Especialización en
Inteligencia Artificial y Big Data

5075. Big Data Aplicado

Módulo - 3

# Apache Kafka

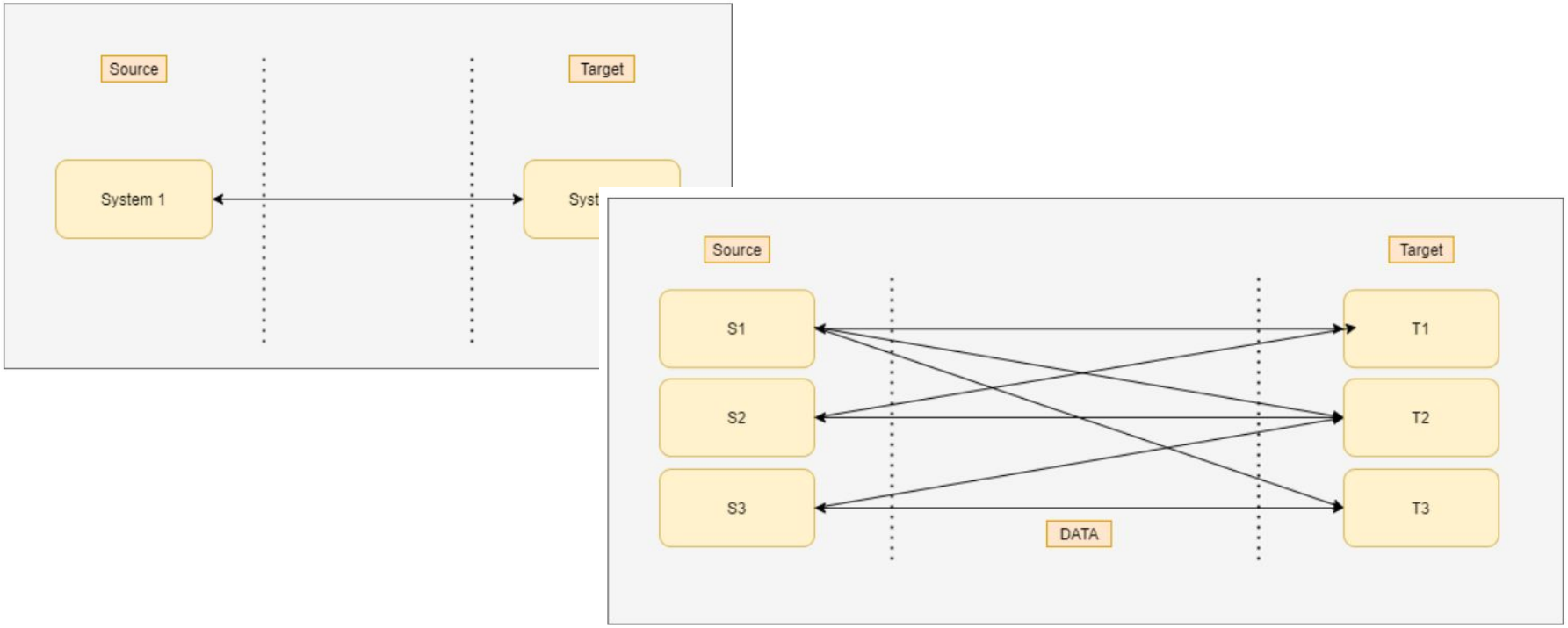

Vicent P. Tortosa Lorenzo

Ismael Torres Boigues

# Índice

# Problema a resolver

El intercambio de datos entre sistemas



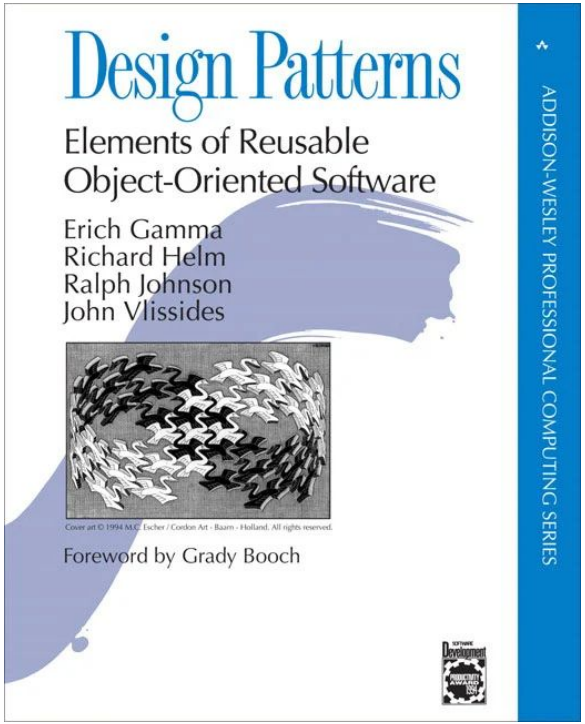
# Consideraciones de los datos en plataformas big data

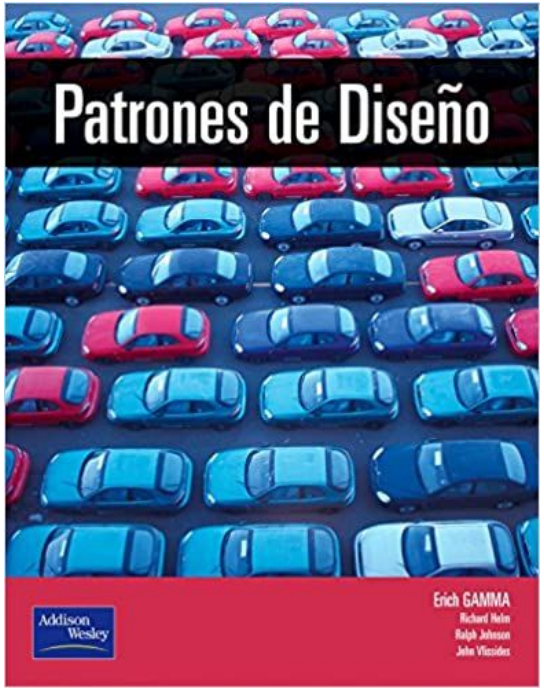
- Datos no estructurados
- Baja calidad del dato
- Temporalidad
- Persistencia de los datos
- Semántica
- procesamiento Síncronamente  & asíncronamente

