

Trabajo Práctico Módulo 3. Gestión de datos en tiempo real (streaming)

Apartado 1: Kafka

Este apartado se corresponde con la primera parte de la práctica. Se debe mencionar que la instalación de Apache Kafka se ha realizado siguiendo el tutorial que aparece en la web de Apache <https://kafka.apache.org/quickstart>. La versión de Kafka instalada es la indicada en el tutorial en el momento de realizar la práctica, que se corresponde con la 2.13-3.0.0. El sistema operativo del ordenador empleado para realizar la práctica es Ubuntu 20.04.

1. Localizar el archivo que contiene la configuración del servidor Zookeeper y levantar dicho servidor utilizando la configuración por defecto. ¿Qué puerto es el utilizado por defecto por Zookeeper?

El archivo que contiene la configuración del servidor Zookeeper se encuentra en la ruta `config/zookeeper.properties`. Esta ruta es relativa al directorio de Kafka. Para levantar el servidor Zookeeper ejecutamos el siguiente comando que indica la ruta del ejecutable y su configuración:

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

Por defecto, el puerto utilizado por zookeeper es el 2181. Esto se puede comprobar ejecutando el comando `cat config/zookeeper.properties` que muestra el contenido del archivo de configuración. La variable “`clientPort`” almacena el valor del puerto configurado por defecto.

2. Localizar el archivo que contiene la configuración de un broker y levantar dicho broker. ¿Qué puerto es el utilizado por defecto por los brokers?

El archivo que contiene la configuración de un broker se encuentra en la ruta `config/server.properties`. Para levantar un broker se ejecuta el siguiente comando:

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

Para obtener el puerto utilizado por defecto por los brokers se ejecuta el comando `cat config/kafka.properties` para leer su fichero de configuración. En la sección `Socket Server Settings` se puede observar que el puerto configurado por defecto es el 9092.

3. Crear un topic nuevo con el nombre “initial”, con una partición y un factor de replicación igual a 1.

Para crear un topic nuevo se utiliza el programa kafka-topics.sh. Este programa puede recibir varios argumentos entre los que se encuentran el nombre del topic, el número de particiones en los que se va a dividir y su factor de replicación. El comando que se ejecuta es el siguiente:

```
bin/kafka-topics.sh --create --partitions 1 --replication-factor 1 --topic initial
--bootstrap-server localhost:9092
```

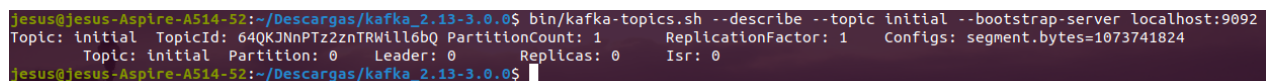
Después de su ejecución aparece un mensaje de confirmación indicando que el topic se ha creado correctamente.

4. Describir el topic “initial” y comentar los resultados que se obtienen.

Para describir un topic se utiliza el ejecutable kafka-topics.sh con la opción “describe”, el nombre del topic y la IP del broker. El comando a ejecutar sería el siguiente:

```
bin/kafka-topics.sh --describe --topic initial --bootstrap-server localhost:9092
```

El resultado obtenido es el siguiente:



```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-topics.sh --describe --topic initial --bootstrap-server localhost:9092
Topic: initial TopicId: 640KJNnPTz2znTRWill6bQ PartitionCount: 1 ReplicationFactor: 1 Configs: segment.bytes=1073741824
Topic: initial Partition: 0 Leader: 0 Replicas: 0 Isr: 0
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$
```

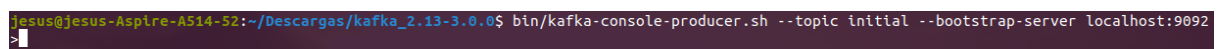
En la imagen se puede observar como el comando devuelve la información relevante de un topic, como: su nombre, su identificador, el número de particiones y de réplicas y el tamaño de los segmentos en bytes. Esta información concuerda con la configuración del topic que se ha establecido en el punto anterior.

