1. **Introducción a Apache Kafka**

En los últimos años el número de aplicaciones a desarrollar por las empresas ha aumentado considerablemente con la llegada de las arquitecturas basadas en microservicios.

Uno de los aspectos más relevantes es la comunicación entre ellos, o la necesidad de tener que integrarse con otros sistemas enviando o recibiendo información.

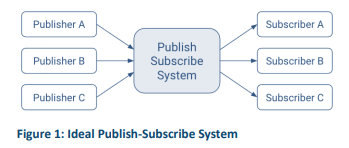
En estos casos, estas comunicaciones deberán ser rápidas, seguras y fiables con una alta disponibilidad.

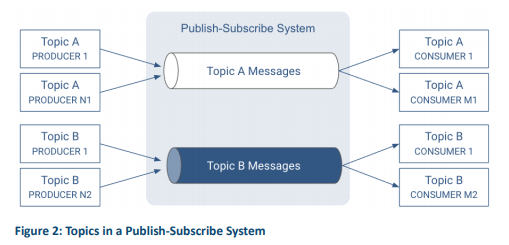
Una de las soluciones para solventar este tipo de casos ha supuesto el uso de **tecnologías basadas en colas de mensajes**, las cuales permiten la **comunicación asíncrona**

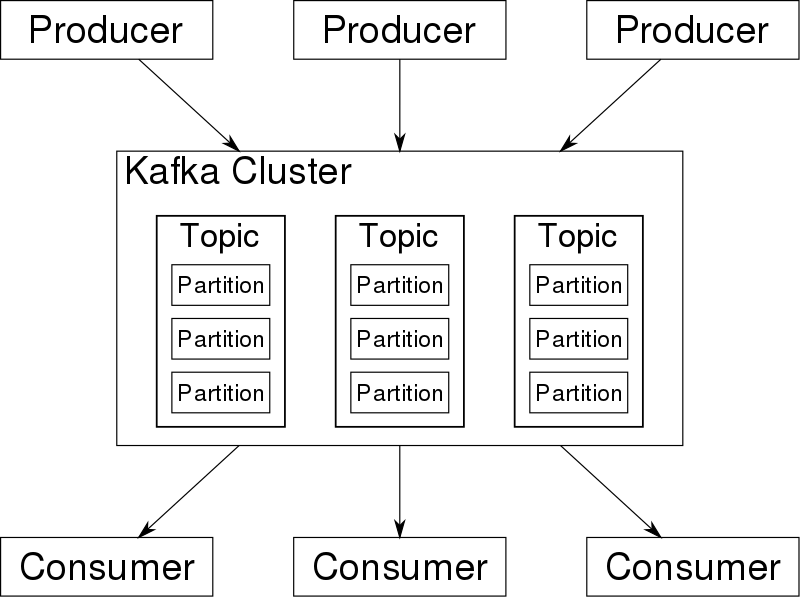
A la hora de utilizar un sistema de colas de mensajes, se ha hecho muy popular el uso de **ActiveMQ** y **RabbitMQ**. Sin embargo, a la hora de enfrentarnos a sistemas que requieren la transmisión de datos a tiempo real encontramos **Apache Kafka como una de nuestras mejores soluciones**.

Apache Kafka es una **plataforma de transmisión distribuida** basada en un sistema de mensajería de **publicación / suscripción**.

Kafka fue creado por LinkedIn y actualmente es un proyecto open source mantenido por Confluent, empresa que está bajo la administración de Apache.







* **Producer**: un productor es un cliente que envía mensajes al servidor de Kafka sobre el tema especificado.
* **Consumer**: los consumidores son los destinatarios que reciben mensajes del servidor de Kafka.
* **Grupo de consumidores:** *Agrupación de uno o más consumidores que trabajan para leer de un topic en base al cumplimiento de algún objetivo / funcionalidad concreta*.

*Cada grupo se considera un "suscriptor" (como si fuera un sólo consumidor)*

*Puede ser un subscriptor de uno o más topic.*

**Mensaje:**

La **unidad de datos** dentro de Kafka se llama mensaje. Piense en esto como una fila en la tabla de la base de datos.

El masaje tiene dos partes: la **llave y el cuerpo**. Ambos son simplemente una matriz de bytes y Kafka no hace nada mágico para leer y dar sentido a estos bytes.