# Paso de información de aplicaciones big data

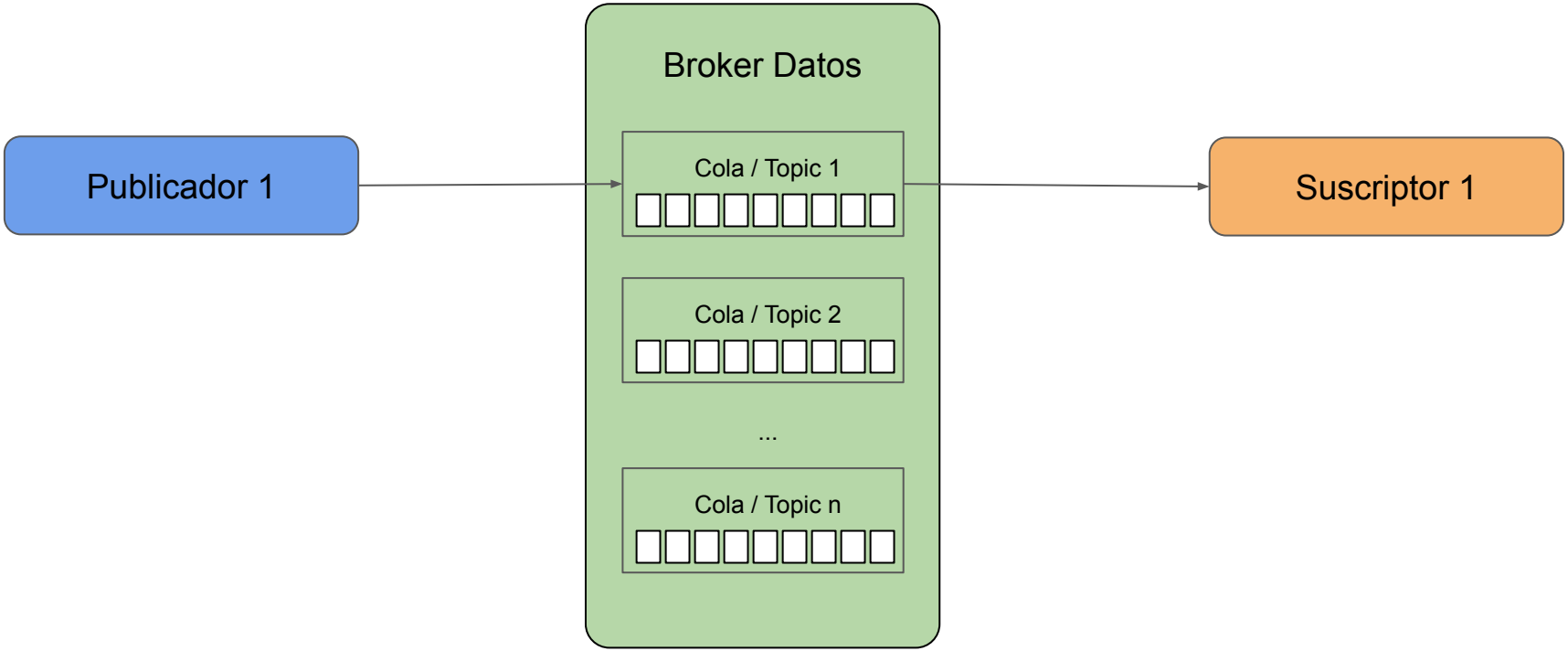
- Las soluciones big data están compuestas por diferentes componentes distribuidos que necesitan intercambiar información.
- La mensajería asíncrona es una forma de desacoplar remitentes de consumidores. Evita bloqueos a la espera de respuesta.

