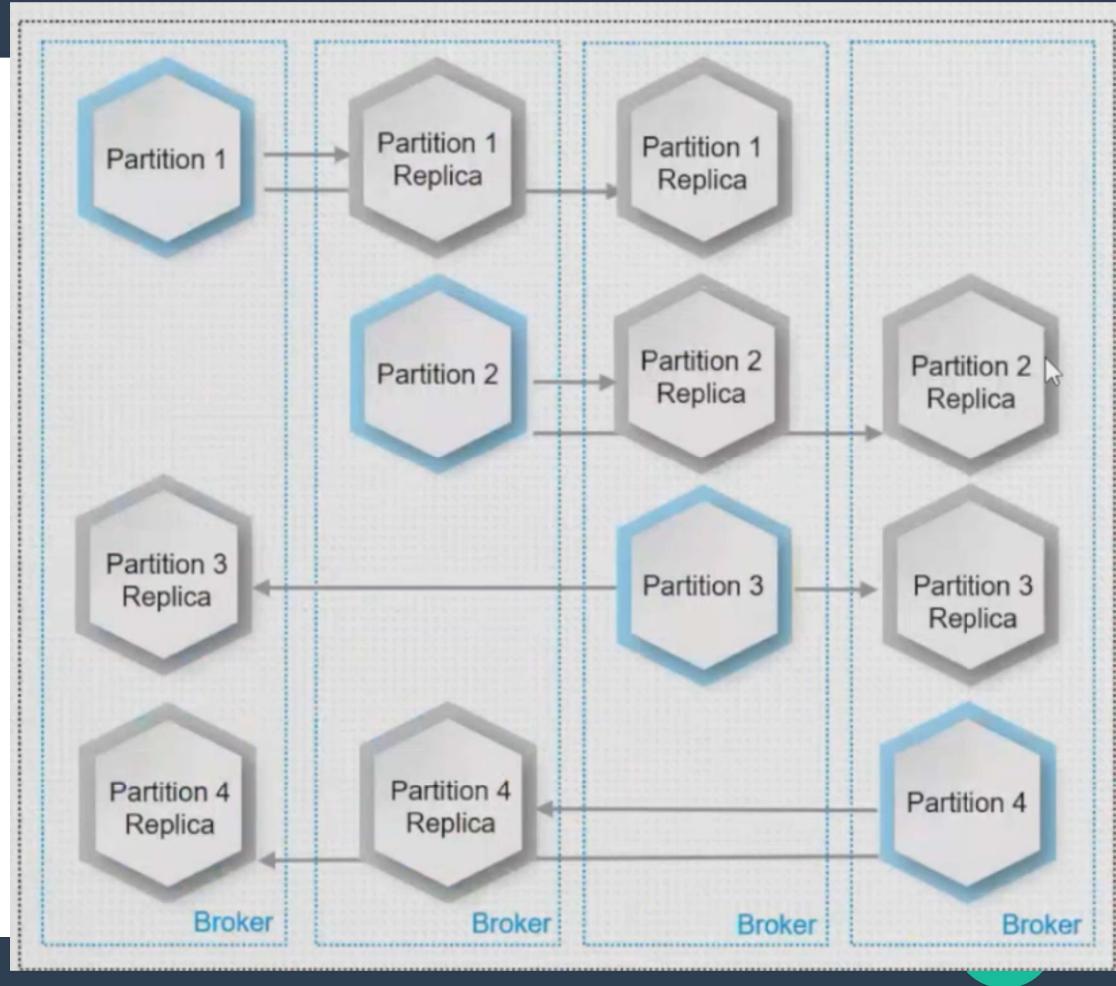
- Si hay **N particiones** en un **topic** y **N brokers**, cada broker tendrá una partición.
- Si hay **N particiones** en un **topic** y más de **N brokers (N + M)**, los primeros N brokers tendrán una partición cada uno y los siguientes M brokers no tendrán ninguna partición para ese topic en particular.
- Si hay **N particiones** en un **topic** y menos de **N brokers (N - M)**, cada broker tendrá una o más particiones, distribuidas entre ellos.

Kafka. Replicas (Bakups)

Las réplicas se distribuyen entre los brokers disponibles:

- Copias de seguridad de una partición
- Se utilizan para prevenir la pérdida de datos.

Estas réplicas no atienden directamente los mensajes; su único trabajo es mantenerse al día con la réplica líder.

