Mensajería Distribuida

KAFKA

Miguel Goyena

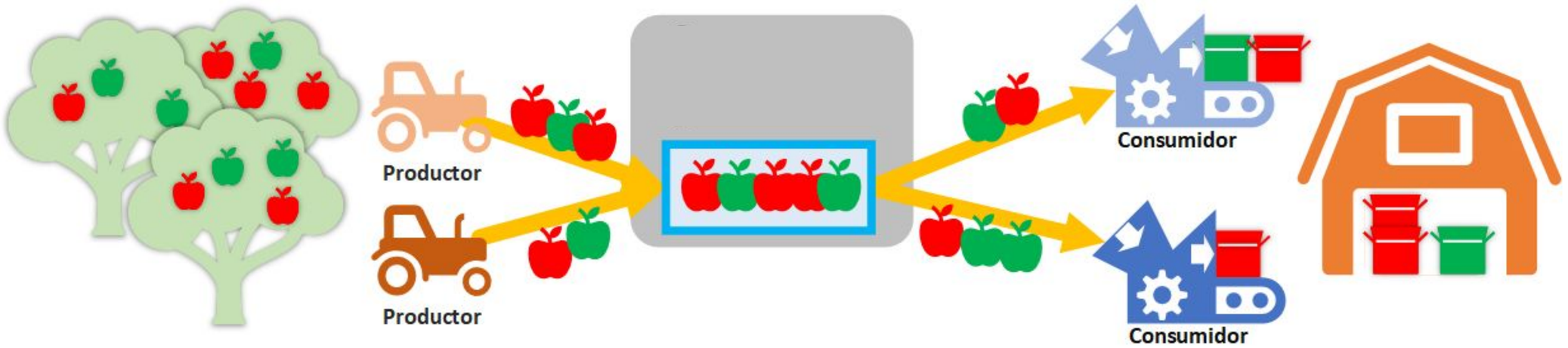
Centro Integrado
Cuatrovientos

ÍNDICE

Patrón Productor-Consumidor
Que és

Empezaremos con un ejemplo simple.

Por un lado, tenemos tractores robotizados que recogen de nuestro campo de manzanos las manzanas (rojas y verdes), por otro lado, tenemos varias máquinas empaquetadoras de manzanas que esperan a las manzanas para ser empaquetadas y enviarlas al almacén.



Patrón Productor-Consumidor
Que és

Pueden producirse 2 tipos de problemas:

- Los empaquetadores son más lentos que los productores.
- Tenemos muchos empaquetadores y están ociosos.

Para solventar estos problemas, necesitamos algo entre medio del productor y consumidor.

Ese intermediario es un sistema de colas de mensajes independiente, tiene las siguientes ventajas:

- Independencia entre productores y consumidores. Pueden ser procesos distintos!!!
- Es robusto, porque puede tener memoria y persistencia en los mensajes.
- Puede escalar, porque podemos montar todas las colas que necesitemos y replicarse a nuestro gusto. N Productores - M Consumidores

Que es Kafka
Características y quien lo utiliza

Kafka es una implementación de este intermediario, o sea un gestor de colas. Su origen es LinkedIN, pero muy pronto se convirtio en un SW Open Source con licencia Apache.
Se vende con las siguientes características:

- Es muy rápido. Por lo tanto es muy bueno en el procesamiento de mensajes en tiempo real.
- Persiste los mensajes y puede garantizar el orden de ellos.
- Permite implementar todos los patrones diferentes para mensajería distribuida. Varios Productores y Consumidores al mismo tiempo.
- Escala muy bien. O sea soporta gran cantidad de colas y mensajes sin perder productividad
- Es robusto. Tiene una arquitectura que permite alta disponibilidad y tolerante a fallos. Tiene un buen sistema de replicación

Este patrón e implementación es tan bueno que lo utilizan muchas compañías. Incluso muchos productos nube PaaS de mensajería tienen Kafka por detrás.