# Patron:  Publish Subscribe

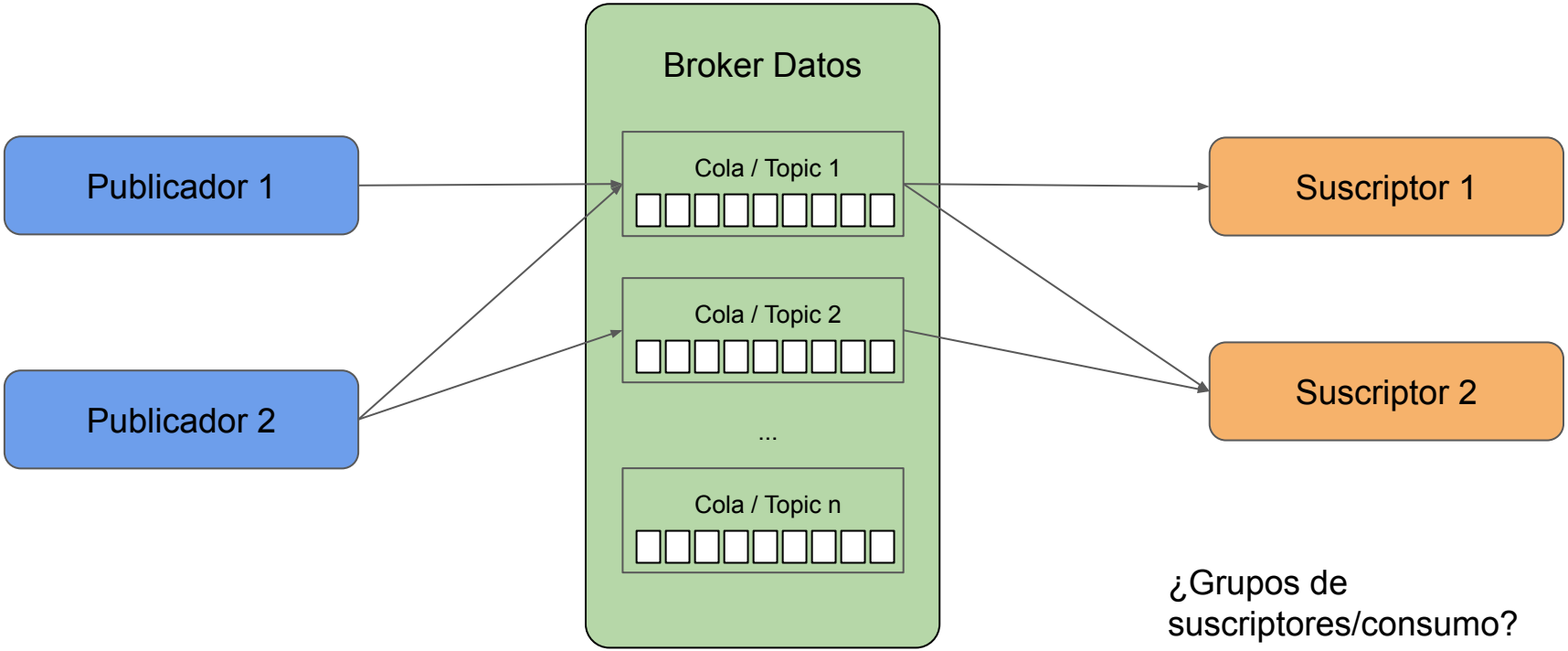


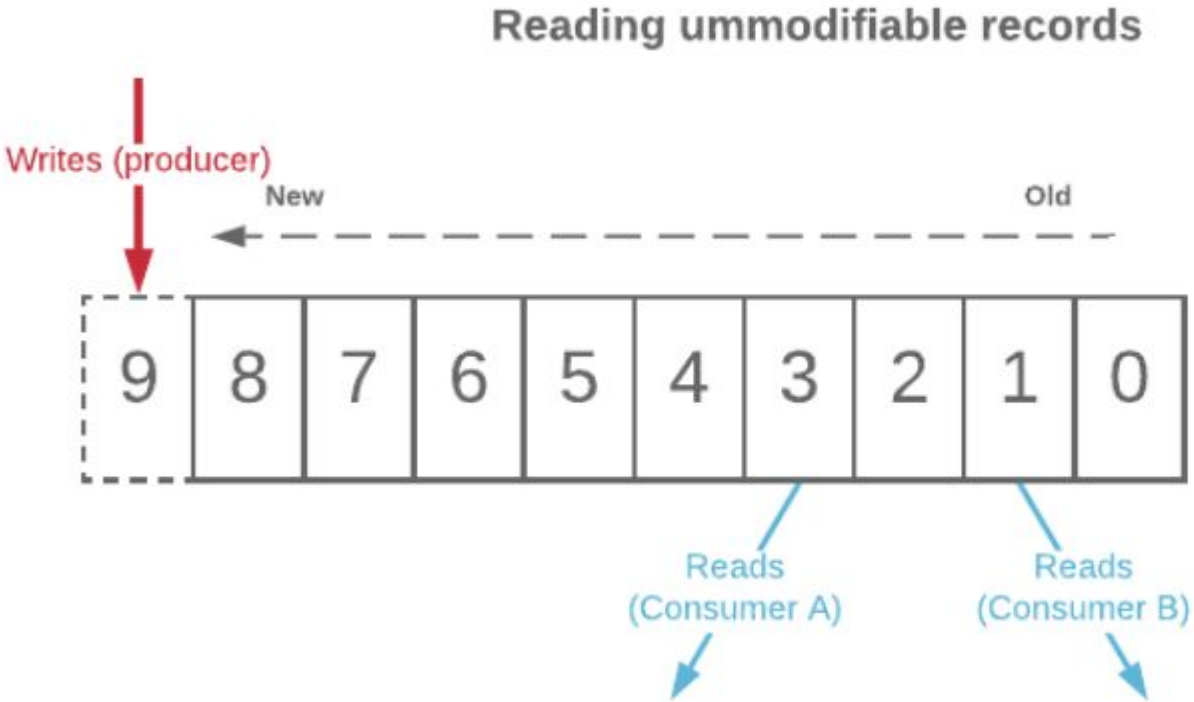
# ¿Cómo funcionan las colas de mensajes?  1 a 1



# ¿Cómo funcionan las colas de mensajes?




¿Grupos de suscriptores/consumo?


Reading ummodifiable records
Writes (producer)
New
Old
9
8
7
6
5
4
3
2
1
0
Reads
(Consumer A)
Reads
(Consumer B)

# Consideraciones de las colas de mensajes

- Estrategias de entrega:
  - **"como máximo una vez" (at most once):**
  - **"al menos una vez" (at least once):**
  - **"exactamente una vez" (exactly once)**
- Estrategias de ordenación:
  - **Sin ordenación (No ordering).**
  - **Ordenación por partición (Partitioned ordering):**
  - **Ordenación Global (Global Order)**

# Consideraciones de las colas de mensajes

- **Disponibilidad (Availability)**: Capacidad para estar el mayor tiempo posible trabajando o activo.
- **Transaccionalidad (Transaction)**: Capacidad para agrupar acciones o elementos en unidades atómicas.
- **Escalabilidad (Scalability)**: Capacidad de evolucionar con el objetivo de poder dar soporte a una mayor cantidad de acciones
- **Rendimiento (Throughput)**: cantidad (nº de bytes) de elementos por unidad de tiempo con los que puede trabajar
- **Latencia (Latency)**: tiempo requerido para procesar un elemento

# Consideraciones de las colas de mensajes

- **Desacoplamiento (Decoupling)**:
  - **Entidad (Entity)**: Los productores y consumidores  no se conocen.
  - **Tiempo (Time)**: Los productores y consumidores  no necesitan participar activamente
  - **Sincronización (Synchronization)**: Si los hilos de ejecución o los clientes requieren algún tipo de bloqueo síncrono
- **Enrutamiento Lógico (Routing logic)**: Capacidad por la que un dato que sale de un productor acaba en un consumidor concreto.
  - **Tema (Topic)**: Los datos se clasifican en temas y los suscriptores sólo reciben datos relacionados con esos temas.
  - **Contenidos (Content)**: Los datos se clasifican en base a una o varias propiedades y los suscriptores pueden establecer filtros o restricciones específicas sobre un grupo de datos.

# Ventajas e inconvenientes de las colas

## Ventajas

- Desacopla los subsistemas que necesitan comunicarse.
- Aumenta la escalabilidad y mejora la capacidad de respuesta del remitente.
- Mejora la confiabilidad.
- Permite el procesamiento diferido o programado.
- Permite una integración más sencilla entre sistemas que utilizan diferentes plataformas
- Facilita flujos de trabajo asincrónicos en toda la empresa.
- Mejora la capacidad de prueba. Tanto los canales como los mensajes se pueden inspeccionar.

## Problemas y consideraciones

- Control de suscripciones.
- Seguridad.
- Subconjuntos de mensajes.
- Comunicación bidireccional.
- Orden de los mensajes.
- Prioridad del mensaje.
- Mensajes dudosos.
- Mensajes repetidos.
- Expiración de mensajes.
- Programación de envío/recepción mensajes.

# Software de colas de mensajes  / Broker de datos





# kafka & kinesis


| | Apache Kafka | Amazon Kinesis |
|---|---|---|
| Concepts | Kafka Streams | Kinesis Analytics |
| Stream of records container | Topic | Stream |
| Data Stored in... | Kafka Partition | Kinesis Shard |
| Unique ID of a record | Offset number | Sequence number |
| Ordering under... | Partition level | Shard level |
| Features | | |
| SDK Support | Kafka SDK supports Java | AWS SDK supports Android, Java, Go, .NET |
| Configuration & Features | More control on configuration and better performance | Number of days/shards can only be configured |
| Reliability | The replication factor can be configured | Kinesis writes synchronously to 3 different machines/data-centers |
| | | Kinesis writes each message synchronously to 3 different machines |

**AWS MSK (kafka Administrado)**

**Ref:** [https://www.softkraft.co/aws-kinesis-vs-kafka-comparison/](https://www.softkraft.co/aws-kinesis-vs-kafka-comparison/)