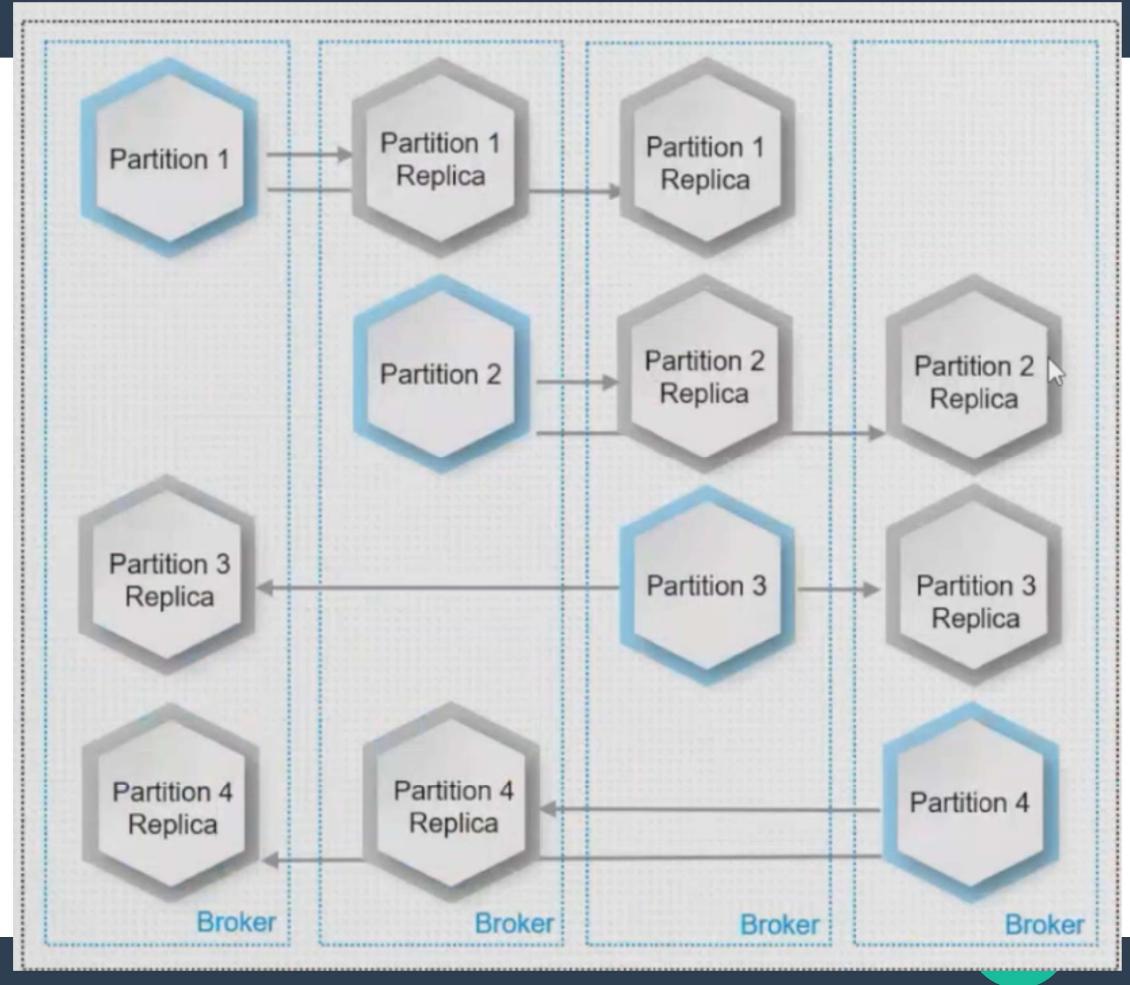


Kafka. Replicas (Backups)

Para ello, las replicas consumen los mensajes desde el líder, de la misma forma que lo haría un consumidor típico de Kafka.

Si una réplica se mantiene razonablemente al día con el líder, se considera parte del conjunto ISR (réplicas sincronizadas).