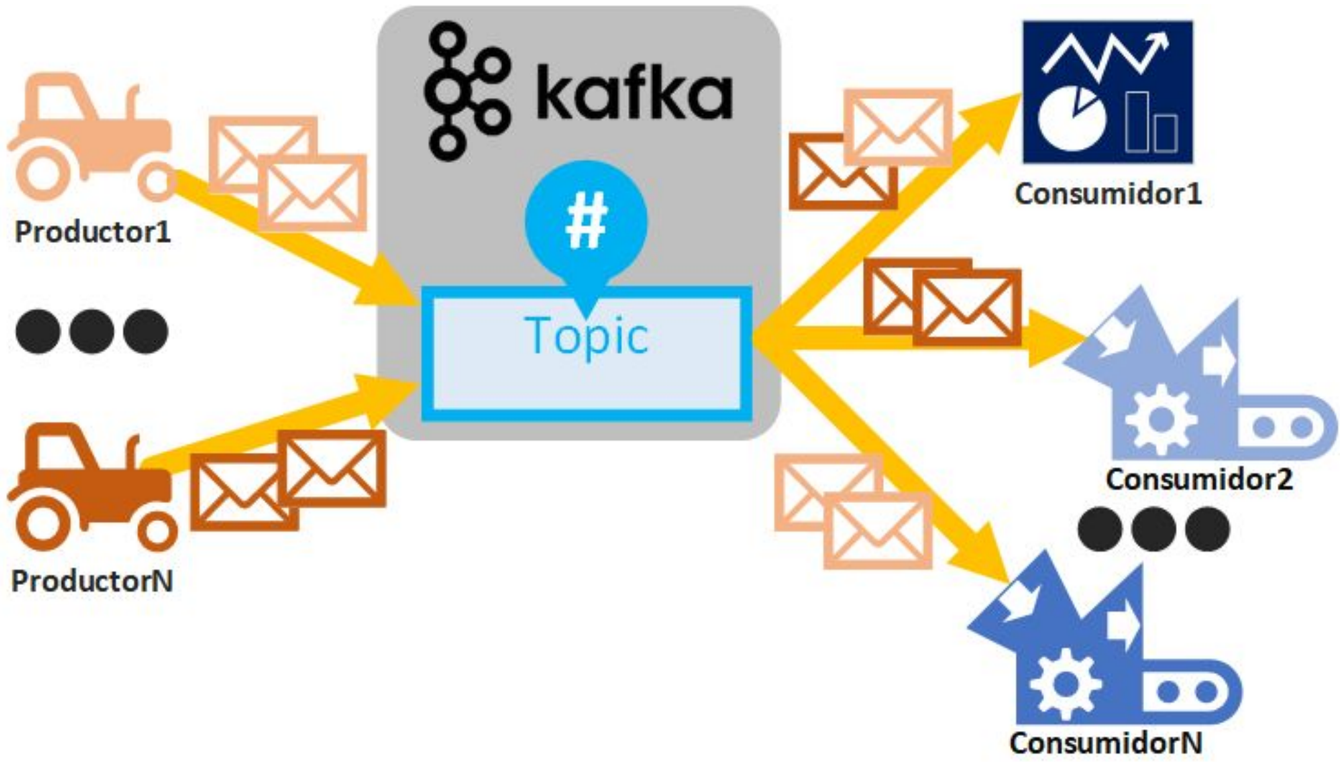
TOPIC: La unidad mínima para Kafka

Que és un Topic

Un **TOPIC** para Kafka es un contenedor de mensajes del mismo tipo. Es entonces mucho más que una cola de mensajes. Permite que haya muchos productores y muchos consumidores simultáneos.

Si seguimos nuestro ejemplo de la granja.

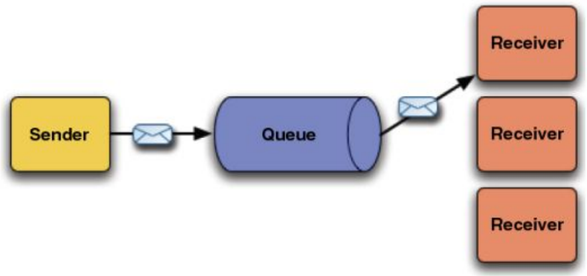
- Podemos tener muchos tractores en diferentes campos creando mensajes
- Podemos utilizar la información de los mensajes para diferentes propósitos.



