

# Proyecto Kafka

María Bellver Carrasco y Alejandro Sáenz Sanchez

27 de mayo de 2019

En este proyecto vamos a realizar una implementación en Kafka de Streaming de datos de la plataforma de Twitter, así como un análisis de polaridad de los datos obtenidos.

## 1. Arquitectura

### 1.1. API de Twitter

Para la realización del trabajo será necesario hacer uso de la API de Twitter, por lo tanto deberemos estar inscritos en el programa de desarrolladores de Twitter, para tener disponibles las claves de acceso. Concretamente las claves de acceso necesarias son las siguientes:

- `consumer_key`
- `consumer_secret`
- `access_token`
- `access_secret`

Para emplear dicha API con Python, haremos uso del paquete Tweepy.

### 1.2. Apache Kafka

En esta sección vamos a explicar la arquitectura del sistema que emplearemos a lo largo del trabajo.

De esta forma explicaremos el proceso de creación de los brokers de Kafka, así como los diferentes scripts de Python, que emplearemos para llevar a cabo este proyecto. Los scripts de Python que hacen uso de Kafka harán uso del paquete Confluent-Kafka.

#### 1.2.1. Creación de los Brokers

La configuración que trae Apache Kafka por defecto es para trabajar con un único broker. Si se levanta el servicio de Kafka con dicha configuración trabajaríamos con un único broker y funcionaría correctamente pero sin obtener todas las ventajas de un servicio distribuido que puede llegar a ofrecer Kafka. Para descubrir las posibilidades que puede aportar la tecnología Apache Kafka, se ha optado por configurar un clúster multi-broker.

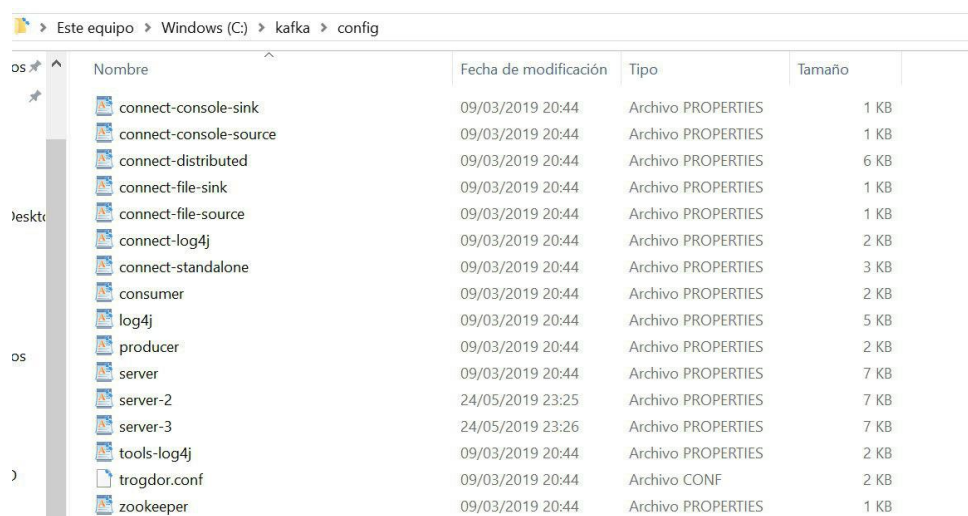
Para ello se va a simular, ya que realmente se va a seguir trabajando con una única máquina en la que se van a levantar tres brokers de Kafka. Para realizar este clúster multibroker, se va a crear la configuración de los brokers.

