

Servicios en red: Kafka

Administración de Sistemas

Unai Lopez Novoa
unai.lopez@ehu.eus

eman ta zabal zazu



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

Contenido

1. Introducción
2. Configuración
3. Replicación



Introducción

- En muchos entornos necesitamos que los mensajes enviados por un cliente no se pierdan.
 - Aunque no haya “suscriptores” conectados.
- En estos entornos se debe utilizar un sistema que gestione la persistencia.
 - MQTT no es un sistema de colas.
 - A pesar de la Q en su nombre.
 - Apache Kafka es uno de los más utilizados.



Apache Kafka



- Sistema distribuido de gestión de mensajes
 - Open Source
 - Web: <https://kafka.apache.org/>
- Escalable, robusto y confiable
 - Más complejo que Mosquitto
- En estas diapositivas veremos su uso básico.



Apache Kafka



- Muy utilizado en entornos Big Data, p.e.:
 - Industria 4.0, para monitorizar máquinas de fabricación



Apache Kafka

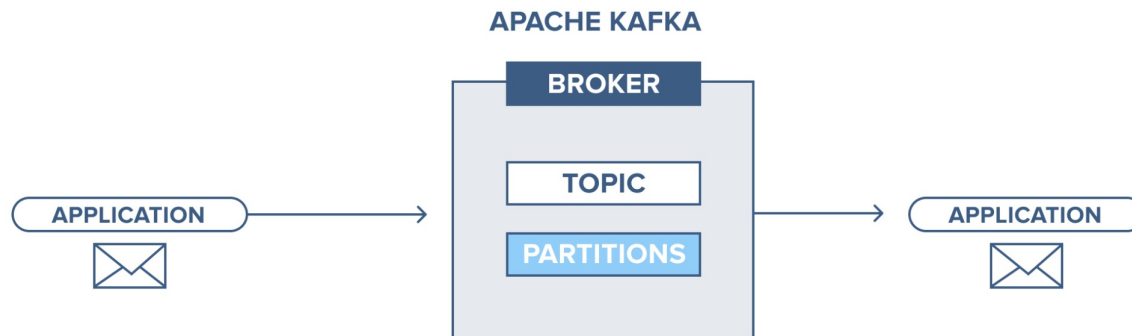


- Muy utilizado en entornos Big Data, p.e.:
 - Sector financiero, para registrar transacciones



Apache Kafka

- Arquitectura
 - De forma general, un sistema de comunicación entre aplicaciones



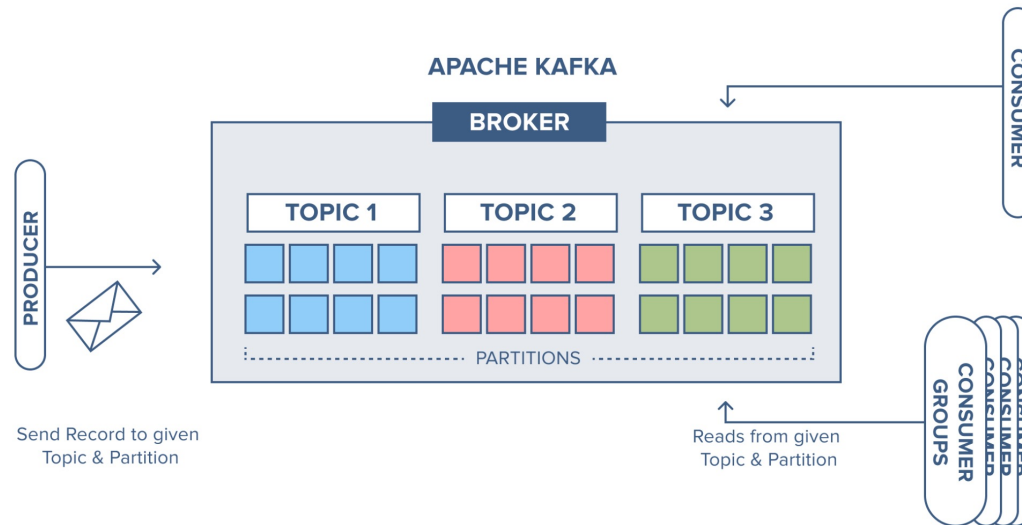
- Broker: Servicio gestor de los mensajes
- Topic: Categoría que sirve para ordenar mensajes
- Particiones: Almacenan los mensajes de los *topic*