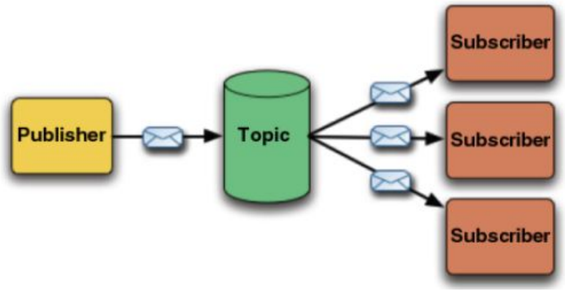
TOPIC: La unidad mínima para Kafka
Diferentes tipos de patrones a implementar

Debido a la flexibilidad que ofrecen los TOPIC de Kafka podemos implementar los patrones más conocidos dentro de Productor-Consumidor:

- **Colas de mensajes**: Cada consumidor consume el mensaje y el resto no lo reciben.



- **Publicador-Suscriptor**: Una aplicación envía mensajes de manera asíncrona a varios Consumidores. Pensemos en un TOPIC de Twitter, donde todos los suscriptores recibirán todos los mensajes.



TOPIC: La unidad mínima para Kafka
Puesta en marcha

Vamos entonces a instalar KAFKA

- Instalamos la última versión 4.1.1. Es un ZIP que debemos descomprimir en un directorio sin espacios en blanco y cerca de C:.. Por supuesto también tenemos la versión Docker. Necesitamos de la JDK > 17.

- Tenemos que crear el almacenamiento de los topics. Dentro de bin/windows:
  - *.\kafka-storage.bat random-uuid*: Genera un UUID, que nos guardamos.
  - *.\kafka-storage.bat format --standalone -t $KAFKA_UUID -c ..\..\config\server.properties*: Generamos el almacenamiento según la configuración

- Arrancamos el servicio: *.\kafka-server-start.bat ..\..\config\server.properties*

TOPIC: La unidad mínima para Kafka
Puesta en marcha

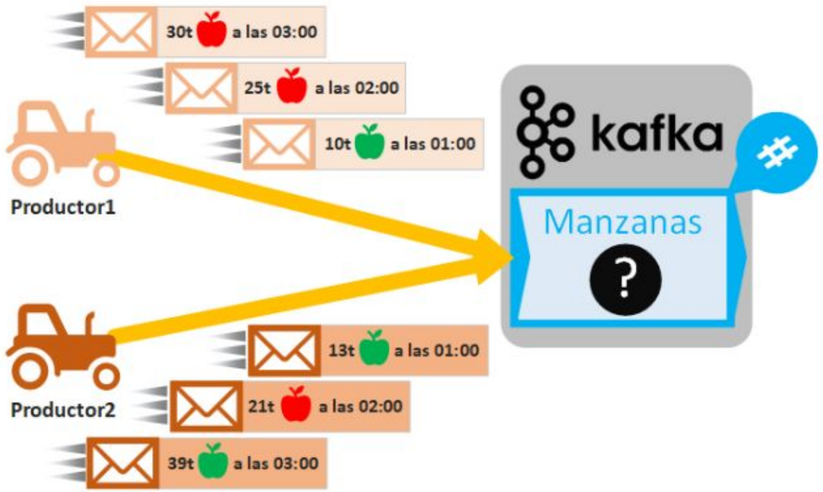
Vamos a crear nuestro primer Topic

- Al topic le llamaremos recolector-manzanas:
  - *.\kafka-topics.bat -create --topic recolector-manzanas --bootstrap-server localhost:9092*
- Podemos consultar el topic generado:
  - *.\kafka-topics.bat --describe --topic recolector-manzanas --bootstrap-server localhost:9092*

Productores
¿Que son?

Los productores son los creadores de los **mensajes** que enviamos al Topic. Esos mensajes pueden ser de cualquier formato (JSON, XML, etc), mientras sean serializables. Cada mensaje se compone de:

- Cuerpo del mensaje. Por ejemplo: 35 KG Manzanos rojas.
- Clave del mensaje: Si no lo proveemos se generará sólo y no tiene porque ser único. Tiene que ver con las particiones que veremos más adelante.



Un productor recoge los datos de una **fuente de datos (source)**

Puede ser Sensores IOT, consulta APIS, Otros topics, Streaming Datos, Logs, métricas, etc...