Las claves se utilizan para escribir mensajes en particiones de una manera más controlada. Kafka simplemente encuentra el hash de la clave y lo usa para encontrar el número de partición donde se debe escribir el mensaje (la lógica no es tan simple, por supuesto). Esto asegura que los mensajes con la misma clave siempre se escriban en la misma partición.

**Topics:**

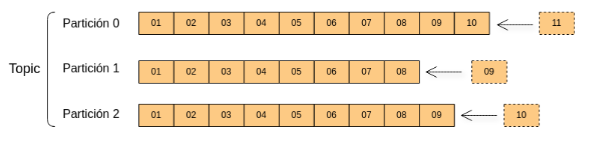
Un tema (Topic) de Kafka es muy similar a una tabla de base de datos.

Siempre se trabaja sobre **topics**. Poniendo un símil con la prensa, un **topic** sería el periódico al que nos hemos subscrito. Solo recibiremos las ediciones (mensajes en Kafka) de ese periódico. Por supuesto una misma persona (suscriptor) mismo puede estar subscrito a muchos periódicos.

Además, los temas se dividen en varias particiones.

Por ejemplo, considere que tenemos un tema con el nombre " activity-log" que tiene 3 particiones con nombres:

* activity-log-0
* activity-log-1
* activity-log-2



**Ejemplo crear Topic con 3 particiones.**

@Bean  
public NewTopic adviceTopic*() {* return new NewTopic*(*topicName, 3, *(*short*)* 1*)*;  
*}*

**Bokers y Cluster**

Un único servidor de Kafka se denomina Bokers. El Bokers recibe mensajes de los “Producer”.

Producer:

**Ejemplo de enviar topic al Bokers.**

@Autowired  
private KafkaTemplate*<*Object, Object*>* KafkaTemplate;

KafkaTemplate.send*(*"topic1", new Foo1*(*what*))*;

También atiende a los consumidores, respondiendo a las solicitudes de recuperación de particiones y respondiendo con los mensajes que se han enviado al disco.

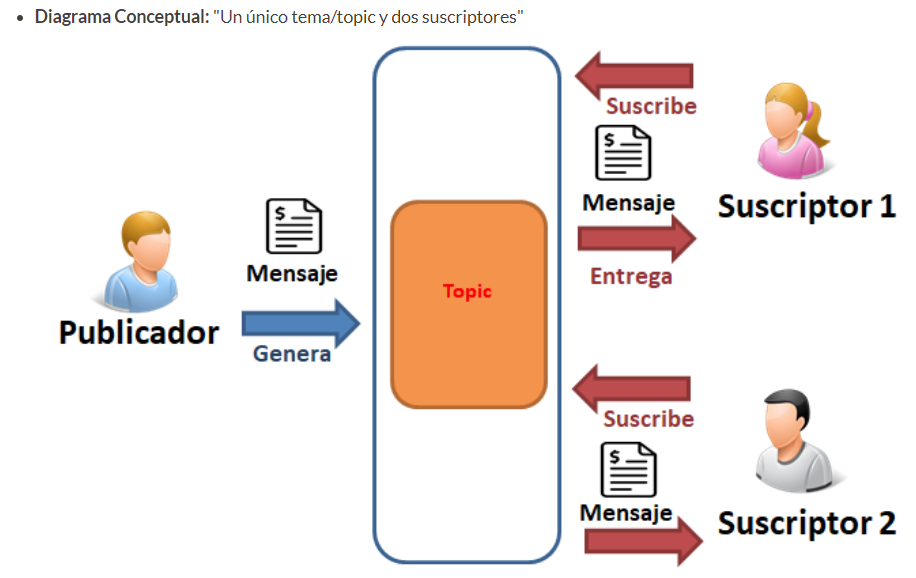
Consumer:

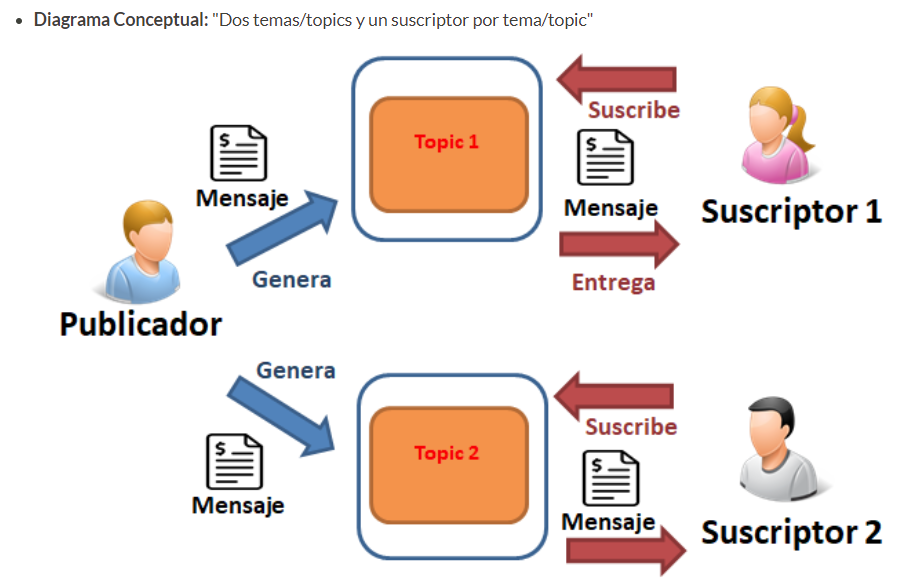
**Ejemplo de consumir un topic al Bokers.**

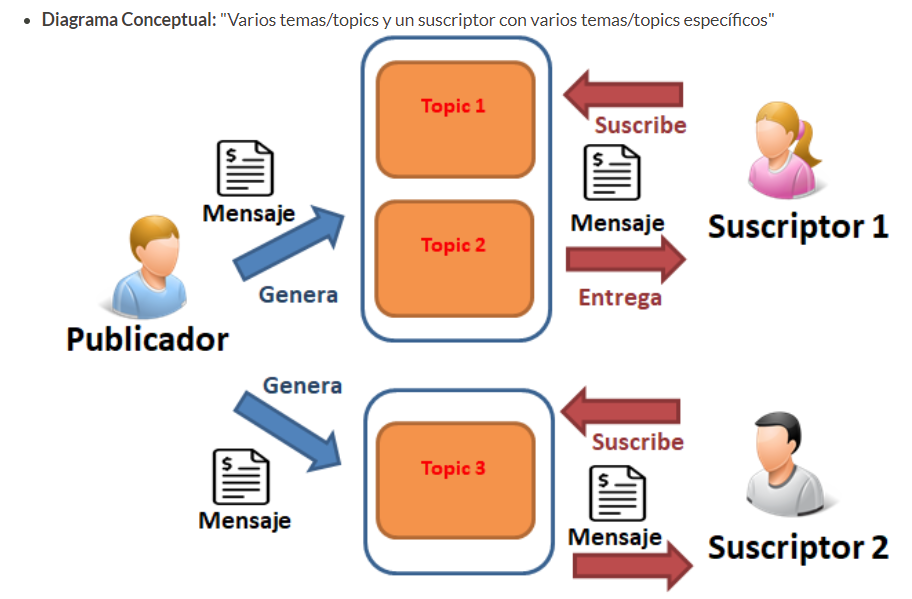
@KafkaListener*(*id = "fooGroup", topics = "topic1"*)*public void listen*(*Foo2 foo*) {* logger.info*(*"Received: " + foo*)*;  
*}*

Los Bokers de Kafka están diseñados para operar como parte de un clúster.

Para fortalecer los conceptos se muestras a continuación varios diagramas conceptuales:







**Ventajas de KAFKA**

* Kafka puede proporcionar un **alto rendimiento** mientras maneja **múltiples productores que** emiten conjuntos de datos para un solo tema o múltiples temas. Esto hace que Kafka procese eventos / mensajes masivos desde sistemas front-end que registran páginas vistas, seguimiento del mouse o comportamiento del usuario.
* Kafka administra **múltiples consumidores** para leer cualquier flujo de mensajes sin interferir entre sí. Cada mensaje se puede leer N número de veces porque los **mensajes son duraderos** .
* Los mensajes duraderos también significan que el consumidor puede trabajar con **datos históricos**. Sin embargo, Kafka también admite **el procesamiento en tiempo real**.
* Kafka ofrece un rendimiento excelente y es capaz de gestionar millones de registros por segundo dentro del hardware o la infraestructura de soporte.