Apache Kafka

- Arquitectura

- Provee persistencia y replicación a nivel de *topic* usando particiones



- Productor: equivalente a *Publisher* en MQTT
- Consumidor: equivalente a un *Subscriber* en MQTT



Apache Kafka

- Instalación en un sistema Unix/Linux
 - Descargar última versión

```
wget https://downloads.apache.org/kafka/3.5.1/kafka_2.13-3.5.1.tgz
```

- Kafka funciona sobre Java
 - Si Java no está instalado, es necesario instalar un entorno.
 - Ejemplo para instalar el entorno de ejecución Java por defecto:

```
sudo apt install default-jre
```



Apache Kafka



- Zookeeper
 - Servicio de coordinación de procesos distribuidos.
 - Directorio de los nodos disponibles en el sistema, guarda sus nombres y servicios.
 - Necesario para Kafka, coordina los Brokers.
 - Viene incluido en la instalación actual de Kafka.
 - Aunque desaparecerá en la versión 4.0
 - Iniciar Zookeeper con la configuración por defecto:
 - *Usar desde la carpeta en la que se ha descomprimido Kafka.*

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```



Apache Kafka

- Configuración de Kafka
 - Fichero *config/server.properties*
 - Contiene la configuración del Broker en el nodo local.
- Permitir conexiones desde cualquier IP, añadir las siguientes líneas¹:
 - Incluir la IP pública de vuestra máquina en la 2ª línea

```
##### Socket Server Settings #####  
...  
listeners=PLAINTEXT://:9092  
advertised.listeners=PLAINTEXT://<IP-pública>:9092
```



Apache Kafka

- Iniciar un Broker

- Iniciar Broker con la configuración por defecto
 - Usar desde la carpeta en la que se ha descomprimido Kafka.

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

- El proceso Broker queda a la espera de conexiones.
 - Muestra un log similar al siguiente:

```
...
[2023-10-09 09:47:02,315] INFO Initiating client connection,
connectString=34.155.0.196:2181 sessionTimeout=18000
watcher=kafka.zookeeper.ZooKeeperClient$ZooKeeperClientWatcher$@7bf3a5d8
(org.apache.zookeeper.ZooKeeper)
...
[2023-10-09 09:47:02,333] INFO [ZooKeeperClient Kafka server] Waiting until connected.
(kafka.zookeeper.ZooKeeperClient)
[2023-10-09 09:47:02,335] INFO Opening socket connection to server /34.155.0.196:2181.
(org.apache.zookeeper.ClientCnxn)
```



Apache Kafka

- En Kafka, los *topic* deben crearse de antemano
 - A diferencia de MQTT
- Los mensajes de un *topic* se almacenan de forma ordenada y persistente.
- Desde línea de comandos:
 - El parámetro *--bootstrap-server* indica a qué Broker conectarse.
 - Ejemplo para crear un *topic* “miTopic” en un Broker local.

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



Apache Kafka

- Herramientas de línea de comando para uso básico:

- Consumir eventos de un *topic*

- El parámetro *--from-beginning* recupera todos los eventos del topic.
 - Ejemplo:

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

- Enviar eventos a un topic

- Cada nueva línea se envía como un evento.
 - Ejemplo:

```
bin/kafka-console-producer.sh --topic miTopic --bootstrap-server  
localhost:9092
```



Ejercicio 1

- *Este ejercicio se realizará en parejas (seréis A y B)*
 - A será productor de mensajes
 - B será consumidor y ejecutará el Broker Kafka
- B crea un *topic* llamado "colores" y se conecta a él.
- A envía mensajes "verde" y "azul" al *topic*.
- B se desconecta del *topic*.
- A envía mensajes "amarillo" y "naranja" al *topic*.
- B se conecta al *topic*. Debe recibir todos los mensajes.