Productores
Creamos productores: Script

Un primer ejemplo sería crear un productor mediante los scripts que nos da Kafka

```
.\kafka-console-producer.bat --topic recolector-manzanas
--bootstrap-server localhost:9092
```

Productores
Creamos productores: JAVA

Pero es mucho más interesante si creamos un programa JAVA que produce mensajes, simulando cada uno de los tractores!!

1. Creamos un nuevo proyecto maven. Por ejemplo con el arquetipo: maven-archetype-quickstart

2. Añadimos la dependencia de KAFKA:

3. Creamos una clase RecolectorManzana con un Main

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>4.1.1</version>
</dependency>
```

Productores
Creamos productores: JAVA

```java
public class RecolectorManzana {

   private static final String KAFKA_SERVER_IP_PORT = "127.0.0.1:9092";
   private static final String MANZANAS_TOPIC = "recolector-manzanas";

   public static void main(String[] args) {

      KafkaProducer<String, String> productor = null;

      try {

         Properties props = new Properties();
         props.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, KAFKA_SERVER_IP_PORT);
         props.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
         props.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

         productor = new KafkaProducer<>(props);

         productor.send(new ProducerRecord<>(MANZANAS_TOPIC, "45 KG - Manzanas Rojas"));

      }
      catch (Exception e){
         System.err.println("Excepcion general en el Recolector de Manzanas: "+e.getMessage());
         e.printStackTrace();
      }
      finally {
         if (productor != null) {
            productor.flush();