# ¿Qué es Apache Kafka?

- Apache Kafka se trata de un sistema de mensajes **"publish / subscribe"** Open Source basado en una arquitectura P2P (arquitectura Peer to Peer). http://kafka.apache.org
- *Distributed Messaging Queue*
- Tiene mucha popularidad y es una pieza clave en las arquitecturas BIG DATA + procesamiento tiempo real
- Programado en Java + se ejecuta en JVM
- API (producer, consumer,streams, connector) → TCP múltiples lenguajes
- Características
  - Estructura de datos ordenada y persistente
  - Sólo permite añadir elementos a la cola por el final
  - No se pueden modificar ni borrar registros (se pueden invalidar según ciertos criterios)
  - Proporciona procesamiento determinista
  - Evita pérdida de datos + réplicas en cluster
  - Alto rendimiento + clustering y escalado horizontal
  - Seguridad: SSL + autenticación zookeeper + cifrado
  - KSQL

# Conceptos Kafka

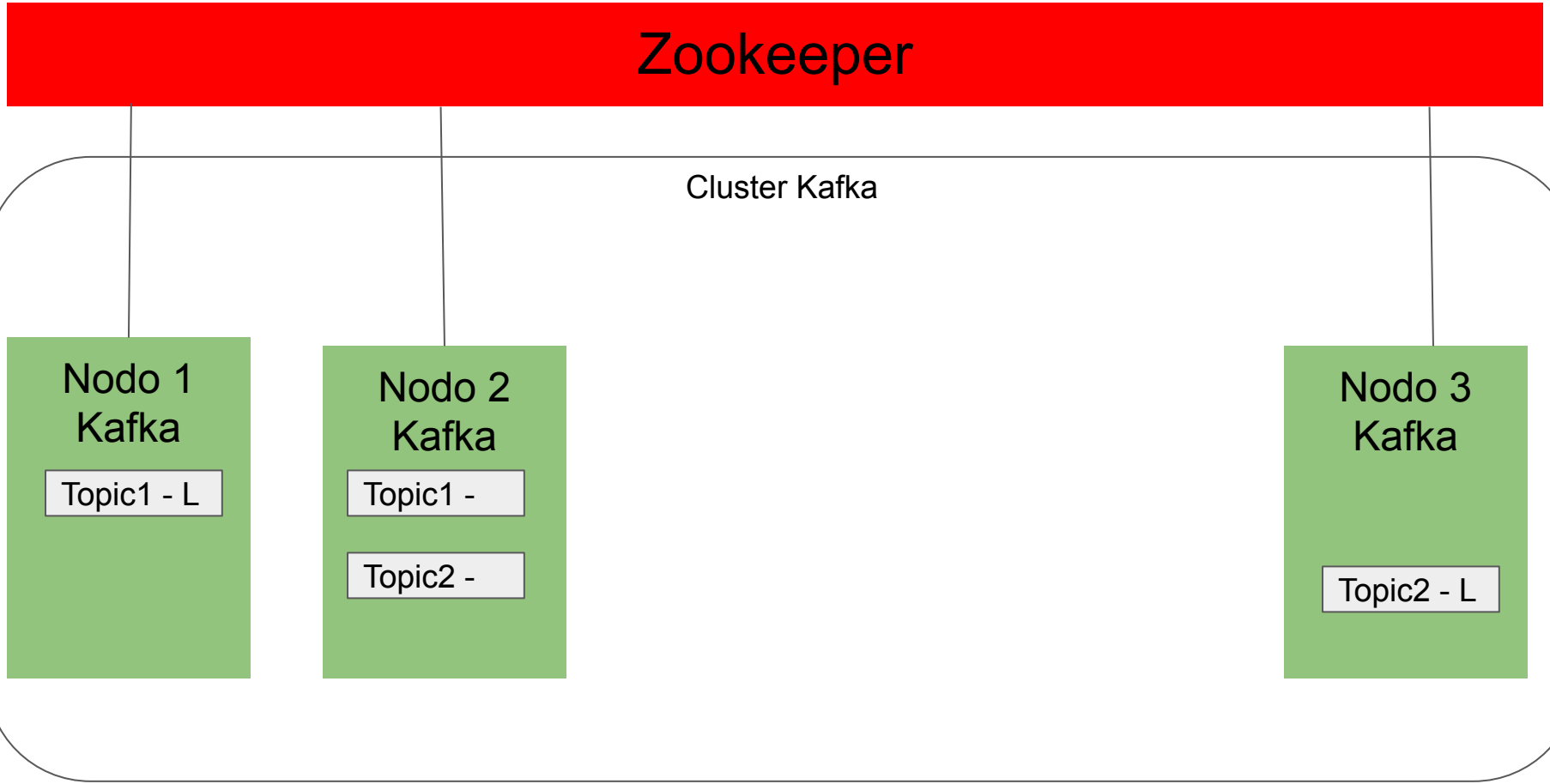
- Broker : cada una de las instancias o servidores kafka
- Clúster de brokers -- agrupación de 1 ó N brokers trabajando juntos
- productor / Consumidor /Grupo de consumidores
- Zookeper: coordinador cluster kafka
- Mensaje: es un paquete de datos que se quiere transmitir vía kafka
- Topic / canales: diferentes canales que se crean, donde se publican los topics
- partición: partes en las que se parte un canal
- offset: posición al último mensaje leído
- kafka connect
- kafka streams
- Factor de replicación: nº de réplicas de una partición

# Zookeeper

- *Gestor del clúster (coordina la topología de los brokers / clúster) y servicio centralizado para la gestión de la configuración, registro de cambios, servicio de descubrimiento, etc.*
- *Intercambia metadatos con : brokers, productores y consumidores*
  - *Direcciones de brokers*
  - *Offsets de los mensajes consumidos*
  - *Descubrimiento y control de los brokers del clúster*
  - *Detección de la carga de trabajo y asignación de trabajo*
- *Proporciona una vista sincronizada de la configuración del clúster*

# Brokers - Kafka

- Un broker es una instancia de un nodo kafka.
- Cada broker tiene un Id
- *Contiene la/s partición/es de uno o varios topic/s*
  - *Cada broker puede contener la partición leader o bien una partición réplica de un topic*
  - *Sólo un broker del clúster podrá tener la partición leader de un topic*
- *Está en comunicación frecuente con el Zookeeper*
- *El rendimiento depende del hw y de la configuración:*
  - *Se aconseja que cada broker tenga un nº menor a 2000 particiones*


Zookeeper
Cluster Kafka
Nodo 1
Kafka
Topic1 - L
Nodo 2
Kafka
Topic1 -
Topic2 -
Nodo 3
Kafka
Topic2 - L