Apache Kafka ya trae configurado un broker por lo que hay que configurar los otros dos restantes. Para ello hay que dirigirse a la carpeta de configuración de Kafka que dependerá del sistema operativo y la ruta donde esté situado Kafka. Una vez situados en la carpeta de Kafka, deberemos acceder a `config/`. Creamos 2 brokers adicionales partiendo del que ya tenemos, `server.properties` (del broker ya existente), esto se hace haciendo dos copias de este archivo y modificando los siguientes campos:

- `broker.id= "value"`
- `listeners=PLAINTEXT://:"value"`
- `log.dirs=/var/lib/kafka/data-"value"`

Tendremos lo siguiente:

- Para el primer broker con nombre `server.properties` :  
`broker.id= 0`  
`listeners=PLAINTEXT://:9092`  
`log.dirs=/var/lib/kafka/data`
- Para el segundo broker con nombre `server2.properties` :  
`broker.id= 1`  
`listeners=PLAINTEXT://:9093`  
`log.dirs=/var/lib/kafka/data-1`
- Para el tercer broker con nombre `server3.properties` :  
`broker.id= 2`  
`listeners=PLAINTEXT://:9094`  
`log.dirs=/var/lib/kafka/data-2`



Nombre	Fecha de modificación	Tipo	Tamaño
connect-console-sink	09/03/2019 20:44	Archivo PROPERTIES	1 KB
connect-console-source	09/03/2019 20:44	Archivo PROPERTIES	1 KB
connect-distributed	09/03/2019 20:44	Archivo PROPERTIES	6 KB
connect-file-sink	09/03/2019 20:44	Archivo PROPERTIES	1 KB
connect-file-source	09/03/2019 20:44	Archivo PROPERTIES	1 KB
connect-log4j	09/03/2019 20:44	Archivo PROPERTIES	2 KB
connect-standalone	09/03/2019 20:44	Archivo PROPERTIES	3 KB
consumer	09/03/2019 20:44	Archivo PROPERTIES	2 KB
log4j	09/03/2019 20:44	Archivo PROPERTIES	5 KB
producer	09/03/2019 20:44	Archivo PROPERTIES	2 KB
server	09/03/2019 20:44	Archivo PROPERTIES	7 KB
server-2	24/05/2019 23:25	Archivo PROPERTIES	7 KB
server-3	24/05/2019 23:26	Archivo PROPERTIES	7 KB
tools-log4j	09/03/2019 20:44	Archivo PROPERTIES	2 KB
trogdor.conf	09/03/2019 20:44	Archivo CONF	2 KB
zookeeper	09/03/2019 20:44	Archivo PROPERTIES	1 KB

Figura 1: Configuración de un cluster de 3 brokers.

### 1.2.2. Inicio Zookeeper y nodos.

Mediante consola iniciamos Zookeeper y los brokers.

```
Microsoft Windows [Versión 10.0.17134.765]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Windows\system32>cd C:\kafka

C:\kafka>.bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
[2019-05-24 23:27:50,009] INFO Reading configuration from: .\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2019-05-24 23:27:50,013] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DataDirCleanupManager)
[2019-05-24 23:27:50,014] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DataDirCleanupManager)
[2019-05-24 23:27:50,014] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DataDirCleanupManager)
```

Figura 2: Inicio Zookeeper

```
Microsoft Windows [Versión 10.0.17134.765]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Windows\system32>cd C:\kafka

C:\kafka>.bin\windows\kafka-server-start.bat .\config\server.properties
[2019-05-24 23:29:00,570] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2019-05-24 23:29:00,889] INFO starting (kafka.server.KafkaServer)
[2019-05-24 23:29:00,890] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2019-05-24 23:29:00,905] INFO [ZooKeeperClient] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2019-05-24 23:29:00,917] INFO Client environment:zookeeper.version=3.4.13-2d71af4dbe22557fda74f9a9b4309b15a7487f03, built on 06/29/2018 00:39 GMT (org.apache.zookeeper.ZooKeeper)
```

Figura 3: Inicio broker 1

```
Microsoft Windows [Versión 10.0.17134.765]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Windows\system32>cd C:\kafka

C:\kafka>.bin\windows\kafka-server-start.bat .\config\server-2.properties
[2019-05-24 23:29:40,725] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2019-05-24 23:29:41,163] INFO starting (kafka.server.KafkaServer)
[2019-05-24 23:29:41,164] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2019-05-24 23:29:41,198] INFO [ZooKeeperClient] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2019-05-24 23:29:41,218] INFO Client environment:zookeeper.version=3.4.13-2d71af4dbe22557fda74f9a9b4309b15a7487f03, built on 06/29/2018 00:39 GMT (org.apache.zookeeper.ZooKeeper)
```