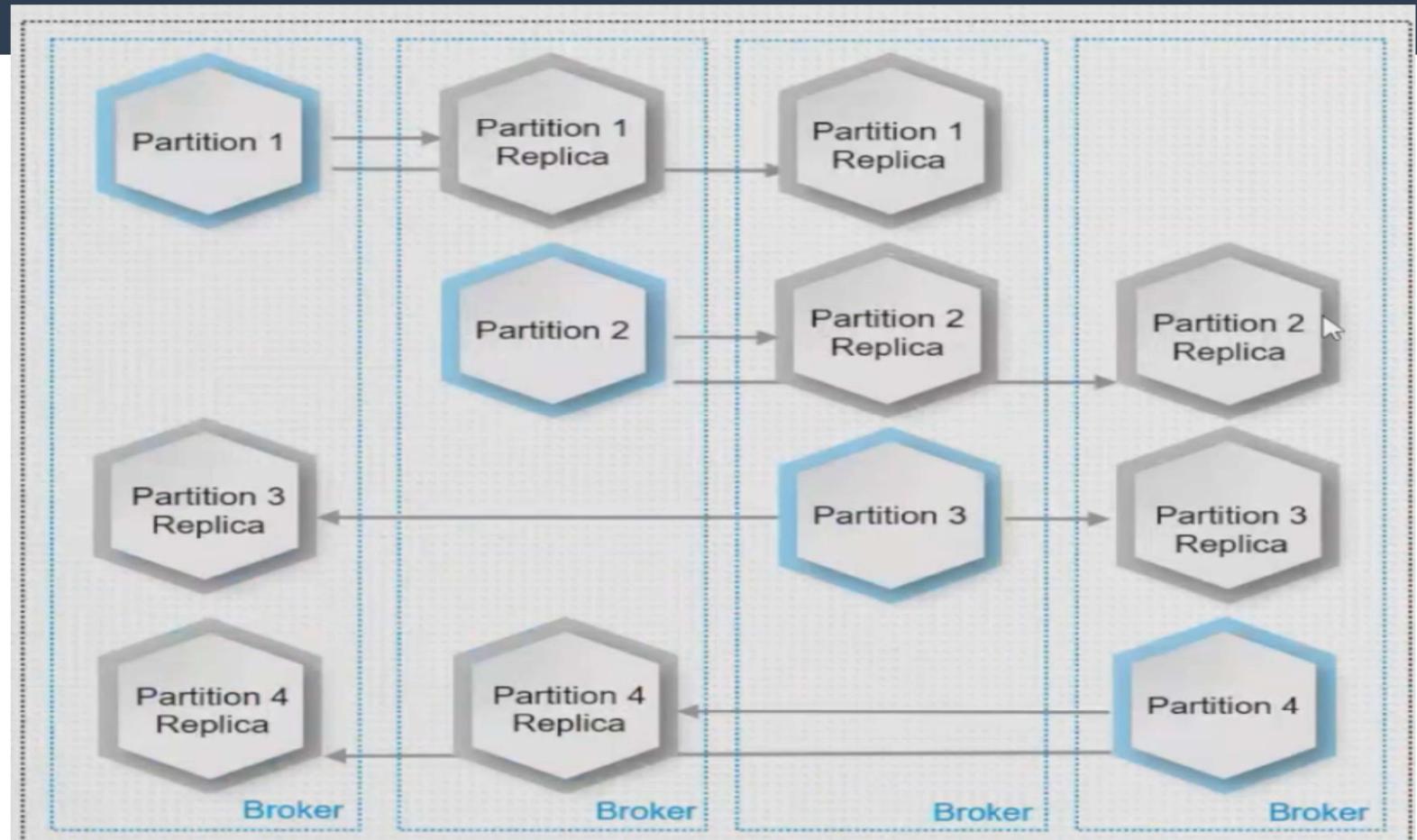
En caso de que la réplica líder falle, una de las réplicas dentro del ISR será elegida automáticamente como nueva líder.