Puede ser necesario
abrir el puerto TCP
9092 en GCP



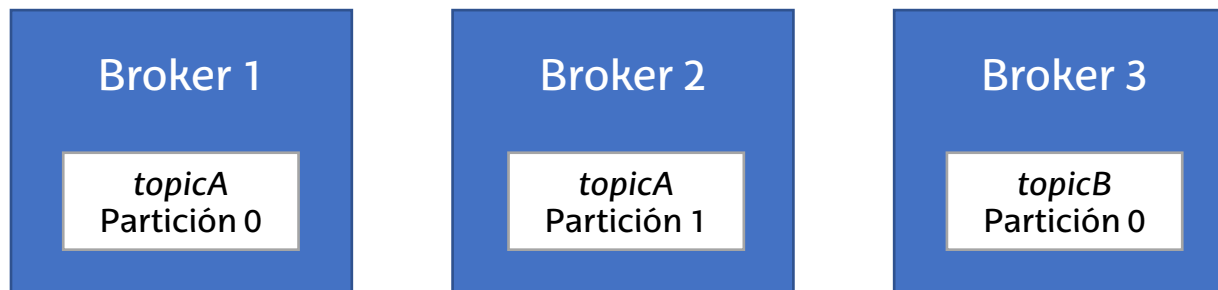
Apache Kafka

- En determinados entornos necesitamos medidas para evitar pérdidas de datos.
 - Aunque sucedan fallos.
- Kafka permite configurar técnicas de replicación para gestionar copias automáticas de los mensajes.



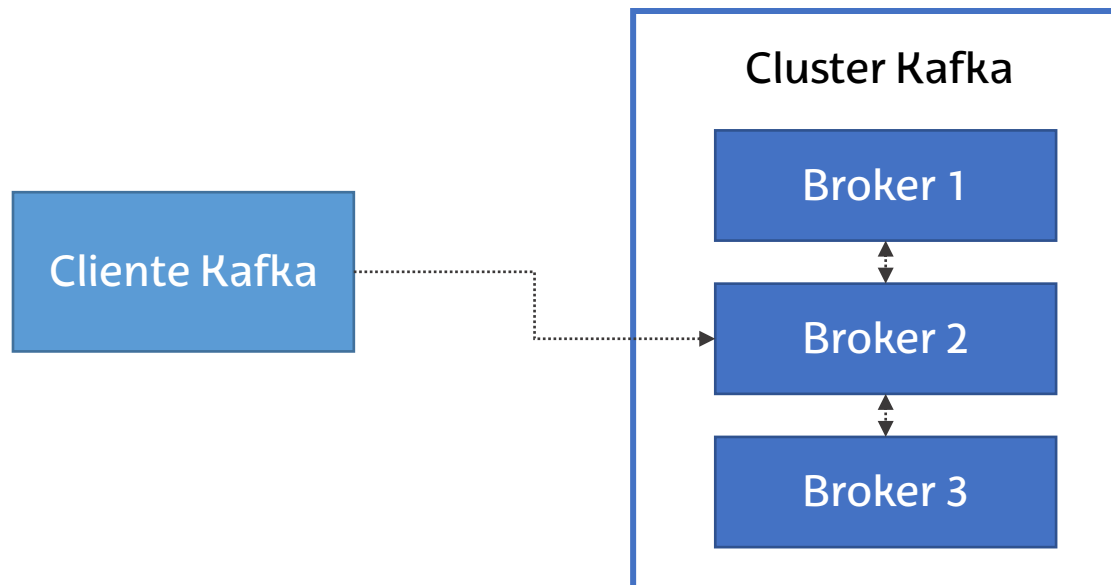
Apache Kafka

- Un cluster Kafka está compuesto por múltiples Brokers
- Cada Broker:
 - Está identificado por un número.
 - Contiene determinadas particiones de un *topic*.
 - Ejemplo:



Apache Kafka

- Al conectarnos a un Broker, tenemos acceso al cluster completo.
 - El Broker al que nos conectamos se denomina *Bootstrap*.
 - *Ejemplo:* Productor se conecta a Broker 2, que hace de *Bootstrap*.