            productor.close();
         }
      }
   }
}
```

- Definimos las propiedades de nuestro productor. Serializador y Servidor Kafka
- Mandamos nuestro mensaje, en este caso sin clave.
- No olvidarnos al acabar de cerrar nuestro productor.

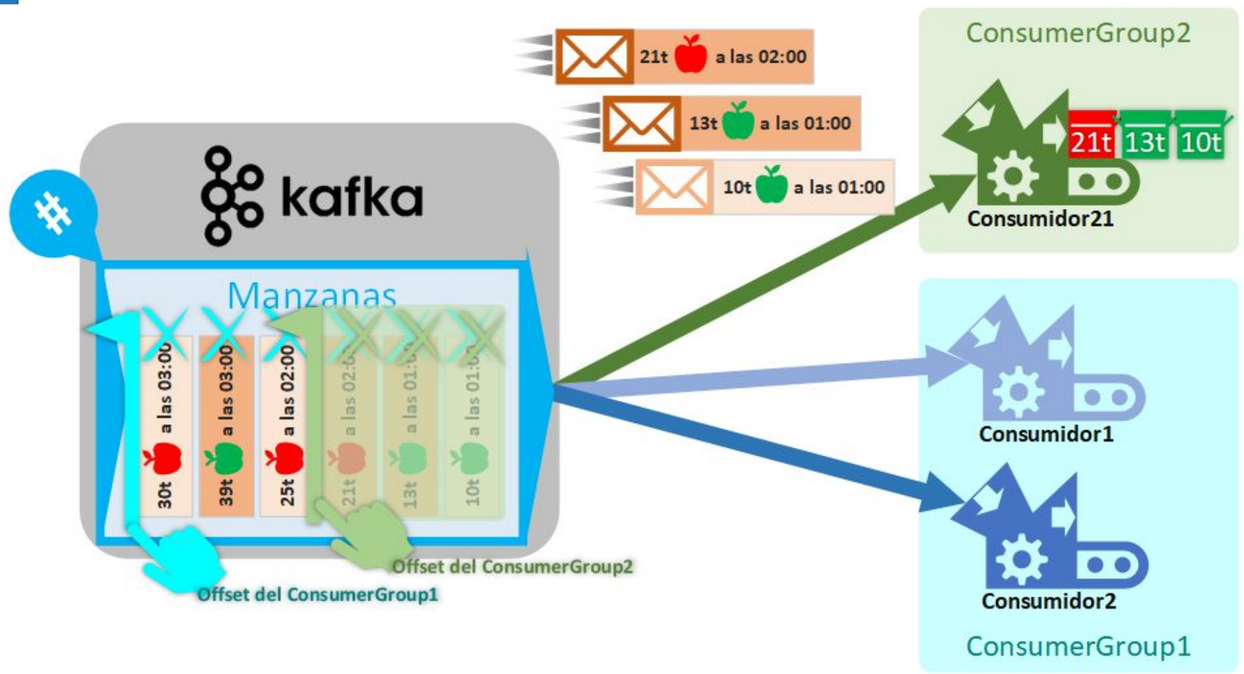
Consumidores
¿Que son?

Los consumidores son los que procesos que recogen los mensajes de un Topic de Kafka. Parece sencillo, pero tiene una customización que lo hace muy flexible.

CASO 1 (Cola de mensajes): Un consumidor consume todos los mensajes de un topic. Ojo que los va a recoger en el orden de llegada.

Consumidores
¿Que son?

CASO 2 (Suscribers): Varios consumidores consumen todos los mensajes del TOPIC. Para ello hay que configurar por cada consumidor su **consumer-group**. Todos los consumidores de un mismo consumer-group consumen los mensajes de un topic en Orden!!



Kafka mantiene un contador separado por cada uno de los consumer-group, llamado **OFFSET**.

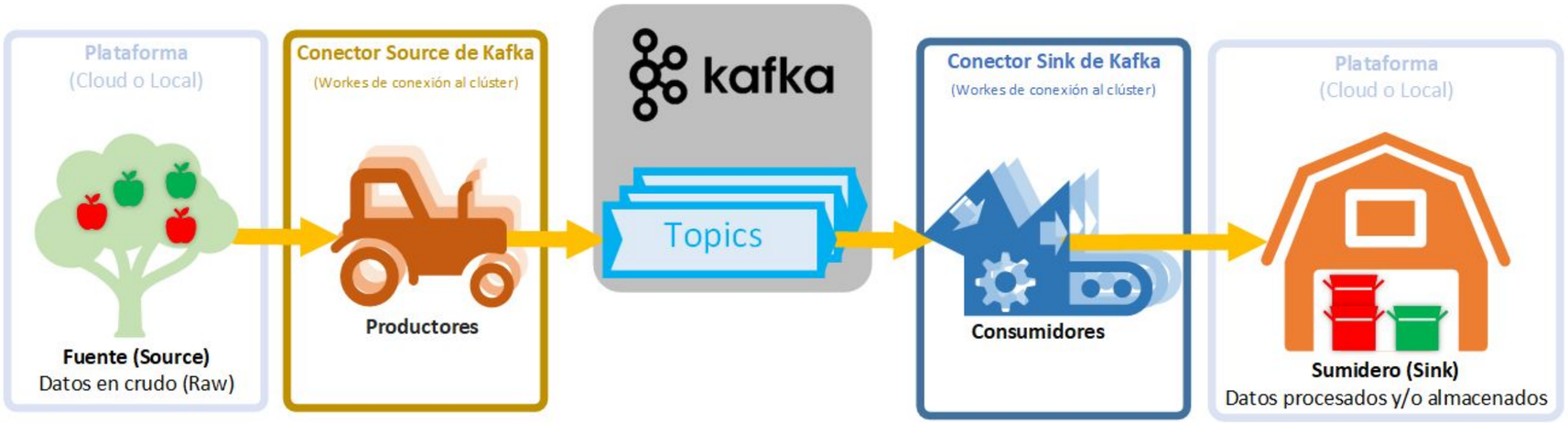
Su función es tener constancia del ritmo de los consumidores, porque en KAFKA los mensajes por defecto no se borran aunque se consuman.

Consumidores
Que se hace con los mensajes

Se considera que los mensajes sirve para un propósito. Entonces se habla de **SINKS** o **sumideros**. Como ejemplo están los siguientes:

1. BBDD
2. Analiticas y auditorias
3. Sistemas de mensajes
4. Procesados ETL.

Tanto SOURCES, como SINKS se consideran **conectores** para KAFKA. Por lo tanto el flujo de los mensajes es:



Consumidores
Creamos consumidores: Script

Un primer ejemplo sería crear un consumidor mediante los scripts que nos da Kafka