Kafka. Replicas (Backups)

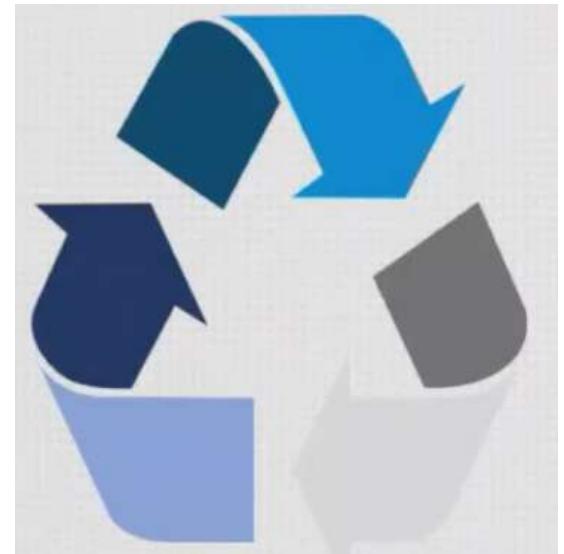
Topic

- 4 particiones
- Factor de replicación 2



Kafka. Record Retention

- Los registros publicados (hayan sido consumidos o no) se retienen durante un período configurable.
- Los registros están disponibles para su consumo hasta que son eliminados según alguno de los siguientes criterios:
 - **Tiempo**: período de retención configurable basado en el tiempo.
 - **Tamaño**: retención configurable basada en el tamaño de la partición.
 - **Compaction** (compactación): conserva únicamente el último registro para cada clave del mensaje.
- El rendimiento no se ve afectado por particiones grandes, por lo que los períodos de retención pueden ser extremadamente largos.
 - ¿Ilimitado? Solo si el almacenamiento también lo es.



Kafka. Log Compaction

Los topics compactados ofrecen un tipo de flujo muy distinto, ya que mantienen únicamente el mensaje más reciente asociado a una clave determinada.

Kafka hace un uso intensivo de este tipo de topics internamente. Por ejemplo, para la persistencia y el seguimiento de los offsets de los consumidores.

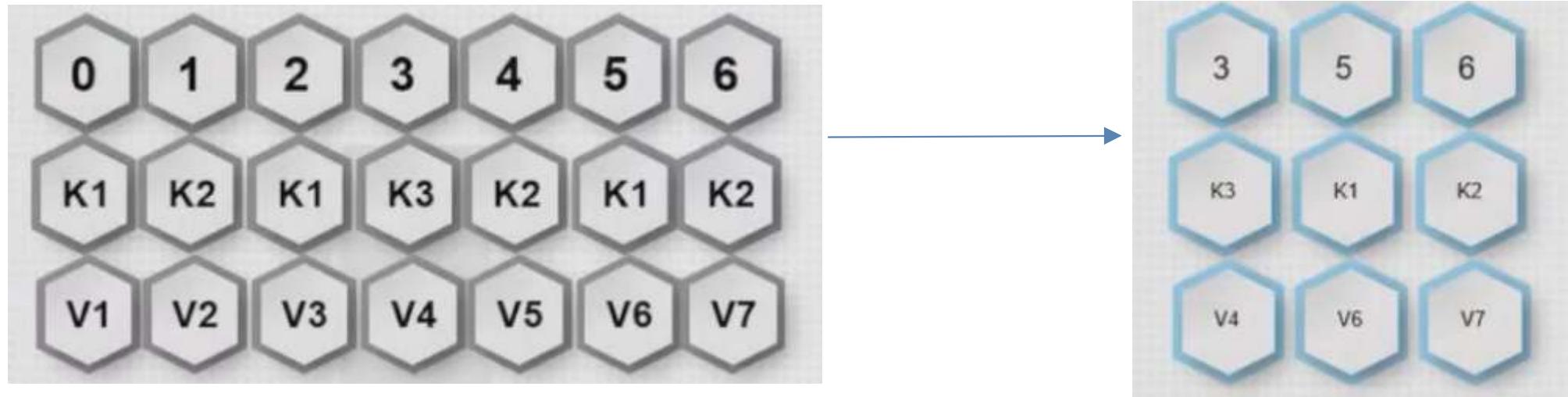
En aplicaciones de usuario, resultan especialmente útiles para datos tipo changelog, donde solo interesa la última actualización. Por ejemplo, para restaurar el estado de un servicio en memoria tras un fallo, recargar una caché, etc.

La compactación del log se ejecuta periódicamente en segundo plano.

La compactación también permite eliminar registros: se hace mediante mensajes con clave y valor nulo (tombstone records).

Todos los offsets siguen siendo válidos, incluso si el registro correspondiente a un offset ha sido compactado.