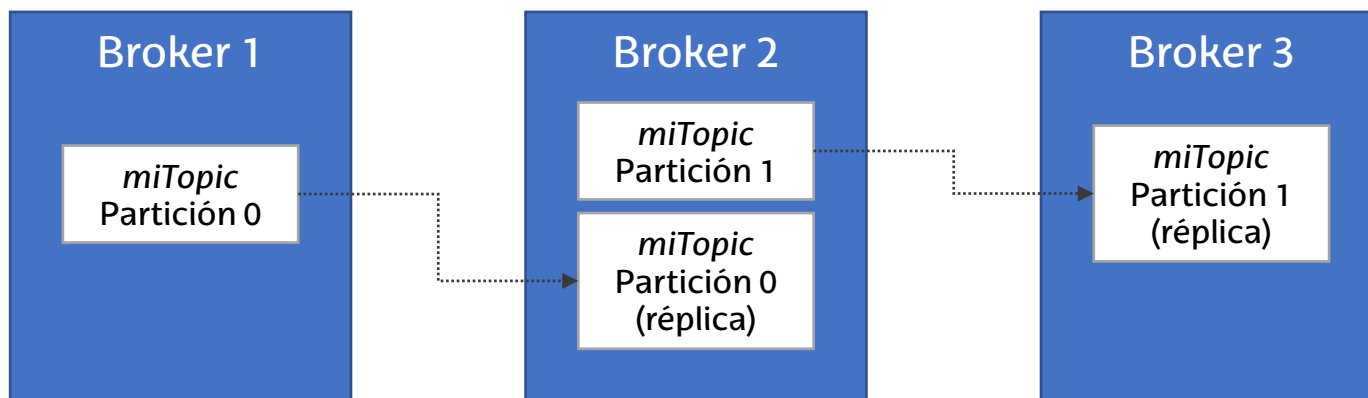
Apache Kafka

- El factor de replicación de un topic controla cuántas copias de los mensajes se realizan en los diferentes Brokers.
 - Por defecto, 1.
 - Recomendable elegir 2 o 3.
- Si un Broker está caído, otro Broker que contenga una replica puede contestar a los clientes.



Apache Kafka

- Las réplicas se organizan en base a las particiones de cada *topic*.
- Ejemplo:
 - *Topic* miTopic con 2 particiones y factor 2 de replicación.
 - Si falla el Broker 2, los datos siguen estando disponibles.



Apache Kafka

- Configuración multi-Broker
 - Además de los parámetros *listener* y *advertised.listeners*, se debe configurar:
 - El número de Broker.
 - Parámetro *broker.id*, número entero.

```
##### Server Basics #####  
# The id of the broker. This must be set to a unique integer for each broker.  
broker.id=<ID>
```

- La IP y puerto de Zookeeper

```
##### Zookeeper #####  
# Zookeeper connection string ...  
zookeeper.connect=<IP>:<Puerto>
```



Apache Kafka

- Gestión de topics multi-Broker
 - Indicar factor de replicación al crear *topic*:
 - Añadir parámetro *--replication-factor N* en la creación, donde N es el número de replicas.

```
bin/kafka-topics.sh --create ... --replication-factor N
```