# Mensaje / registro

- *Unidad de datos con la que trabaja Kafka*
- *No tiene una estructura, formato o significado específico para Kafka (se pueden usar schemas)*
- *Cada mensaje puede tener una clave/Key ( todos los mensajes con la misma key, usan la misma partición)*
- *Los mensajes se pueden agrupar en Batch / lotes*
  - *colección de mensajes producidos por el mismo topic y particion*
- *Los mensajes tienen periodo de retención*
- *Estado : confirmado y no confirmado (no todas las particiones se han actualizado)*

# Esquema

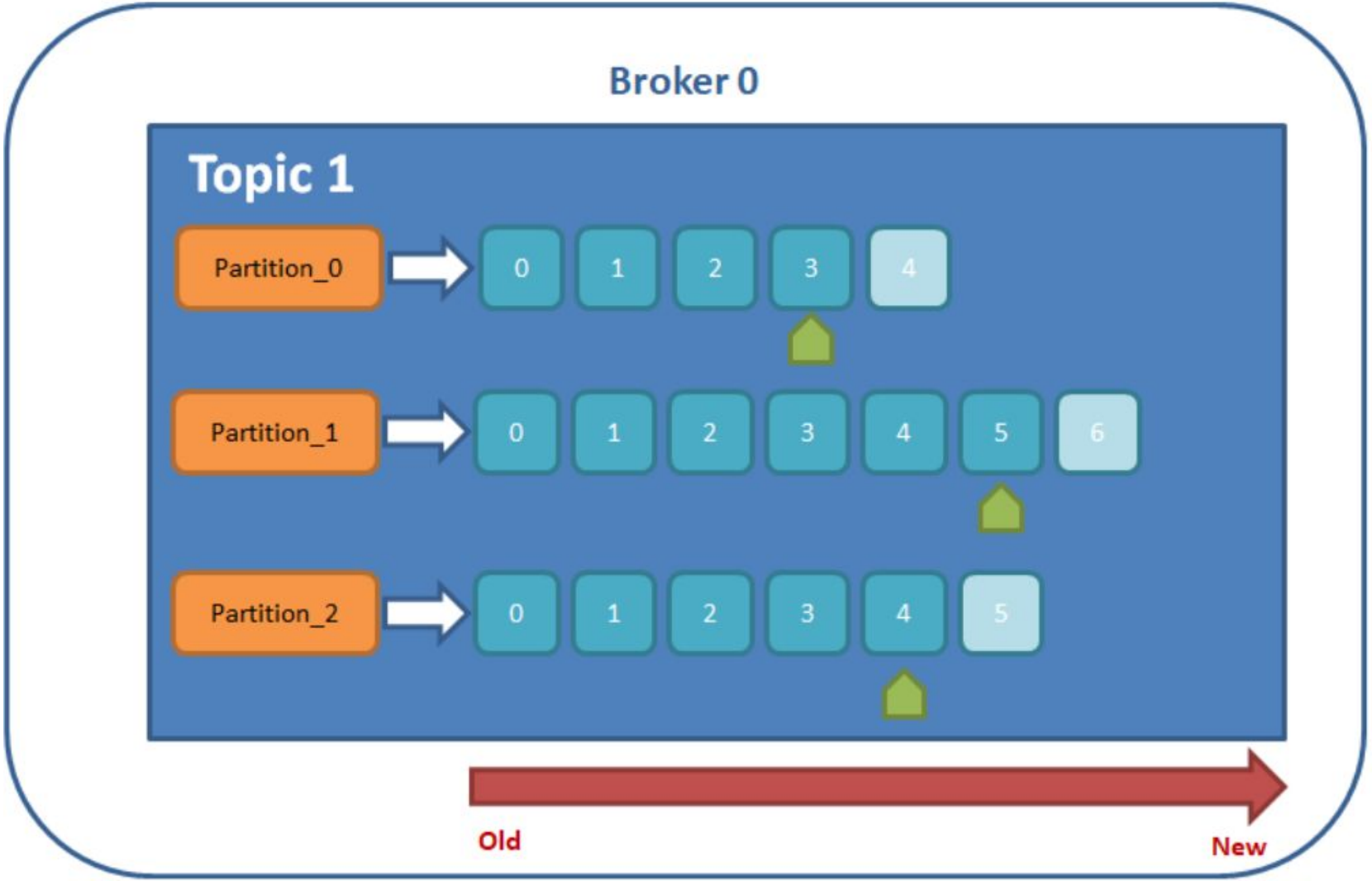
- Estructura **opcional** que se aplica sobre los mensajes garantizando una distribución concreta de la información
- Formato de los datos con los que se trabajará (JSON, XML)

# Topic / Canal / Tema

- Criterio por el que se agrupan los mensajes
- Cada topic tiene su configuración
- Almacena de forma permanente los mensajes a menos que le indiquemos criterio de retención
- Productor → introduce los mensajes en el topic (añadir al final)
- Consumidor → extrae los mensajes del topic de manera ordenada
- Un topic puede tener n particiones (1 por defecto)
- Factor de replicación
- Criterios de retención (tiempo, tamaño, sólo último mensaje)

# Particiones

- Cada topic se puede fragmentar en varias particiones (1 por defecto)
- Secuencia de mensajes ordenada e inmutable
- Las particiones permiten manipulación paralela
- Habrá tantas réplicas de cada partición como el factor de replicación definido a nivel de cluster.
- Cada partición tiene su offset


Broker 0
Topic 1
Partition_0
0
1
2
3
4
Partition_1
0
1
2
3
4
5
6
Partition_2
0
1
2
3
4
5
Old
New

# Partición Leader & partición Follower

- De todas las réplicas de una partición, 1 es leader y el resto son follower
- La partición leader es la que recibe las peticiones de publicación y la que notifica a los suscriptores
- Si una partición leader falla, se asigna otra como leader

# Ordenación de mensajes

**1 sóla partición:**

- Se mantiene el orden de los registros
- Es la opción por defecto

**N particiones:**

- Se mantiene el orden a nivel de partición
- Se recomienda agrupar mensajes en una partición usando claves, en caso de no usar claves, los mensajes de distribuyen aleatoriamente en las diferentes particiones

# Productor

- Se encarga de publicar mensajes mensajes en kafka
- Los mensajes se almacenan en un topic específico, el mensaje se encola al final del topic
- Configuración
  - Síncrona & Asíncrona
  - Batch
    - Tamaño = nº de bytes
    - Balance entre tamaño y velocidad. mayor tamaño > mayor velocidad y mayor latencia
  - Reparto de mensajes
    - Round-robin (cada vez a una partición)
    - por clave. todas las claves van a la misma partición
  - Nivel de consistencia
- Partes de un mensaje:
  - datos
  - topic
  - offset
  - partición

# Nivel de consistencia

ACK=0:

- El productor no espera ningún ACK del broker
- Los mensajes añadidos al topic son considerados enviados
- El mensaje se pierde si la partición leader se cae