Kafka. Log Compaction

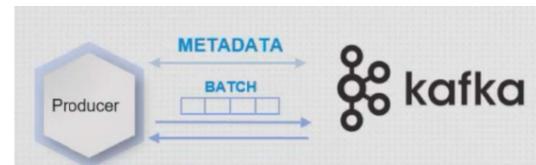


Kafka. Producers

- Un productor de Kafka es una aplicación que actúa como fuente de datos.
- Un productor puede publicar mensajes en uno o varios topics de Kafka.
- Métodos existentes para el envío de mensajes:
 - **Fire-and-forget**: no espera confirmación (ACK), ofrece un rendimiento muy alto. acks=0
 - **Asíncrono**: espera la confirmación del líder. acks=1
 - **Síncrono**: espera la confirmación del líder y de las réplicas. Método seguro. acks=all
- Los productores de Kafka intentan agrupar los mensajes enviados en batches para mejorar el throughput.

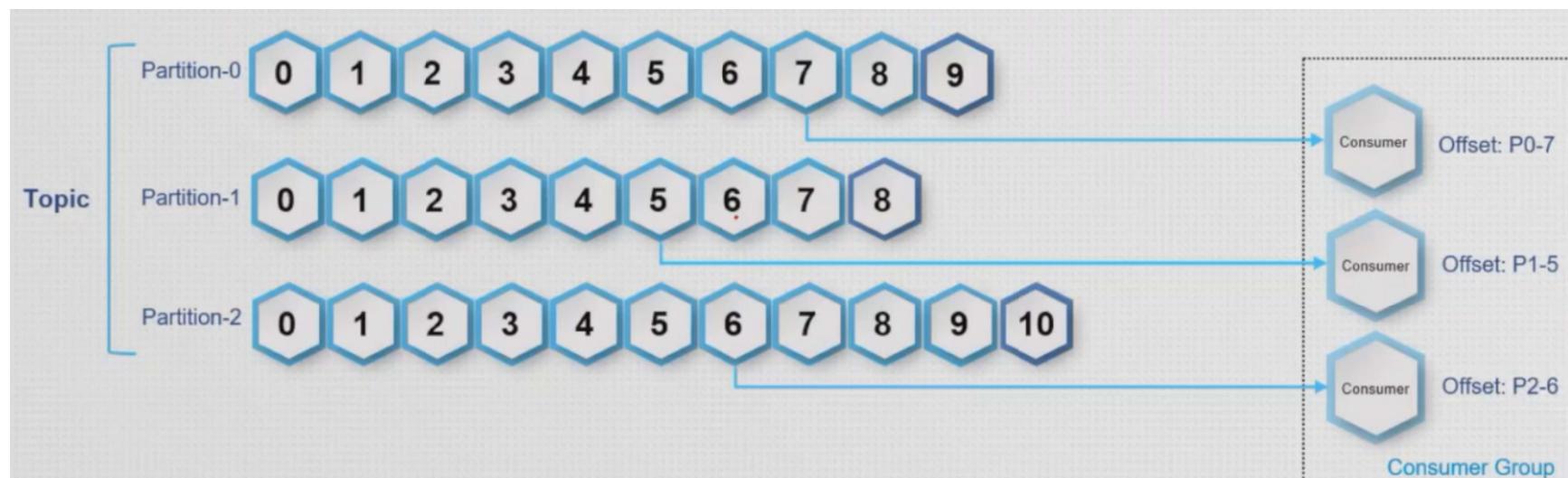
Asignación de particiones:

- Round-robin: distribución equitativa entre particiones.
- Función de partición basada en la clave: por ejemplo, todos los registros con el mismo employee-id se envían a la misma partición



Kafka. Consumers

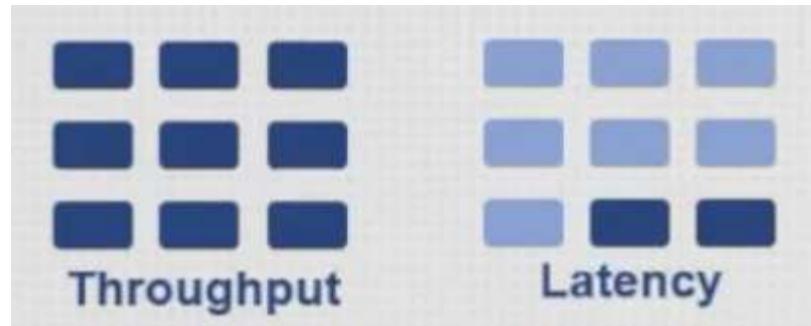
- Las particiones se asignan a los consumidores dentro de un **Consumer Group**.
- Los consumidores recuerdan el offset donde se quedaron. Este offset se guarda (**commit**) en Kafka.
- El proceso de transferir la propiedad de una partición de un consumidor a otro se denomina **rebalance**. Durante el rebalance, hay una breve ventana de indisponibilidad para todo el Consumer Group.