5. Arrancar un productor sobre el topic “initial”.

Para arrancar un productor sobre un topic se utiliza el ejecutable bin/kafka-console-producer.sh. Este programa recibe como argumentos el nombre del topic y la IP y puerto del broker al que conectarse. El comando ejecutando es el siguiente:

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

Tras ejecutar este comando en la consola aparece una entrada para poder escribir un mensaje en ese topic.



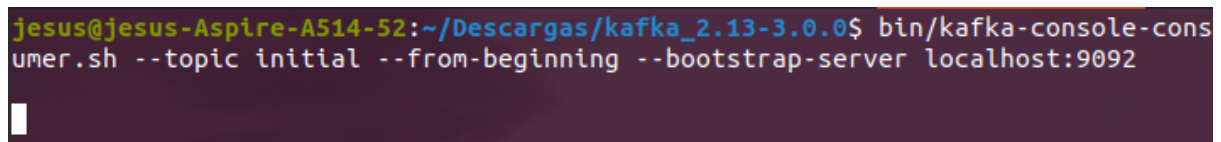
```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-console-producer.sh --topic initial --bootstrap-server localhost:9092
>
```

6. Arrancar un consumidor sobre el topic “initial” que consuma todos los mensajes desde el principio.

Para arrancar un consumidor sobre un tema se utiliza el ejecutable bin/kafka-console-consumer.sh. Este programa recibe como argumento el nombre del topic y la IP y puerto del broker al que se va a conectar. Además, se puede incluir el argumento --from-beginning que especifica leer todos los mensajes desde el inicio. El comando a ejecutar es el siguiente:

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

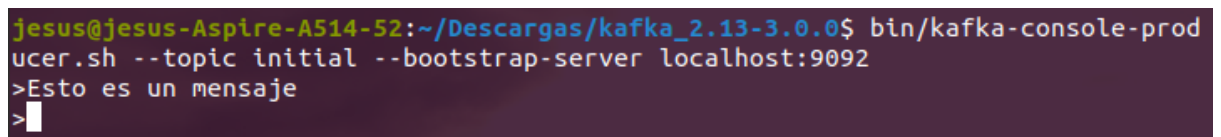
Tras ejecutar el comando en la consola, el programa se queda a la espera de empezar a recibir mensajes. Como no se ha escrito ningún mensaje anteriormente no se muestra nada inicialmente.



```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-console-consumer.sh --topic initial --from-beginning --bootstrap-server localhost:9092
```

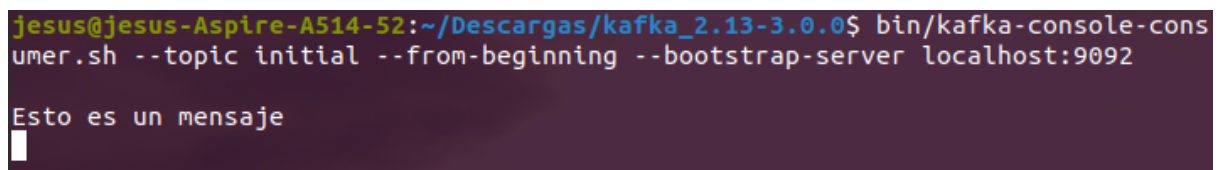
7. Generar algunos mensajes en el productor levantado en el ejercicio 5 y observar cómo se reciben en el consumidor levantado en el ejercicio 6. Comentar la salida obtenida. Detener a continuación el productor y el consumidor.

Para generar mensajes en el productor simplemente se escribe en la consola correspondiente un mensaje y se pulsa la tecla intro para enviarlo. Tras enviarlo se crea una nueva línea para enviar los mensajes sucesivos.



```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-console-producer.sh --topic initial --bootstrap-server localhost:9092
>Esto es un mensaje
>
```

Por parte del consumidor, se puede observar que los mensajes producidos aparecen en la consola:



```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-console-consumer.sh --topic initial --from-beginning --bootstrap-server localhost:9092
Esto es un mensaje
```

Por último, para detener al consumidor y al productor es suficiente con pulsar la combinación Ctrl + C.

8. Crear un topic nuevo con el nombre “testRep”, con dos particiones y un factor de replicación igual a 2. ¿Qué sucede? ¿Cuál es la razón?