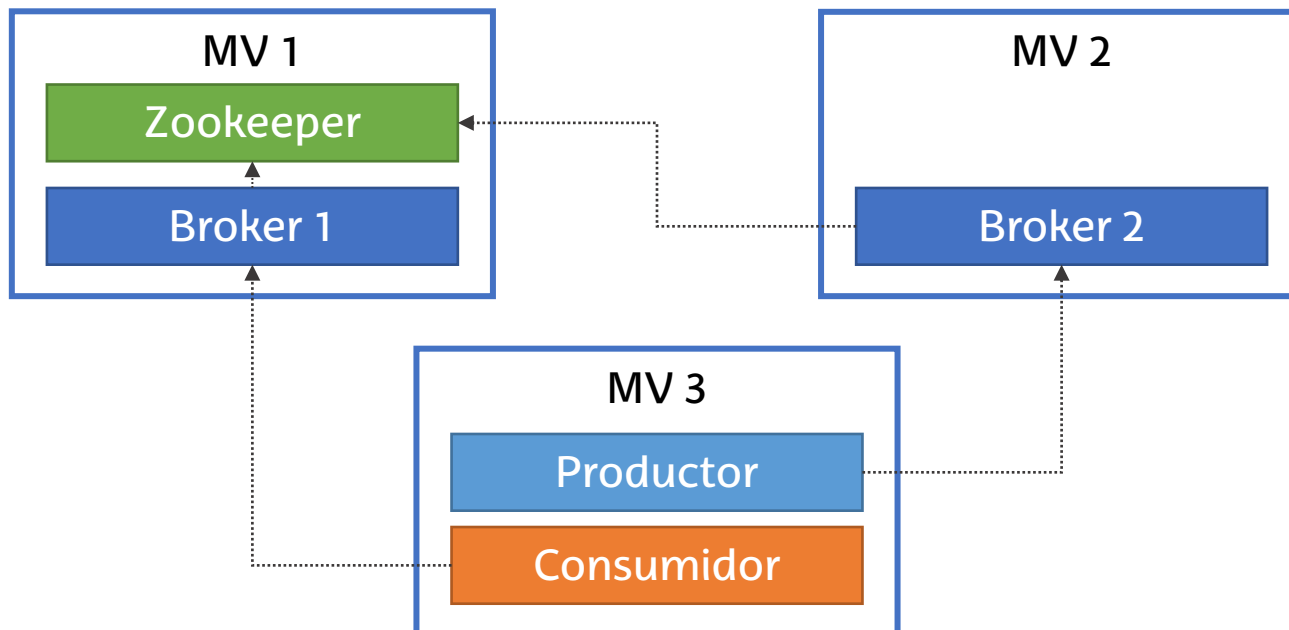
- Mostrar información del *topic*:
 - Parámetro *--describe*.

```
bin/kafka-topics.sh --topic <nombre> ... --describe
```



Ejercicio 2

- *Este ejercicio se realizará en parejas (seréis A y B)*
- Preparar el siguiente sistema usando 3 MVs:



Ejercicio 2

- *Este ejercicio se realizará en parejas (seréis A y B)*
 - A será productor y consumidor de mensajes
 - B ejecutará Brokers Kafka
- B prepara las MV1 y MV2 con Zookeeper y 2 Brokers.
- B crea un *topic* "ciudades" con factor de replicación 2.
 - Usa MV2 como *bootstrap-server*.
- A envía varios mensajes al *topic* "ciudades"
 - Usa MV2 como *bootstrap-server*.
- B apaga MV2.
- A lee los mensajes de "ciudades" usando MV1.