Figura 4: Inicio broker 2

```
Microsoft Windows [Versión 10.0.17134.765]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Windows\system32>cd C:\kafka

C:\kafka>.bin\windows\kafka-server-start.bat .\config\server-3.properties
[2019-05-24 23:30:26,074] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2019-05-24 23:30:26,459] INFO starting (kafka.server.KafkaServer)
[2019-05-24 23:30:26,460] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2019-05-24 23:30:26,472] INFO [ZooKeeperClient] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2019-05-24 23:30:26,485] INFO Client environment:zookeeper.version=3.4.13-2d71af4dbe22557fda74f9a9b4309b15a7487f03, built on 06/29/2018 00:39 GMT (org.apache.zookeeper.ZooKeeper)
```

Figura 5: Inicio broker 3

Cabe destacar que no se ha modificado la política de eliminación de mensajes de la configurado por defecto de 168 horas, es decir, una semana.

### 1.2.3. Scripts

La arquitectura del sistema consta de un primer productor (Producer\_Tweets.py) que genera el tópic "Tweets" seguido de un consumidor (Consumer\_Producer.py) que se suscribe a éste. Este consumidor a su vez también actuará como productor generando tres tópicos más: "ClasiNeu", "ClasiNeg" y "ClasiPos". Por último, tendremos tres consumidores (Consumer\_tweets\_neu.py, Consumer\_tweets\_neg.py y Consumer\_tweets\_pos.py) suscritos, respectivamente, a los tópicos anteriores.

Se ha decidido crear estos tópicos con 2 particiones y factor de replicación 3 para así, trabajar con un sistema resistente a fallos o posibles incidencias, ya que el sistema realizará tres copias de los datos de las dos particiones para cada tópic donde cada copia residirá en cada broker. De esta forma si el broker con la partición leader del topic se cae, se le podrá asignar otro broker y su partición replica se convertirá en el leader debido al ISR (in-sync replica).

Cabe destacar que en la api de confluent kafka para python no se puede determinar el factor de replicación dentro de la configuración del productor, que es el que crea el tópic, por lo que se crean directamente los tópicos desde la consola con las características explicadas(fig.6).

```
C:\kafka>.bin\windows\kafka-topics.bat --bootstrap-server localhost:9092 --list

C:\kafka>.bin\windows\kafka-topics.bat --bootstrap-server localhost:9092 --create --topic Tweets --partitions 2 --replication-factor 3

C:\kafka>.bin\windows\kafka-topics.bat --bootstrap-server localhost:9092 --create --topic Clasi_neu --partitions 2 --replication-factor 3
WARNING: Due to limitations in metric names, topics with a period (".") or underscore ("_") could collide. To avoid issues it is best to use either, but not both.

C:\kafka>.bin\windows\kafka-topics.bat --bootstrap-server localhost:9092 --create --topic ClasiNeu --partitions 2 --replication-factor 3

C:\kafka>.bin\windows\kafka-topics.bat --bootstrap-server localhost:9092 --create --topic ClasiNeg --partitions 2 --replication-factor 3

C:\kafka>.bin\windows\kafka-topics.bat --bootstrap-server localhost:9092 --create --topic ClasiPos --partitions 2 --replication-factor 3

C:\kafka>.bin\windows\kafka-topics.bat --bootstrap-server localhost:9092 --list
ClasiNeg
ClasiNeu
ClasiPos
Clasi_neu
Tweets

C:\kafka>.bin\windows\kafka-topics.bat --describe --zookeeper localhost:2181 --topic Tweets
Topic:Tweets    PartitionCount:2    ReplicationFactor:3    Configs:
Topic: Tweets   Partition: 0        Leader: 0               Replicas: 0,2,1 Isr: 0,2,1
Topic: Tweets   Partition: 1        Leader: 2               Replicas: 2,1,0 Isr: 2,1,0
```