Para crear un nuevo topic utilizamos el mismo ejecutable de la tercera pregunta. El comando es el siguiente:

```
bin/kafka-topics.sh --create --partitions 2 --replication-factor 2 --topic testRep
--bootstrap-server localhost:9092
```

Tras ejecutarlo, aparece el siguiente error:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-topics.sh --create --partitions 2
--replication-factor 2 --topic testRep --bootstrap-server localhost:9092
Error while executing topic command : Replication factor: 2 larger than available brokers: 1.
[2022-01-11 23:16:18,536] ERROR org.apache.kafka.common.errors.InvalidReplicationFactorException: Re
plication factor: 2 larger than available brokers: 1.
(kafka.admin.TopicCommand$)
```

El mensaje de error indica que no se puede crear el topic indicado con un factor de replicación dos, que es mayor que el número de brokers actualmente lanzados, uno. Esto se debe a que no hay el suficiente número de brokers en ejecución para poder crear las réplicas del topic. Como el factor de replicación indica el número de copias a realizar de un topic y estas copias se distribuyen en cada broker es necesario que el factor de replicación sea menor o igual al número de brokers actualmente en ejecución.

9. Detener el broker levantado en el ejercicio 2.

Para detener el broker creado anteriormente es suficiente con pulsar las teclas Ctrl-C.

```
[2022-01-11 23:32:56,257] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Shutting down socket server (kafka.network.SocketServer)
[2022-01-11 23:32:56,279] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Shutdown completed (kafka.network.SocketServer)
[2022-01-11 23:32:56,279] INFO Metrics scheduler closed (org.apache.kafka.common.metrics.Metrics)
[2022-01-11 23:32:56,279] INFO Closing reporter org.apache.kafka.common.metrics.JmxReporter (org.apache.kafka.common.metrics.Metrics)
[2022-01-11 23:32:56,280] INFO Metrics reporters closed (org.apache.kafka.common.metrics.Metrics)
[2022-01-11 23:32:56,281] INFO Broker and topic stats closed (kafka.server.BrokerTopicStats)
[2022-01-11 23:32:56,281] INFO App info kafka.server for 0 unregistered (org.apache.kafka.common.utils.AppInfoParser)
[2022-01-11 23:32:56,282] INFO [KafkaServer id=0] shut down completed (kafka.server.KafkaServer)
```

10. Levantar tres brokers distintos: generar nuevos ficheros de configuración para cada uno de ellos indicando las modificaciones que han sido necesarias. Los identificadores de los brokers serán 0, 1 y 2.

En primer lugar, para generar tres brokers distintos se necesita crear un archivo de configuración para cada uno de ellos. En este caso se ha decidido realizar tres copias, una para cada broker, a partir del fichero de configuración inicial y posteriormente modificar las variables necesarias de estos ficheros para poder ejecutar los brokers. Para copiar el contenido del fichero se han ejecutado los comandos:

```
cp server.properties server0.properties
```

```
cp server.properties server1.properties
```

```
cp server.properties server2.properties
```

Una vez creados los tres ficheros de configuración se han modificado. Las variables necesarias de modificar han sido: broker.id, que guarda el número identificador del broker, listeners, que configura el puerto al que se conecta el broker, y log.dirs, que contiene la ruta donde se almacenan los logs producidos por el broker. A cada una de estas variables se les ha añadido un valor único para evitar conflictos con la configuración de otros brokers. Por ejemplo, para broker 0, su broker.id vale “0”, su puerto es el 9092 y la ruta de sus logs “/tmp/kafka-logs-0. El resto de brokers tienen estos valores pero aumentados en una unidad de forma correspondiente. La modificación de estos ficheros se ha realizado con el comando nano:

```
nano server0.properties
```

Finalmente, ejecutamos los 3 brokers:

```
bin/kafka-server-start.sh config/server0.properties
```

```
bin/kafka-server-start.sh config/server1.properties
```

```
bin/kafka-server-start.sh config/server2.properties
```

11. Describir el topic “initial” comentar los resultados que se obtienen.

Para describir el topic initial creado anteriormente, ejecutamos el siguiente comando:

```
bin/kafka-topics.sh --describe --topic initial --bootstrap-server localhost:9092
```

Tras ejecutarlo obtenemos el siguiente resultado:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-topics.sh --describe --topic initial --bootstrap-server localhost:9092
Topic: initial TopicId: sLuGvIOaT_uwhw7RfrwRRg PartitionCount: 1 ReplicationFactor: 1 Configs: segment.bytes=1073741824
Topic: initial Partition: 0 Leader: 0 Replicas: 0 Isr: 0
```

Este resultado es el mismo si ejecutamos el comando en los otros dos brokers:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-topics.sh --describe --topic initial --bootstrap-server localhost:9093
Topic: initial TopicId: sLuGvIOaT_uwhw7RfrwRRg PartitionCount: 1 ReplicationFactor: 1 Configs: segment.bytes=1073741824
Topic: initial Partition: 0 Leader: 0 Replicas: 0 Isr: 0
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-topics.sh --describe --topic initial --bootstrap-server localhost:9094
Topic: initial TopicId: sLuGvIOaT_uwhw7RfrwRRg PartitionCount: 1 ReplicationFactor: 1 Configs: segment.bytes=1073741824
Topic: initial Partition: 0 Leader: 0 Replicas: 0 Isr: 0
```

De esta forma se pone de manifiesto que la configuración de un topic ya creado no se modifica según se hayan creado brokers adicionales. A pesar de que haya más brokers disponibles no se puede aprovechar la tolerancia a fallos porque el topic creado tiene un factor de réplica uno. Para hacer uso de esta característica la configuración del topic debería de ser modificada.

12. Crear un topic nuevo con el nombre “testOrder1”, con 1 partición y factor de replicación igual a 3.

Para crear un topic con la configuración indicada ejecutamos el siguiente comando:

```
bin/kafka-topics.sh --create --partitions 1 --replication-factor 3 --topic testOrder1
--bootstrap-server localhost:9092
```

Tras esto, obtenemos que el topic se ha creado correctamente:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-topics.sh --create --partitions 1 --replication-factor 3 --topic testOrder1 --bootstrap-server localhost:9092
Created topic testOrder1.
```

13. Crear un topic nuevo con el nombre “testOrder2”, con 3 particiones y factor de replicación igual a 3.

Para crear un topic con la configuración indicada ejecutamos el siguiente comando:

```
bin/kafka-topics.sh --create --partitions 3 --replication-factor 3 --topic testOrder2
--bootstrap-server localhost:9092
```

Tras esto, obtenemos que el topic se ha creado correctamente:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-topics.sh --create --partitions 3 --replication-factor 3 --topic testOrder2 --bootstrap-server localhost:9092
Created topic testOrder2.
```

14. Describir los topics “testOrder1” y “testOrder2” y comentar los resultados que se obtienen.

Para describir el topic testOrder1 se ejecuta el siguiente comando:

```
bin/kafka-topics.sh --describe --topic testOrder1 --bootstrap-server localhost:9092
```

Se ha obtenido el siguiente resultado:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-topics.sh --describe --topic testOrder1 --bootstrap-server localhost:9092
Topic: testOrder1      TopicId: HT1Ubh_9Sua4izEhTxw7Ew PartitionCount: 1      ReplicationFactor: 3      Configs: segment.bytes=1073741824
Topic: testOrder1      Partition: 0      Leader: 1      Replicas: 1,0,2 Isr: 1,0,2
```

Además de la información sobre la configuración del topic también se incluye el identificador del broker que es el líder, en este caso es el 1, y una lista con los identificadores de las réplicas ISR (in-sync-replicas). Esta lista sirve para identificar quién es el líder, quiénes son las réplicas y el orden de prioridad en el caso de que un broker falle. En este caso el broker líder es el número 1 seguido de las réplicas 0 y 2. Comentar que el líder se encarga de responder a las peticiones mientras que las réplicas se ocupan de sincronizar los datos entre ellas.