```
.\kafka-console-consumer.bat --topic recolector-manzanas
--from-beginning --bootstrap-server localhost:9092
```

Consumidores
Creamos consumidores: JAVA

Pero es mucho más interesante si creamos un programa JAVA que consumen los mensajes, simulando cada uno de los empaquetadores

1. Utilizamos el mismo proyecto Maven.

2. Creamos una nueva clase EmpaquetadorManzanas con un Main.

El proceso de los consumidores es independiente al proceso de los productores.

Consumidores
Creamos consumidores: JAVA

```
public class EmpaquetadorManzanas {

    private static final String KAFKA_SERVER_IP_PORT = "127.0.0.1:9092";
    private static final String MANZANAS_TOPIC = "recolector-manzanas";
    private static final String CONSUMER_GROUP = "empaquetadora_sanadrian";

    public static void main(String[] args) {

            KafkaConsumer<String, String> consumidor = null;
            boolean empaquetadorAbierta = true; // Abrimos la empaquetadora

            try {

                // Definimos las propiedades
                Properties props = new Properties();
                props.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, KAFKA_SERVER_IP_PORT);
                props.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
                props.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
                props.setProperty(ConsumerConfig.GROUP_ID_CONFIG, CONSUMER_GROUP);
                props.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

                //Creamos el consumidor
                    consumidor = new KafkaConsumer<>(props);