Figura 6: Creación de los topics

**Nota:** En la imagen hay un error en la creación de los tópicos. El tópic "Clai\_neu" no será utilizado.

Como se quiere emplear 3 brokers deberemos tener en cuenta los siguientes parámetros a la hora de configurar los productores, primero creamos la variable `KAFKA_BOOTSTRAP_SERVERS = "localhost:9092,localhost:9093,localhost:9094"`, con la configuración de los puertos de cada uno de los brokers y a la hora de crear el productor, le introducimos los siguientes parámetros:

- `'bootstrap.servers': "KAFKA_BOOTSTRAP_SERVERS"`
- `'default.topic.config' : {'acks' : 'all'}`

Empleamos la configuración `acks: all`, para tener la configuración más fiable ya que de esta forma el broker con la partición leader espera la confirmación de escritura en todas los brokers de particiones replicas antes de enviar el ACK al productor. Esto garantiza que los mensajes no se pierdan mientras uno de los brokers siga operativo, aunque hace que el procesado sea más lento.

Para la configuración de los consumidores también se hará uso de los 3 brokers disponibles, de la misma forma, emplearemos como nombre del grupo de consumidores “Grupo” y haremos uso de las siguientes opciones:

- `'bootstrap.servers': "KAFKA_BOOTSTRAP_SERVERS"`
- `'group.id': "KAFKA_CONSUMER_GROUP"`
- **Producer\_Tweets.py**

En este script se ha emplea la API de Twitter para realizar un stream de los tweets publicados, con filtro que se desee utilizar (modificar el campo `stream.filter(track = ['nuevofiltro'])`). De los campos disponibles del tweet publicado nos quedamos con los siguientes: Hora de creación, nombre de usuario, el texto del tweet y el idioma. Estos campos se guardan en un diccionario y se pasa éste al consumidor de Kafka asociando los datos al `TOPIC1 = "Tweets"`. Con codificación utf-8. Cada vez que se produzca la captura de un tweet se mostrará en pantalla el siguiente mensaje '...', esto es para corroborar el correcto funcionamiento del script.

- **Consumer\_producer.py**

En este script tenemos un consumidor y un productor, así como una función que viene del paquete de python Vader Sentiment Analysis, que nos permite analizar el contenido de los tweets ya que este paquete esta formado por un lexicon que está especialmente configurado para el analisis de sentimientos con datos de redes sociales y permite procesar los emoticonos.

Aquí se consume el `TOPIC1 = "Tweets"` y se produce información en los tópicos `TOPIC2 = "ClasiNeu"`, `TOPIC3 = "ClasiNeg"`, `TOPIC4 = "ClasiPos"`.

Primero, se hace uso de un consumidor que esta suscrito al topic donde se ha codificado la información del productor de `Productor_tweets` y que descodifica los datos en utf-8.

El siguiente paso realizado es pasar los datos obtenidos del consumidor a la función `sentiment_analyzer_scores`, que evalúa y clasifica el contenido del texto de los tweets mediante el score llamado `compount`, que contiene la información relativa a la polaridad.

Finalmente, una vez obtenido el score de polaridad haremos uso de un productor que en función del score obtenido tras analizar el sentimiento del Tweet, guardará dicho mensaje en un tópic diferente con codificación utf-8.

De esta forma:

- Si la polaridad $<0$ , se pasa el mensaje al productor que nos asocia los datos del tweet al *TOPIC3* = "*ClasiNeg*" y mostraremos el mensaje 'Produciendo tweet negativo'
- Si la polaridad $=0$ , se pasa el mensaje al productor que nos asocia los datos del tweet al *TOPIC2* = "*ClasiNeu*" y mostraremos el mensaje 'Produciendo tweet neutro'
- Si la polaridad $>0$ , se pasa el mensaje al productor que nos asocia los datos del tweet al *TOPIC4* = "*ClasiPos*" y mostraremos el mensaje 'Produciendo tweet positivo'

La razón detrás de los prints anteriores es para que se pueda observar que está en funcionamiento el script y funciona de forma adecuada.