ACK=1

- La partición leader escribe el mensaje en su log local pero sin esperar confirmación de los followers
- Si el leader falla después de enviar el ACK, el mensaje se puede perder

ACK2:

- La partición leader espera la confirmación de todos los followers antes de enviar el ACK
- Siempre que haya una réplica de la partición, el mensaje estará disponible

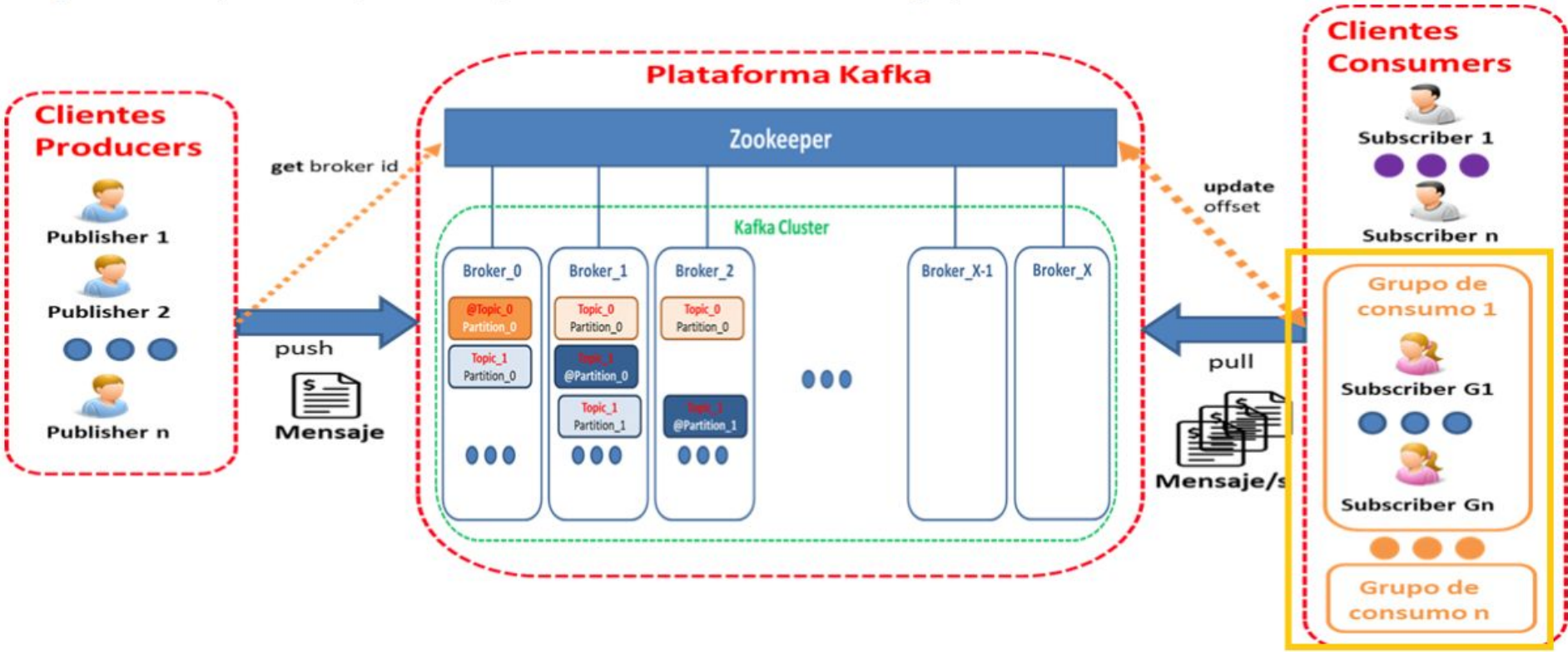
# Consumidor/suscriptor

- Se encarga de consumir mensajes de kafka
- Puede estar suscrito a tantos topics como desee
- Cada consumidor gestiona sus offset  (por partición y topic)
- Un topic puede tener varios consumidores
- Lee los mensajes en el orden en el que se han metido en la cola

# Grupo consumidores

- Permite agrupar consumidores para consumir mensajes de kafka en paralelo.
- Existe un offset de grupo (por topic y partición)
- Los grupos disponen de un nombre único.
- Cada mensaje es entregado a un único consumidor en un grupo.
- Cada partición de un topic es consumida por un miembro del grupo.
- Las particiones se comparten entre los miembros del grupo

# Grupo de consumidores



# Prácticas con kafka

# Instalación + pruebas mononodo (1 zookeeper + 1 kafka )

https://tecadmin.net/how-to-install-apache-kafka-on-ubuntu-20-04/

## 1- instalamos java

Actualizamos sistema → sudo apt update

instalamos java → sudo apt install default-jdk

verificamos versión de java → java --version

## 2- Descargamos kafka (2.8.2)

wget https://dlcdn.apache.org/kafka/2.8.2/kafka_2.13-2.8.2.tgz

tar xzf kafka_2.13-2.8.2.tgz

mv kafka_2.13-2.8.2 /usr/local/kafka

//variables de entorno https://programmerclick.com/article/4020552429/

# Instalación (1 zookeeper + 1 kafka )

## 3. creamos el servicio de zookeeper

vi /etc/systemd/system/zookeeper.service

```
[Unit]
Description=Apache Zookeeper server
Documentation=http://zookeeper.apache.org
Requires=network.target remote-fs.target
After=network.target remote-fs.target
[Service]
Type=simple
ExecStart=/usr/local/kafka/bin/zookeeper-server-start.sh /usr/local/kafka/config/zookeeper.properties
ExecStop=/usr/local/kafka/bin/zookeeper-server-stop.sh
Restart=on-abnormal
[Install]
WantedBy=multi-user.target
```

# Instalacion  (1 zookeeper + 1 kafka )

## 4. creamos el servicio de kafka

vi /etc/systemd/system/kafka.service

```
[Unit]
Description=Apache Kafka Server
Documentation=http://kafka.apache.org/documentation.html
Requires=zookeeper.service
[Service]
Type=simple
Environment="JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64"
ExecStart=/usr/local/kafka/bin/kafka-server-start.sh /usr/local/kafka/config/server.properties
ExecStop=/usr/local/kafka/bin/kafka-server-stop.sh
[Install]
WantedBy=multi-user.target
```

5. Arrancamos los servicios

systemctl daemon-reload

sudo systemctl start zookeeper

sudo systemctl start kafka

sudo systemctl status kafka

# kafka-topic.sh  kafka-topic.bat

kafka-topics.bat

--create ← crea un topic

-- list ← lista los topics

--delete ← borra un topic

--describe ← muestra información de un topic

--alter ← modifica la configuración de un topic

--zookeeper localhost:2181 ← ubicación de zookeeper

--replication-factor 1 ← número de replicas (1 si no tenemos cluster)