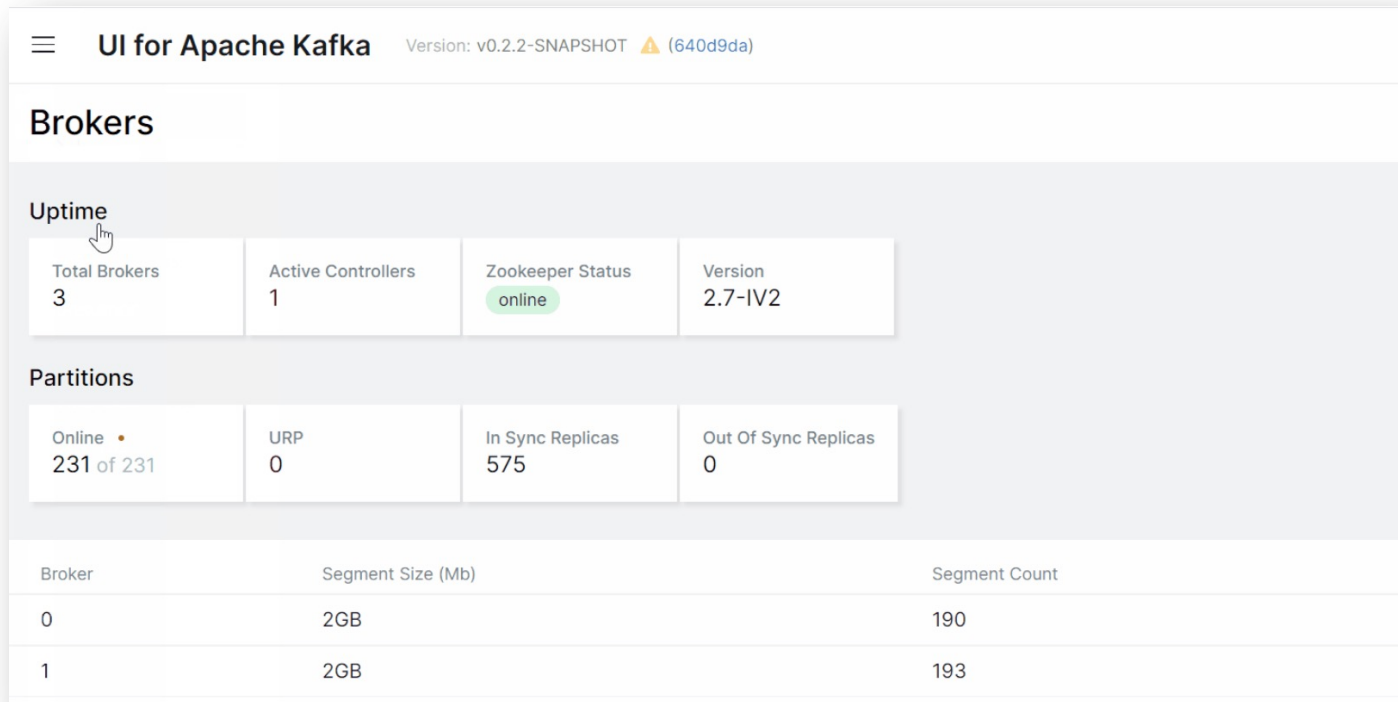
Más información

- Más detalles sobre cómo funciona la replicación:
 - https://medium.com/@_amanarora/replication-in-kafka-58b39e91b64e
- Existen librerías de Kafka para múltiples lenguajes de programación.
 - P.e. en Java: <https://learn.microsoft.com/es-es/azure/hdinsight/kafka/apache-kafka-producer-consumer-api>
- Se pueden configurar múltiples mecanismos de seguridad:
 - +info: <https://kafka.apache.org/documentation/#security>



Más información

- Existen interfaces Web para monitorizar Kafka.
 - P.e. *UI for Apache Kafka*: <https://github.com/provectus/kafka-ui>



The screenshot displays the 'UI for Apache Kafka' dashboard. At the top, it shows the title 'UI for Apache Kafka' and the version 'v0.2.2-SNAPSHOT' with a warning icon and a commit hash '(640d9da)'. Below this, the 'Brokers' section is visible, featuring a 'Uptime' label with a hand cursor icon. The 'Uptime' section contains four cards: 'Total Brokers' with the value '3', 'Active Controllers' with the value '1', 'Zookeeper Status' with the value 'online' (highlighted in green), and 'Version' with the value '2.7-IV2'. Below the 'Uptime' section is the 'Partitions' section, which contains four cards: 'Online' with the value '231 of 231' (indicated by a red dot), 'URP' with the value '0', 'In Sync Replicas' with the value '575', and 'Out Of Sync Replicas' with the value '0'. At the bottom of the dashboard is a table with three columns: 'Broker', 'Segment Size (Mb)', and 'Segment Count'. The table has two rows of data: Broker 0 with a segment size of 2GB and a segment count of 190, and Broker 1 with a segment size of 2GB and a segment count of 193.

Broker	Segment Size (Mb)	Segment Count
0	2GB	190
1	2GB	193



Bibliografía

- Apache Kafka Documentation, "Getting Started", 2022.
 - <https://kafka.apache.org/documentation/>
- L. Johansson, "Apache Kafka for beginners: What is Apache Kafka?", CloudKarafka, 2020.
 - <https://www.cloudkarafka.com/blog/part1-kafka-for-beginners-what-is-apache-kafka.html>
- S. Maarek, "Apache Kafka Series - Learn Apache Kafka for Beginners v3", Udemy, 2023.
 - <https://www.udemy.com/course/apache-kafka/>
- Consultados en septiembre 2022 y octubre 2023.