#### ■ **Consumer\_tweets\_neu.py**

Considerando lo realizado anteriormente en el script **Consumer\_producer.py**, en este script hacemos uso de un consumidor que se suscribe al topic "*ClasiNeu*" y muestra por pantalla la información del tweet.

#### ■ **Consumer\_tweets\_pos.py**

Considerando lo realizado anteriormente en el script **Consumer\_producer.py**, en este script, hacemos uso de un consumidor que se suscribe al topic "*ClasiPos*" y muestra por pantalla la información del tweet.

#### ■ **Consumer\_tweets\_neg.py**

Considerando lo realizado anteriormente en el script **Consumer\_producer.py**, en este script, hacemos uso de un consumidor que se suscribe al topic "*ClasiNeg*" y muestra por pantalla la información del tweet.



## 2. Test de funcionamiento y test de fallo de broker

Es necesario primero iniciar Zookeeper que es el encargado de la gestión y administración de los distintos brokers, posteriormente se inician los diferentes brokers de Kafka que se vayan a emplear. Cabe destacar que tanto el inicio de Zookeeper como el de los brokers es necesario realizarlo mediante la consola de windows CMD o Ubuntu Terminal. El puerto por defecto de Zookeeper es 2181 y para un único broker, 9092, es el puerto por defecto.

Con Zookeeper en funcionamiento, así como los 3 brokers de Kafka, ahora ponemos en funcionamiento los scripts comentados anteriormente. El orden adecuado de ejecución es el siguiente:

- Primero ejecutamos el script **Producer\_Tweets.py**.
- Después ejecutamos el script **Consumer\_producer.py**
- Finalmente ejecutamos los 3 scripts, **Consumer\_tweets\_neu.py**, **Consumer\_tweets\_pos.py**, **Consumer\_tweets\_neg.py**

En las siguientes imágenes podemos ver la evolución de los tópicos una vez el código ha estado ejecutándose durante un periodo corto de tiempo:

```
C:\kafka>. \bin\windows\kafka-consumer-groups.bat --bootstrap-server localhost:9092 --describe --group Grupo
```

TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST	CLIENT-ID
ClasiNeg	0	14	14	0	rdkafka-3d27a79d-e6c4-4d8e-9eab-6405a7abc91a	/192.168.56.1	rdkafka
ClasiNeg	1	8	8	0	rdkafka-3d27a79d-e6c4-4d8e-9eab-6405a7abc91a	/192.168.56.1	rdkafka
ClasiNeu	0	10	10	0	rdkafka-345c0bb4-3e04-4a50-8c2c-473329de7d41	/192.168.56.1	rdkafka
ClasiNeu	1	15	15	0	rdkafka-345c0bb4-3e04-4a50-8c2c-473329de7d41	/192.168.56.1	rdkafka
ClasiPos	0	9	9	0	rdkafka-8143d3b7-3cae-410c-b936-64ee23670d09	/192.168.56.1	rdkafka
ClasiPos	1	12	12	0	rdkafka-8143d3b7-3cae-410c-b936-64ee23670d09	/192.168.56.1	rdkafka
Tweets	0	34	34	0	rdkafka-3cb4de18-4741-4a63-a8cc-a2eb3dea3af6	/192.168.56.1	rdkafka
Tweets	1	37	37	0	rdkafka-3cb4de18-4741-4a63-a8cc-a2eb3dea3af6	/192.168.56.1	rdkafka

Figura 7: Topics para el broker 1 al inicio

```
C:\kafka>. \bin\windows\kafka-consumer-groups.bat --bootstrap-server localhost:9092 --describe --group Grupo
```

TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST	CLIENT-ID
ClasiNeu	0	50	50	0	rdkafka-31ad570a-0d1a-47a9-96c0-37b3be5b3e41	/192.168.56.1	rdkafka
ClasiNeu	1	80	80	0	rdkafka-31ad570a-0d1a-47a9-96c0-37b3be5b3e41	/192.168.56.1	rdkafka
Tweets	0	153	153	0	rdkafka-14c50222-9a6f-4f99-850c-9e193c0a7be6	/192.168.56.1	rdkafka
Tweets	1	161	161	0	rdkafka-14c50222-9a6f-4f99-850c-9e193c0a7be6	/192.168.56.1	rdkafka
ClasiPos	0	57	57	0	rdkafka-c26a5c3b-a1c8-407e-ab2f-43b3c88296c6	/192.168.56.1	rdkafka
ClasiPos	1	43	43	0	rdkafka-c26a5c3b-a1c8-407e-ab2f-43b3c88296c6	/192.168.56.1	rdkafka
ClasiNeg	0	43	43	0	rdkafka-f53ffa56-3af4-4d3d-93dd-af8a3ca7cb97	/192.168.56.1	rdkafka
ClasiNeg	1	38	38	0	rdkafka-f53ffa56-3af4-4d3d-93dd-af8a3ca7cb97	/192.168.56.1	rdkafka

Figura 8: Evolución del broker 1, tras un periodo de ejecución

Durante el periodo de captura hacemos un test de fallo de uno de los brokers, concretamente el broker 1(puerto 9092 y broker.id= 0), en el cual se encuentran los topics originalmente y que han sido replicado en los brokers 2 y 3 (puertos 9093 y 9094).

```

C:\kafka>.\bin\windows\kafka-topics.bat --describe --zookeeper localhost:2181 --topic Tweets
Topic:Tweets      PartitionCount:2      ReplicationFactor:3      Configs:
    Topic: Tweets  Partition: 0      Leader: 0      Replicas: 0,2,1 Isr: 0,1,2
    Topic: Tweets  Partition: 1      Leader: 2      Replicas: 2,1,0 Isr: 0,1,2

C:\kafka>wmic process where "caption = 'java.exe' and commandline like '%server.properties%'" get processid
ProcessId
12876

C:\kafka>taskkill /pid 12876 /f
Correcto: se terminó el proceso con PID 12876.

C:\kafka>.\bin\windows\kafka-topics.bat --describe --zookeeper localhost:2181 --topic Tweets
Topic:Tweets      PartitionCount:2      ReplicationFactor:3      Configs:
    Topic: Tweets  Partition: 0      Leader: 2      Replicas: 0,2,1 Isr: 1,2
    Topic: Tweets  Partition: 1      Leader: 2      Replicas: 2,1,0 Isr: 0,1,2

```

Figura 9: Test de fallo de un broker

Con el comando `kafka-topics.bat --describe --zookeeper localhost:2181 --topic tweets`, mostramos la información relente sobre el topic que hayamos escogido:

- Partition: Describe el número de particiones de cada topic.
- Leader: Es el broker responsable de todas las lecturas y escrituras de cada partición. Indica en que broker se encuentra la replica leader para cada una de las particiones. En nuestro caso, tenemos que para el topic Tweets la replica leader de la partición 0 se encuentra en el broker 0. En cambio, para la partición 1 se encuentra en el broker 2.
- Replicas: Es la lista de brokers que replican el registro de ésta en cada partición, independientemente de si son el “leader” o incluso si están actualmente levantados.
- Isr: Conjunto de réplicas sincronizadas que están activas y dependen del líder, según la partición. Es decir, brokers sincronizados en los que la información está replicada. Como la información esta replicada y sincronizada en los distintos brokers.

Tras esto ejecutamos el comando `wmic process where “caption = 'java.exe' and commandline like %server.properties%” get proccessid`, que nos dice el identificador del proceso del primer broker de Kafka que tenemos corriendo. Y, posteriormente, ejecutamos el comando `taskkill \pid "processid" \f` que nos cierra por consola el broker con `broker.id= 0`.