                    consumidor.subscribe(Collections.singleton(MANZANAS_TOPIC));
                    // Consumimos, mientras la empquetadora abierta
                    while (empaquetadorAbierta) {
                        ConsumerRecords<String, String> mensajes = consumidor.poll(Duration.ofSeconds(1));
                        for (ConsumerRecord<String, String> mensaje: mensajes) {
                            System.out.println(mensaje.value());
                        }
                    }
            }
            catch (Exception e){
                    System.err.println("Excepcion general en el Recolector de Manzanas: "+e.getMessage());
                    e.printStackTrace();
            } finally {
                    if (consumidor != null) {
                            consumidor.close();
                    }
            }
    }
}
```

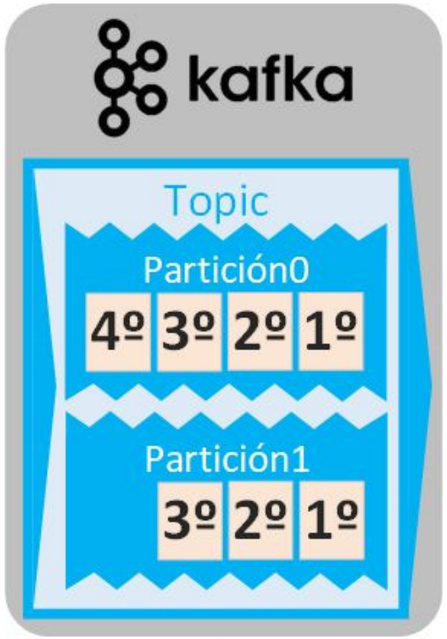
- Definimos las propiedades de nuestro productor. Serializador y Servidor Kafka
- Creamos un consumidor para un consumer-group
- Hacemos un bucle para consumir todos los mensajes, con cada mensaje decidimos que hacemos.

Dividir un TOPIC en particiones
Particiones, que son.

Una de las formas de escalar y poder distribuir los mensajes de un TOPIC para que se puedan consumir de forma concurrente es dividir los TOPIC por particiones.

Las **particiones** son realmente las colas de un mismo TOPIC, por eso KAFKA utiliza el nombre de Topics porque son algo más que colas.

OJO: Nunca se guarda el orden entre particiones

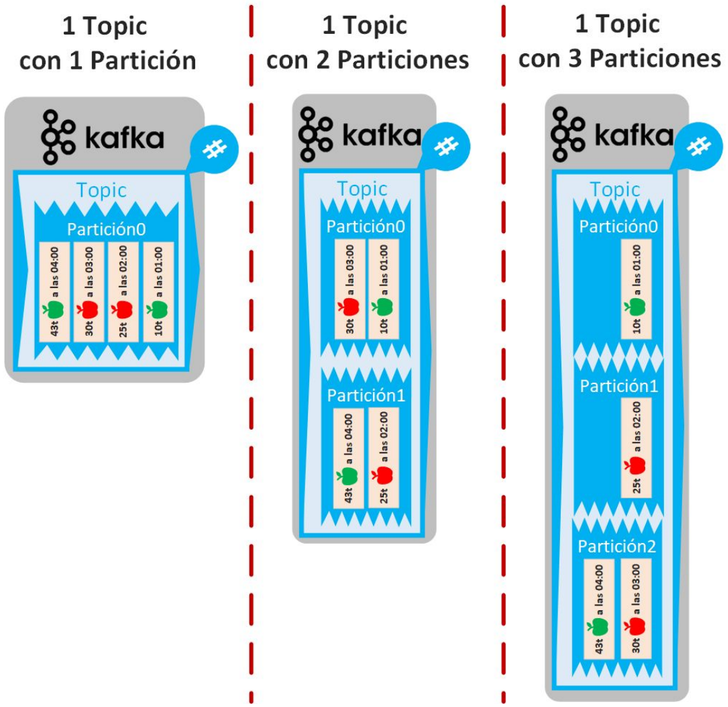