--partitions 1 ← número de particiones

--topic demo ← nombre del topic

# Primeros pasos con kafka-topics.sh

6. Listamos los topics creados

bin/kafka-topics.sh --list --zookeeper localhost:2181

7. Creamos un topic

$ cd /usr/local/kafka

$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic testTopic

8. borramos un topic

$ bin/kafka-topics.sh --delete --zookeeper localhost:2181 --topic testTopic

9. Consultamos información de un topic

$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic testTopic

```
root@itorres-Virtual-Machine:/usr/local/kafka# bin/kafka-topics.sh --describe --zookeeper localhost:2181  --topic
Topic1
Topic: testTopic1       PartitionCount: 1       ReplicationFactor: 1    Configs:
        Topic: testTopic1       Partition: 0    Leader: 0       Replicas: 0   Isr: 0
root@itorres-Virtual-Machine:/usr/local/kafka#
```

# Envio datos a kafka (producer)

| | |
|---|---|
| kafka-console-producer.sh | ← crea un producer para mandar mensajes |
| --broker-list localhost:9092 | ← ubicación de los brokers con los que trabajará |
| --topic testTopic | ← topic al que se quiere mandar el mensaje |

## 10. creamos un producer y mandamos mensajes

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

> hola

> esto es una prueba de envío de mensaje

# Recepcion de mensajes (consumer)

```
bin/kafka-console-consumer.sh    ← creacion de consola de consumidor
--bootstrap-server localhost:9092   ← ubicacion del servidor
--topic testTopic   ← topic del que se quiere consumir --from-beginning
```

11. creamos un consumer y recibimos mensajes

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

> hola

> esto es una prueba de envío de mensaje

¿Cómo hacemos para que el nodo lea desde el principio de los mensajes?

En la realidad:

- cada nodo se instalaría en una máquina, del mismo modo que si se instalara solo
- se debe configurar el fichero de propiedades de kafka (server.properties)
  - broker.id= 0,1,2,3,  #cada broker debe tener un id Distinto
  - listeners=PLAINTEXT://:9092  # cada uno escucha en un puerto
  - zookeeper.connect=localhost:2181 # se especifica la url de zookeeper

# Pruebas multinodo  (cluster de kafka)

Simulación multinodo en una máquina:

- Se van a simular 4 brokers kafka en una misma máquina

Preparación del entorno

1. se crean 4 copias del fichero server.properties
    - server-0.properties
    - server-1.properties
    - server-2.properties
    - server3.properties

# Pruebas multinodo  (cluster de kafka)

$ cd /usr/local/kafka/config

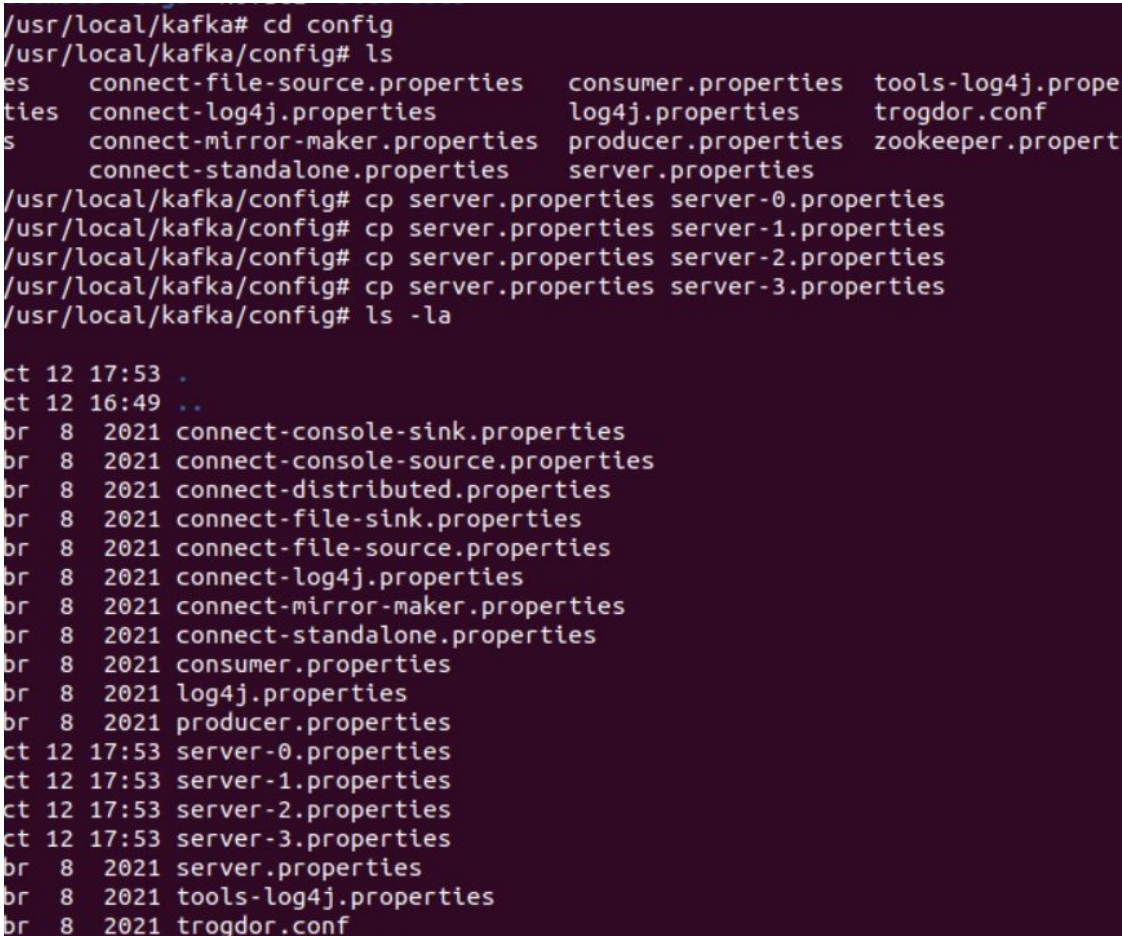
$ ls

$ cp server.properties server-0.properties

$ cp server.properties server-1.properties

$ cp server.properties server-2.properties

$ cp server.properties server-3.properties



# Pruebas multinodo  (cluster de kafka)

## server-0.properties

broker.id=0

listeners=PLAINTEXT://:9092

log.dirs=/tmp/kafka-logs-0

zookeeper.connect=localhost:2181

## server-1.properties

broker.id=1

listeners=PLAINTEXT://:9093

log.dirs=/tmp/kafka-logs-1

zookeeper.connect=localhost:2181

# Pruebas multinodo  (cluster de kafka)

## server-0.properties

broker.id=2

listeners=PLAINTEXT://:9094

log.dirs=/tmp/kafka-logs-2

zookeeper.connect=localhost:2181

## server-1.properties

broker.id=3

listeners=PLAINTEXT://:9095

log.dirs=/tmp/kafka-logs-3

zookeeper.connect=localhost:2181

# Pruebas multinodo  (cluster de kafka)

Arrancamos zookeeper

```
zookeeper-server-start.sh ..\..\config\zookeeper.properties
```