**Kafka destaca por:**

* *Plataforma para la nueva generación de aplicaciones distribuidas.*
* *Tiene mucha popularidad en la actualidad (muchas grandes empresas lo tienen incorporado en sus arquitecturas: Netflix, Microsoft, Bancos, aseguradoras, empresas ticketing, etc).*
* *Tiene una orientación hacia el mundo "Big Data" aunque también destaca en otros ámbitos como la comunicación de aplicaciones, la ejecución de procesos batch.*
* *Alternativa a JMS, AMQP y RabbitMQ cuando se maneja volúmenes importantes de datos/información y se requiere un gran capacidad de respuesta -> mejora sus características más importantes.*
* *Facilita el trabajo con otras tecnologías como: Flume / Flafka, Spark Streaming, Storm, HBase, Flink y Spark para ingerir, analizar y procesar datos en tiempo real.*
  + *Aplicaciones online: Apache Solr, Logstar.*
  + *Procesamiento streams: SAMZa, Storm, Flink, Wallarro.*
  + *Procesamiento offline: Hadoop.*
* *Dispone de un API de "Producer": Facilita que una aplicación publique una secuencia de mensajes en uno o más topics de diferentes formas.*
* *Dispone de un API de "Consumer": Facilita que una aplicación pueda suscribirse a uno o más topics y así poder procesar la secuencia de mensajes.*
* *Dispone de un API de "Streams": Facilita procesar un flujo consumiendo un flujo de entrada de uno o más topics y produciendo un flujo para uno o más topics de salida.*
* *Dispone de un API de "Connector": Facilita implementar/ejecutar productores o consumidores reutilizables con el objetivo de conectar topics con aplicaciones o sistemas de datos existentes.*

Casos de uso o aplicaciones de Kafka

Puede ayudar a mejorar la forma de trabajo:

* Uso en arquitecturas de streaming de datos en tiempo real.
* Envió/intercambio/transformación de datos entre aplicaciones.
* Manejar flujos de datos en tiempo real (por ejemplo: procesamiento de streams, agregación de registros, ingesta de datos en Spark, etc.)
* Recopilar grandes datos (por ejemplo: logs, changelogs, tracking actividad de una web, monitoreo de métricas, seguimiento de clicks, etc), lo que facilita el aprendizaje y mejora
* Procesamiento de eventos complejos (CEP)
* Funcionamiento como una "cache" persistente
* Proporciona durabilidad en microservicios in-memory
* Proporciona eventos a sistemas IoT

### **Zookeeper**

Se trata de un **servicio centralizado imprescindible para el funcionamiento de Kafka**, al cual **envía notificaciones en caso de cambios** como: creación de un nuevo topic, caída de un broker, levantamiento de un broker, borrado de topics, etc.

Ofrece a sistemas externos capacidades de **almacenamiento clave-valor, sincronización y registro de nombres**, entre otros.

Para gestionar un clúster de Kafka, **Zookeeper almacena información del estado del clúster**: detalles de los topics como el nombre, las particiones, las réplicas y los grupos de consumidores.

En el momento en el que Zookeeper detecta que uno de los brokers de Kafka está caído, realiza las siguientes acciones:

1. Elige un nuevo broker para tomar el lugar del broker caído.
2. Actualiza los metadatos para la distribución de carga de los productores y los consumidores para que no exista pérdida de servicio.

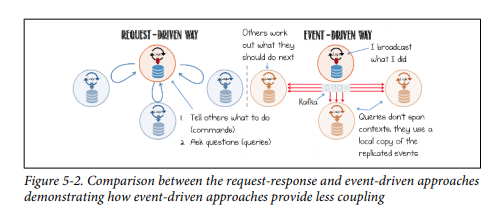
Tras estas acciones, se pueden volver a escribir y leer mensajes con normalidad.

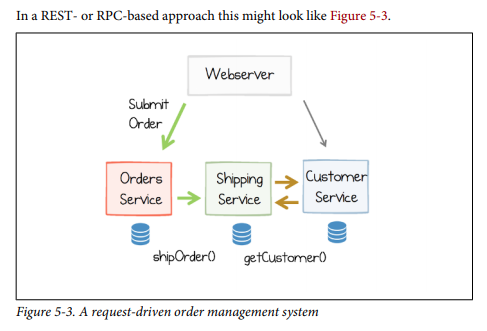
Quienes usan Kafka.