Las particiones se pueden hacer por varias razones:

1. Poder distribuir los Topics y así hacerlo más escalable.
2. Poder crear procesos consumidores paralelos dentro de un topic (consumer-group). + Rapidez

Dividir un TOPIC en particiones
Particiones, criterios de partición

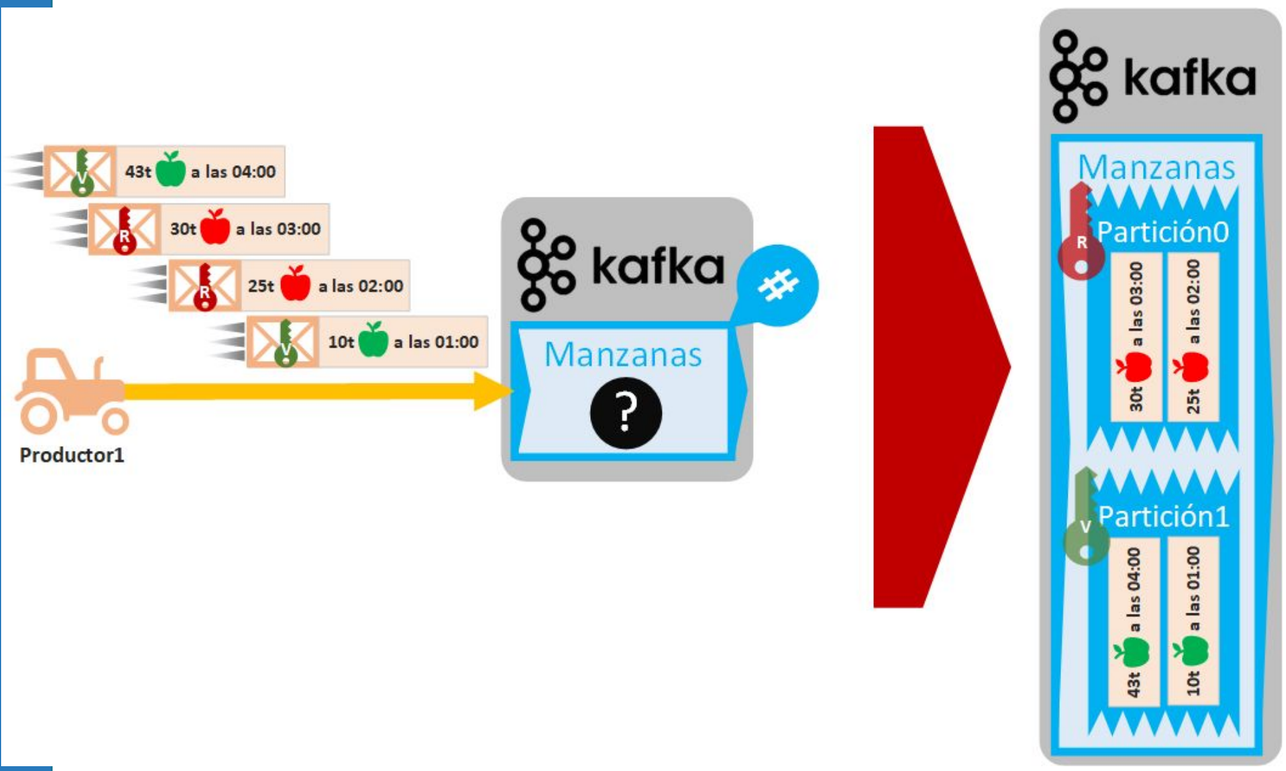
Teniendo en cuenta que el orden entre particiones no se garantiza y que podemos tener 1 consumidor por cada partición, podemos utilizar criterios diferentes para particionar:



CASO 1: Podemos definir round robin:

1. Definimos un máximo de particiones.
2. Los mensajes se distribuyen de forma equitativa.
3. Hay que procurar siempre que las particiones sean equitativas, para que escale bien.

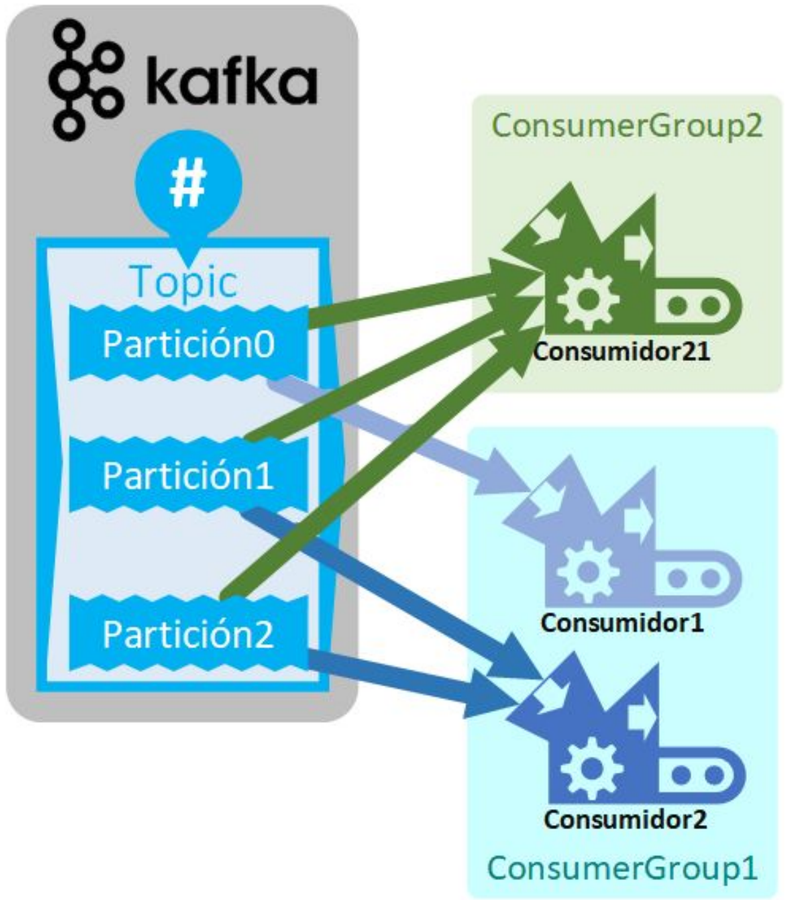
Dividir un TOPIC en particiones
Particiones, criterios de partición



CASO 2: Podemos definir un criterio por algún parámetro del mensaje, que utilizamos con CLAVE.

1. En este caso utilizamos si las manzanas son rojas o verdes.
2. Así podemos definir un empaquetador para Verdes y otro para las Rojas
3. Pero OJO, todos son manzanas!!

Dividir un TOPIC en particiones
Particiones, criterios de partición



CASO 3: Que pasa si juntamos particiones y consumer groups

1. Aquí ya la flexibilidad es total, podemos jugar.
2. Un proceso por consumer-group y partición concreta.
3. Los OFFSET nos sirven para gestionar todo esto.

Conclusiones

KAFKA es mucho más que una cola de mensajes. Es un sistema flexible con el que podemos jugar sabiendo que los conceptos clave son:

- TOPIC
- PARTICIÓN
- CONSUMER-GROUP
- PRODUCTOR
- CONSUMIDOR