Abrimos 4 terminales y en cada una arrancamos un nodo de kafka

```
kafka 0 >    kafka-server-start.sh ..\..\config\server-0.properties
kafka 1 >    kafka-server-start.sh ..\..\config\server-1.properties
kafka 2 >    kafka-server-start.sh ..\..\config\server-2.properties
kafka 3 >    kafka-server-start.sh ..\..\config\server-3.properties
```

# Ejemplo Multinodo 1:

1) creamos un topic con 3 replicas

```
$ kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 1 --topic topicf3p1

$ kafka-topics.sh --list --zookeeper localhost:2181

$kafka-topics.sh --describe --zookeeper localhost:2181 --topic topicf3p1
```

# Ejemplo Multinodo 1:  1productor + 2 consumidor

## 2) creamos un productor (consola a parte)

$ kafka-console-producer.sh --broker-list localhost:9092,localhost:9093 --topic topicf3p1

## 3) Creamos el 1er consumidor (consola a parte)

$ kafka-console-consumer-sh --bootstrap-server localhost:9092,localhost:9093 --from-beginning --topic topicf3p1

## 4) Escribimos varios mensajes mediante el productor

## 5) Creamos el 2do consumidor (consola a parte)

$ kafka-console-consumer.sh --bootstrap-server localhost:9092,localhost:9093 --from-beginning --topic topicf3p1

# al crear el segundo consumidor se recupera el histórico de mensajes

# Nota listado de servidores

**bootstrap.servers**

A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. The client will make use of all servers irrespective of which servers are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of servers. This list should be in the form `host1:port1,host2:port2,...` . Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list need not contain the full set of servers (you may want more than one, though, in case a server is down).

| | |
|---|---|
| **Type:** | list |
| **Default:** | "" |
| **Valid Values:** | non-null string |
| **Importance:** | high |

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

# Ejemplo Multinodo 1:  1productor + 3 consumidor

6) Creamos el 3do consumidor (consola a parte)

$ kafka-console-consumer.sh --bootstrap-server localhost:9092,localhost:9093 --topic topicf3p1

# en este caso no se recupera ningún mensaje, porque no le hemos indicado que los recupere desde el inicio

6) a través del productor mandamos nuevos mensajes

# los nuevos mensajes son recibidos por todos los consumidores

# el 3er consumidor solo ve los mensajes nuevos desde que se arranco

# Ejemplo Multinodo 2:  1productor + 2 consumidores en grupo- 1 particion

0 ) páramos los productores y consumidores del ejercicio anterior

1) creamos un nuevo topic

$ kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 1 --topic topicf3p1g

$ kafka-topics.sh --list --zookeeper localhost:2181

$kafka-topics.sh --describe --zookeeper localhost:2181 --topic topicf3p1g

# Ejemplo Multinodo 2:  1productor + 2 consumidores en grupo  - 1 particion

2 ) creamos un productor (nueva consola)

$ kafka-console-producer.sh --broker-list localhost:9092,localhost:9093 --topic topicf3p1g

3 ) creamos 1er consumidor (nueva consola)

$ kafka-console-consumer --bootstrap-server localhost:9092,localhost:9093 --topic topicf3p1g --consumer-property group.id=grupo1

4 ) creamos 2do consumidor (nueva consola)

$ kafka-console-consumer --bootstrap-server localhost:9092,localhost:9093 --topic topicf3p1g --consumer-property group.id=grupo1

5 ) desde el productor creamos varios mensajes

#verificar que los mensajes solo se ven en uno de los consumidores

#hay un consumidor asignado a la particion leader

# Ejemplo Multinodo 3:  1productor + 2 consumidores en grupo 2 particiones

0 ) páramos los productores y consumidores del ejercicio anterior

1)  creamos un nuevo topic

$ kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 2 --topic topicf3p2g

$ kafka-topics.sh --list --zookeeper localhost:2181

$kafka-topics.sh --describe --zookeeper localhost:2181 --topic topicf3p2g

# Ejemplo Multinodo 3:  1productor + 2 consumidores en grupo 2 particiones

2 ) creamos un productor (nueva consola)

$ kafka-console-producer.sh --broker-list localhost:9092,localhost:9093 --topic topicf3p2g

3 ) creamos 1er consumidor (nueva consola)

$ kafka-console-consumer --bootstrap-server localhost:9092,localhost:9093 --topic topicf3p2g --consumer-property group.id=grupo1

4 ) creamos 2do consumidor (nueva consola)

$ kafka-console-consumer --bootstrap-server localhost:9092,localhost:9093 --topic topicf3p2g --consumer-property group.id=grupo1

5 ) desde el productor creamos varios mensajes

#verificar que los mensajes se reparten entre los consumidores

# Ejemplo Multinodo 4:  detener servidores

0 ) páramos los productores y consumidores del ejercicio anterior

1)  creamos un nuevo topic

$ kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 2 --topic topic_server

$ kafka-topics.sh --list --zookeeper localhost:2181

$kafka-topics.sh --describe --zookeeper localhost:2181 --topic topic_server

```
root@itorres-Virtual-Machine:/usr/local/kafka/bin# ./kafka-topics.sh --describe --zookeeper localhost:2181 --topic
icf3p2g
Topic: topicf3p2g       PartitionCount: 2       ReplicationFactor: 3    Configs:
        Topic: topicf3p2g       Partition: 0    Leader: 2       Replicas: 2,0,1 Isr: 2,0,1
        Topic: topicf3p2g       Partition: 1    Leader: 0       Replicas: 0,1,2 Isr: 0,1,2
root@itorres-Virtual-Machine:/usr/local/kafka/bin#
```

# Ejemplo Multinodo 4:  detener servidores

2 ) páramos el server 0  (leader de la particion 1)

3)Verificamos que al hacer describe se modifica la lista de servidores

# Ref

- Web oficial https://kafka.apache.org/
- Web Enmilocalfunciona https://enmilocalfunciona.io/aprendiendo-apache-kafka-parte-1/
- Youtube canal Albert Coronado https://www.youtube.com/playlist?list=PLwH0tlWs8nkSQRxizVF5Uuu-sLVYqdjaW
- kafka producer & kafka stream https://www.baeldung.com/java-kafka-streams-vs-kafka-consumer
-