Kafka. Delivery Semantics

- **“At most once” (como máximo una vez).** Un mensaje debe ser entregado, como máximo, una sola vez. Es aceptable perder mensajes si eso evita que se entreguen duplicados.
- **“At least once” (al menos una vez).** Es aceptable que un mensaje sea entregado más de una vez, pero no se debe perder ninguno. El productor garantiza la entrega de todos los mensajes, aunque esto pueda implicar duplicación.
- **“Exactly once” (exactamente una vez).** Cada mensaje debe ser entregado una única vez y no debe perderse ninguno. Entrega más compleja de implementar.

Kafka soporta todas estas modalidades de entrega en función de la configuración del broker y del producer.



Kafka Streams

- Stream (Flujo):

- Un Stream es la abstracción más importante que proporciona Kafka Streams; representa un conjunto de datos no acotado (**unbounded**) que se actualiza de forma continua.

- Una partición de Stream es una secuencia ordenada, reproducible y tolerante a fallos de registros de datos inmutables, donde cada registro de datos se define como un par clave-valor (key-value).

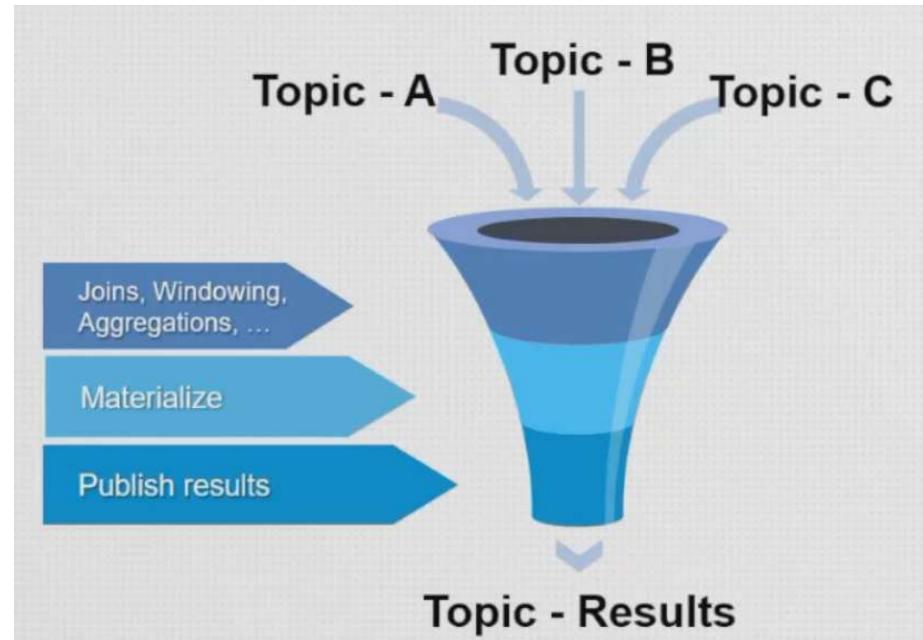
- Stream processing application (Aplicación de procesamiento de flujos):

- Cualquier programa que haga uso de la librería Kafka Streams. Puede definir su lógica computacional mediante una o varias topologías de procesadores (processor topologies).