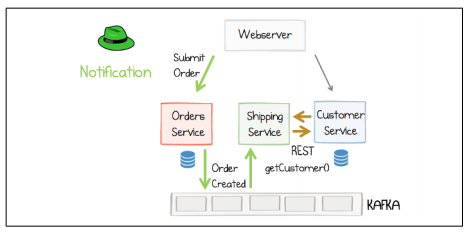
<https://kafka.apache.org/powered-by>

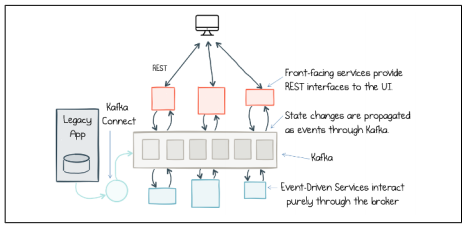
La **Arquitectura dirigida por eventos**, *Event-driven architecture* o EDA,

Es un patrón de [arquitectura software](https://es.wikipedia.org/wiki/Arquitectura_software) que promueve la producción, detección, consumo de, y reacción a [eventos](https://es.wikipedia.org/wiki/Evento_estad%C3%ADstico).









Un ejemplo muy simple de servicios controlados por eventos, los datos se importan desde un

aplicación heredada a través de la API de Kafka; Los servicios orientados al usuario proporcionan API REST para

la interfaz de usuario; los cambios de estado se registran en Kafka como eventos.

**Kakfa desde entorno empresarial:**

Confluentic

docker-compose up -d

Kafka Clients

Suponiendo que Kafka está ejecutándose en un clúster, aquí hay algunos comandos que describen los pasos típicos de Kafka.

**Create a topic**

docker-compose exec broker kafka-topics --create --topic example-topic --bootstrap-server broker:9092 --replication-factor 1 --partitions 1

**Produce messages**

docker-compose exec broker bash

kafka-console-producer --topic example-topic --broker-list broker:9092

**Consume messages**

docker-compose exec broker bash

kafka-console-consumer --topic clasee-topic --bootstrap-server broker:9092

kafka-console-consumer --topic example-topic --bootstrap-server broker:9092 --from-beginning

Iniciando Spring KAFKA.

*<*dependency*>  
 <*groupId*>*org.springframework.kafka*</*groupId*>  
 <*artifactId*>*spring-kafka*</*artifactId*>  
</*dependency*>*

Boot se encarga de la mayor parte de la configuración. Cuando usamos un corredor local, las únicas propiedades que necesitamos son las siguientes:

**Ejemplo 1. application.properties**

spring.kafka.consumer.group-id=foo

spring.kafka.consumer.auto-offset-reset=earliest

Se necesita la primera propiedad porque se usa la administración de grupos para asignar particiones de temas a los consumidores, por lo que se necesita un grupo. La segunda propiedad garantiza que el nuevo grupo de consumidores reciba los mensajes que se envía, porque el contenedor podría iniciarse después de que se hayan completado los envíos.

Repasando:

¿Para qué se usa Apache Kafka?

Apache Kafka se usa como sistema de intermediación de mensajes entre aplicaciones. Varios suscriptores y publicadores externos pueden leer los mensajes almacenados en Kafka y de esta forma abstraer la complejidad asociada al tratamiento de mensajes de la aplicación.

¿Es Apache Kafka una base de datos?

Kafka es un sistema distribuido que trata los datos como un flujo continuo. Aunque cada vez se agrega mayor funcionalidad sobre este sistema no es una base de datos corriente ni tiene los mismos usos ni aplicaciones. Podemos considerar Kafka como una herramienta que puede complementar a una base de datos.

¿Qué es un bróker de Kafka?

Un bróker de Kafka es un nodo con Kafka instalado que forma parte del clúster. Permite a los suscriptores leer mensajes y se comunica con los demás brókers. Para sincronizarse también puede usar Apache Zookeeper.

¿Qué ventajas tiene Apache Kafka?

Apache Kafka permite desacoplar aplicaciones entre sí que necesiten comunicarse mediante paso de mensajes en tiempo real. Es un sistema escalable y con baja latencia, lo que hace una solución ideal para tratar grandes cantidades de datos en sistemas Big Data.

¿Es Kafka un sistema asíncrono?

Kafka permite implementar a las aplicaciones un sistema de comunicación asíncrono con el patrón publicador-subscriptor. Los productores de mensajes no deben esperar a que se consuman para continuar sus operaciones.