Para describir el topic testOrder2 se ejecuta el siguiente comando:

```
bin/kafka-topics.sh --describe --topic testOrder2 --bootstrap-server localhost:9092
```

Tras esto, se obtiene el siguiente resultado:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-topics.sh --describe --topic testOrder2 --bootstrap-server localhost:9092
Topic: testOrder2      TopicId: hKUIRMxYI_ScWz6orkDoNa PartitionCount: 3      ReplicationFactor: 3      Configs: segment.bytes=1073741824
Topic: testOrder2      Partition: 0      Leader: 0      Replicas: 0,2,1 Isr: 0,2,1
Topic: testOrder2      Partition: 1      Leader: 2      Replicas: 2,1,0 Isr: 2,1,0
Topic: testOrder2      Partition: 2      Leader: 1      Replicas: 1,0,2 Isr: 1,0,2
```

En este caso se obtiene la información anterior pero en tres filas, una por cada réplica del topic. Se puede observar que cada partición está asignada a un broker distinto: la 0 al broker 0, la 1 al broker 2 y la 2 al broker 1. De esta forma se asegura la disponibilidad del programa

a pesar de que uno o más de estos brokers dejase de estar operativo. De la misma forma, la lista de ISR en cada réplica es distinta para cada broker.

15. Arrancar un productor sobre el topic “testOrder1” y un consumidor desde el inicio sobre el mismo topic. Introducir algunos mensajes y observar cómo se consumen. Detener el productor y el consumidor.

En primer lugar, se ejecuta un productor sobre el topic testOrder1 y se introducen varios mensajes:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-console-producer.sh --topic testOrder1 --bootstrap-server localhost:9092
>Esto es otra prueba
>
```

Tras esto, se ejecuta el consumidor sobre el mismo topic y se incluye la opción para leer todos los mensajes desde el principio:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-console-consumer.sh --topic testOrder1 --from-beginning --bootstrap-server localhost:9092
Esto es una prueba
Esto es otra prueba
```

Se puede observar que los mensajes se producen y se consumen de forma correcta y ordenada. Finalmente, se detienen ambos programas con la combinación de teclas Ctrl + C.

16. Arrancar un productor sobre el topic “testOrder2” y un consumidor desde el inicio sobre el mismo topic. Introducir algunos mensajes y observar cómo se consumen. Detener el productor y el consumidor.

En primer lugar, ejecutamos el productor y escribimos varios mensajes:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-console-producer.sh --topic testOrder2 --bootstrap-server localhost:9092
>Esto es una prueba
>Esto es otra prueba
>
```

Seguidamente, ejecutamos el consumidor desde el inicio y observamos la respuesta:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-console-consumer.sh --topic testOrder2 --from-beginning --bootstrap-server localhost:9092
Esto es una prueba
Esto es otra prueba
```

Se puede comprobar que los mensajes se envían correctamente y de forma ordenada. Por último, detenemos la ejecución del productor y del consumidor.

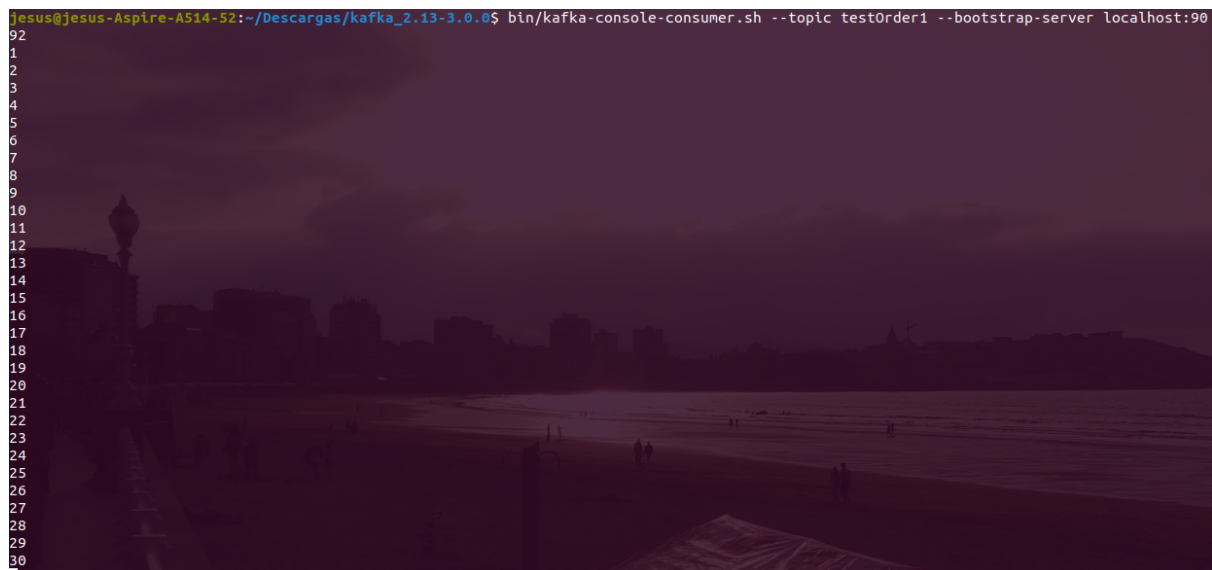
17. Arrancar un consumidor sobre el topic “testOrder1”, sin que consuma desde el principio, únicamente los valores que le lleguen desde el nuevo arranque. En otra consola, generar una secuencia de números del 1 al 30 (utilizar el comando “seq”) y enviársela mediante una tubería o pipe (indicada con el carácter “|”) a un productor sobre el topic “testOrder1”. Observar cómo se consume dicha secuencia en el consumidor.

En primer lugar, ejecutamos el consumidor sobre el topic testOrder1 sin incluir el argumento --from-beginning. Tras esto, se observa que el consumidor no escribe ningún mensaje de los antes creados y se queda esperando a recibir nuevos mensajes.

Seguidamente, se envía la secuencia de número al productor para que se encargue de enviar cada uno de ellos como un mensaje nuevo. Para ello ejecutamos el siguiente comando:

```
seq 30 | bin/kafka-console-producer.sh --topic testOrder1 --bootstrap-server localhost:9092
```

Una vez ejecutado, podemos observar como desde el consumidor han llegado todos los mensajes correctamente de una forma ordenada:

A terminal window with a dark purple background and a cityscape wallpaper. The prompt is 'jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0\$'. The command 'bin/kafka-console-consumer.sh --topic testOrder1 --bootstrap-server localhost:9092' has been executed. The output shows a vertical list of numbers from 1 to 30, with the number 92 at the top of the list. The numbers are displayed in a light blue color.

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-console-consumer.sh --topic testOrder1 --bootstrap-server localhost:9092
92
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

18. Arrancar un consumidor sobre el topic “testOrder2”, sin que consuma desde el principio, únicamente los valores que le lleguen desde el nuevo arranque. En otra consola, generar una secuencia de números del 1 al 30 (utilizar el comando “seq”) y enviársela mediante una tubería o pipe (indicada con el carácter “|”) a un productor sobre el topic “testOrder2”. Observar cómo se consume dicha secuencia en el consumidor.

En primer lugar, ejecutamos el consumidor sobre el topic testOrder2 con el siguiente comando:

```
bin/kafka-console-consumer.sh --topic testOrder2 --bootstrap-server localhost:9092
```

Tras esto, envíamos la secuencia de números a un productor para que publique los mensajes con el comando:

```
seq 30 | bin/kafka-console-producer.sh --topic testOrder2 --bootstrap-server localhost:9092
```


El resultado obtenido en el consumidor es el siguiente:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-console-consumer.sh --topic testOrder2 --bootstrap-server localhost:9092
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

Como se puede observar en la imagen, los mensajes se consumen en el mismo orden en el que han sido producidos.

19. ¿Cuáles son las diferencias entre lo mostrado en el consumidor en los dos ejercicios anteriores? ¿A qué se deben?

En mi caso no existe ninguna diferencia entre el orden de los mensajes consumidos para los topics testOrder1 y testOrder2. Quizá esto se puede deber a que la versión que se está empleando de Kafka para realizar la práctica sea más reciente y resuelva algunos problemas de versiones anteriores.

20. Obtener los últimos offsets de los mensajes del topic “testOrder1” y “testOrder2”. ¿Qué se observa? (Utilizar el comando kafka-run-class con la opción kafka.tools.GetOffsetShell).

En primer lugar, se obtiene el offset del topic testOrder1 con el comando:

```
bin/kafka-run-class.sh kafka.tools.GetOffsetShell --bootstrap-server :9092 --topic testOrder1
```

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-run-class.sh kafka.tools.GetOffsetShell --bootstrap-server :9092 --topic testOrder1
testOrder1:0:32
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$
```

Seguidamente, se ejecuta el mismo comando sobre el topic testOrder2:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-run-class.sh kafka.tools.GetOffsetShell --bootstrap-server :9092 --topic testOrder2
testOrder2:0:1
testOrder2:1:1
testOrder2:2:30
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$
```

Observando las dos imágenes se puede comprobar lo siguiente: el topic testOrder1 solamente tiene una partición, con un identificador 0, y su offset vale 32, es decir, se han leído los 2 primeros mensajes y los 30 siguientes que forman la secuencia. Por otra parte, el topic testOrder2 tiene 3 particiones, cada una con un identificador único, y en la que sus offsets son

diferentes. Esto se debe a que el consumidor utilizado antes para leer la secuencia de números ha empleado la partición con identificador 2. Las particiones con identificador 0 y 1 tienen un offset de 1 porque no se ha realizado ninguna lectura de sus mensajes a través del consumidor que almacena estas particiones.

21. Detener de forma no planificada (utilizando Ctrl-C o cerrando la consola) el broker con identificador 2, y describir el topic “testOrder2”. ¿Cuáles son las diferencias con la descripción que obtuvimos en el ejercicio 14? ¿A qué se deben?

En primer lugar, detenemos el broker cuyo identificador es el número 2 utilizando la combinación Ctrl+C. Tras esto, ejecutamos el siguiente comando para obtener la descripción del topic testOrder2:

```
bin/kafka-topics.sh --describe --topic testOrder2 --bootstrap-server localhost:9092
```

Se obtiene el siguiente resultado:

```
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$ bin/kafka-topics.sh --describe --topic testOrder2 --bootstrap-server localhost:9092
Topic: testOrder2      TopicId: SkXs8lbgShWA3DEf4U3fBg PartitionCount: 3      ReplicationFactor: 3      Configs: segment.bytes=1073741824
Topic: testOrder2      Partition: 0      Leader: 0      Replicas: 0,2,1 Isr: 0,1
Topic: testOrder2      Partition: 1      Leader: 1      Replicas: 2,1,0 Isr: 1,0
Topic: testOrder2      Partition: 2      Leader: 1      Replicas: 1,0,2 Isr: 1,0
jesus@jesus-Aspire-A514-52:~/Descargas/kafka_2.13-3.0.0$
```

Las diferencias entre esta descripción y la descripción del apartado 14 son las siguientes: primero, el líder de la partición 1 ha pasado a ser el broker con identificador 1. Esto se debe a que la lista de ISR de la partición 1 seguía el orden 2,1,0 y a que el broker dos no está disponible. Es decir, el nuevo líder de la partición es el broker con el siguiente identificador de la lista, el broker 1. Además, también es diferente el valor de la lista de las réplicas ISR. En este caso aparecen solamente dos identificadores, de los brokers 0 y 1, ya que hemos finalizado la ejecución del broker 2. Se puede observar que para las particiones donde el líder es el broker 0 el broker que se haría con el control de la partición pasaría a ser el broker 1, y al revés para los casos donde el broker 1 es el líder de la partición.

Conclusiones

Esta primera parte de la práctica me ha resultado muy útil para utilizar y afianzar los conocimientos de Apache Kafka. Era la primera vez que utilizaba este software y tanto su configuración como su uso me han parecido fáciles de asimilar.

Los apartados me han permitido poner en práctica las características principales de Kafka así como conocer los componentes por los que está formado: Zookeeper, brokers, consumidores y productores. También me ha resultado muy interesante comprobar cómo Kafka proporciona la tolerancia a fallos y gestiona las réplicas de las particiones entre los brokers disponibles.

Por otra parte, no he podido obtener los resultados que se esperaban en las preguntas referentes al envío de la secuencia de números. Esto puede deberse a que la versión que he empleado de Kafka soluciona el problema que se pretendía plantear. Sería de gran utilidad que en futuras prácticas se detallaran de forma exacta qué versión del software es necesaria así como su compatibilidad con otros programas empleados durante la asignatura.