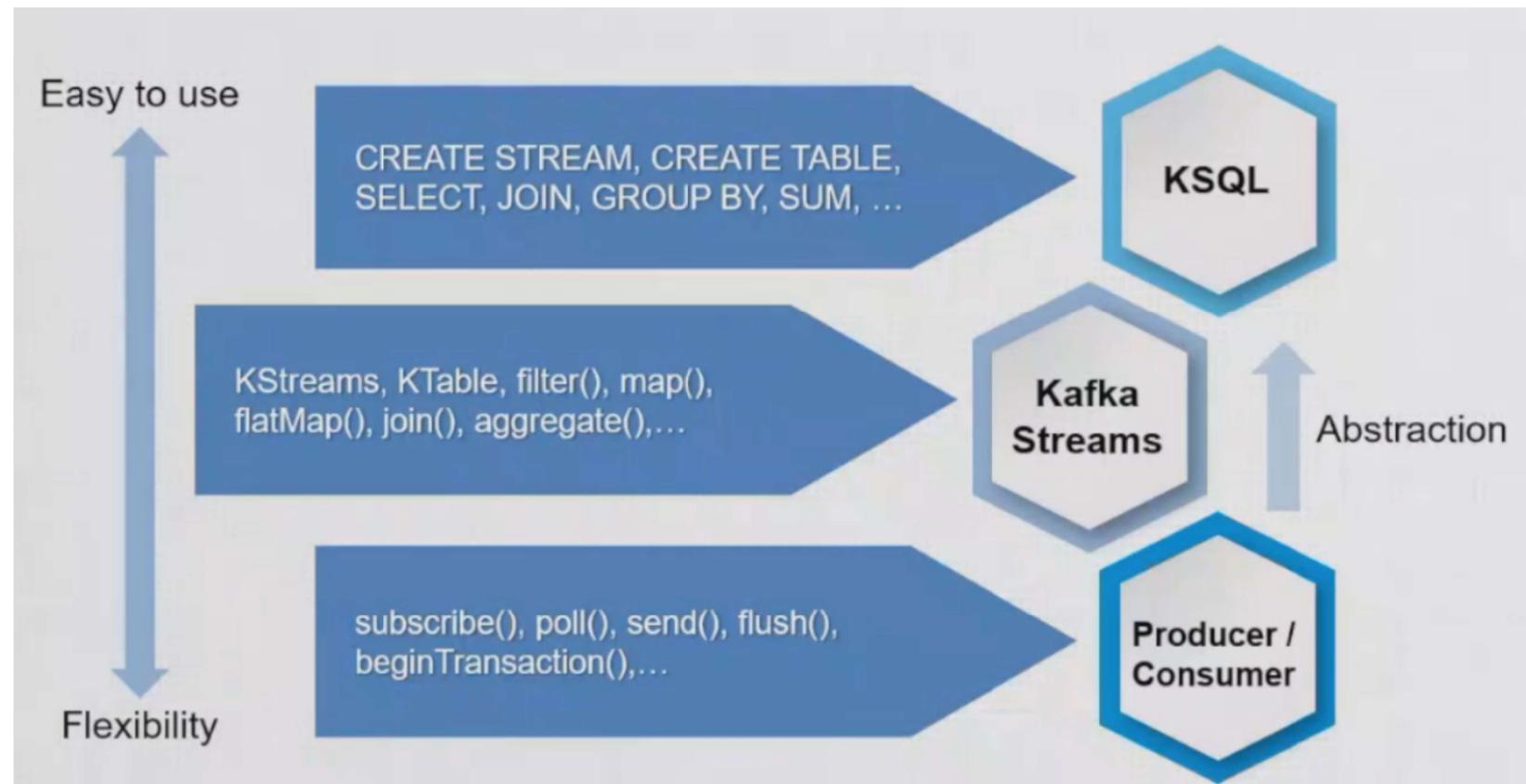
Kafka Streams

Kafka Streams DSL: Es una API de alto nivel que proporciona de forma nativa (out of the box) las operaciones de transformación de datos más comunes, como map, filter, join y agregaciones. Está orientada a facilitar el desarrollo de aplicaciones de procesamiento de flujos mediante una sintaxis declarativa y expresiva.

Processor API: Es una API de bajo nivel que permite añadir y conectar procesadores de forma manual, así como interactuar directamente con los **state stores** (almacenes de estado). Ofrece mayor flexibilidad y control sobre el procesamiento de los datos, siendo ideal para casos de uso avanzados o personalizados.



Kafka. KSQL



Kafka. KSQL

KSQL está construido sobre Kafka Streams:

Motor de KSQL (KSQL Engine): Es el encargado de procesar las sentencias y consultas escritas en KSQL.

Interfaz REST: Permite el acceso de clientes al motor de KSQL mediante peticiones HTTP.

KSQL CLI: Consola que proporciona una interfaz de línea de comandos (Command-Line Interface) para interactuar con el motor de KSQL.

KSQL UI: Interfaz gráfica incluida en Confluent Control Center que permite desarrollar aplicaciones KSQL de forma visual.