Una vez cerrado broker 0, para el tópic Tweets se observa en la imagen como el broker para la replica líder de la partición 0 ha pasado del 0 al 2. Además, este ya no se encuentra en el conjunto de replicas activas para dicha partición, Isr: 1,2.

Finalmente, tras volver a levantar el broker 0, dejamos corriendo durante un tiempo prolongado todos los scripts. Mostramos, ahora, la descripción final de los tópicos que contiene cada uno de los brokers:



```
C:\kafka>.bin\windows\kafka-consumer-groups.bat --bootstrap-server localhost:9092 --describe --group Grupo
```

TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST	CLIENT-ID
ClasiNeu	0	118	118	0	rdkafka-aea3105e-32df-4ff4-ab19-f721a8dce211	/192.168.56.1	rdkafka
ClasiNeu	1	137	137	0	rdkafka-aea3105e-32df-4ff4-ab19-f721a8dce211	/192.168.56.1	rdkafka
ClasiNeg	0	65	65	0	rdkafka-dad01f0a-1079-4c4a-9895-839736e8234c	/192.168.56.1	rdkafka
ClasiNeg	1	75	75	0	rdkafka-dad01f0a-1079-4c4a-9895-839736e8234c	/192.168.56.1	rdkafka
Tweets	0	298	298	0	rdkafka-bf695dbb-d33d-43c4-827f-b838ea10110a	/192.168.56.1	rdkafka
Tweets	1	296	296	0	rdkafka-bf695dbb-d33d-43c4-827f-b838ea10110a	/192.168.56.1	rdkafka
ClasiPos	0	101	101	0	rdkafka-dcf2ee8c-d622-437f-aaf8-c279f1588799	/192.168.56.1	rdkafka
ClasiPos	1	95	95	0	rdkafka-dcf2ee8c-d622-437f-aaf8-c279f1588799	/192.168.56.1	rdkafka

```
C:\kafka>.bin\windows\kafka-consumer-groups.bat --bootstrap-server localhost:9093 --describe --group Grupo
```

TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST	CLIENT-ID
ClasiNeu	0	119	119	0	rdkafka-aea3105e-32df-4ff4-ab19-f721a8dce211	/192.168.56.1	rdkafka
ClasiNeu	1	138	138	0	rdkafka-aea3105e-32df-4ff4-ab19-f721a8dce211	/192.168.56.1	rdkafka
ClasiNeg	0	65	65	0	rdkafka-dad01f0a-1079-4c4a-9895-839736e8234c	/192.168.56.1	rdkafka
ClasiNeg	1	76	76	0	rdkafka-dad01f0a-1079-4c4a-9895-839736e8234c	/192.168.56.1	rdkafka
Tweets	0	299	299	0	rdkafka-bf695dbb-d33d-43c4-827f-b838ea10110a	/192.168.56.1	rdkafka
Tweets	1	298	298	0	rdkafka-bf695dbb-d33d-43c4-827f-b838ea10110a	/192.168.56.1	rdkafka
ClasiPos	0	101	101	0	rdkafka-dcf2ee8c-d622-437f-aaf8-c279f1588799	/192.168.56.1	rdkafka
ClasiPos	1	95	95	0	rdkafka-dcf2ee8c-d622-437f-aaf8-c279f1588799	/192.168.56.1	rdkafka

```
C:\kafka>.bin\windows\kafka-consumer-groups.bat --bootstrap-server localhost:9094 --describe --group Grupo
```

TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST	CLIENT-ID
ClasiNeu	0	120	120	0	rdkafka-aea3105e-32df-4ff4-ab19-f721a8dce211	/192.168.56.1	rdkafka
ClasiNeu	1	138	138	0	rdkafka-aea3105e-32df-4ff4-ab19-f721a8dce211	/192.168.56.1	rdkafka
ClasiNeg	0	65	65	0	rdkafka-dad01f0a-1079-4c4a-9895-839736e8234c	/192.168.56.1	rdkafka
ClasiNeg	1	76	76	0	rdkafka-dad01f0a-1079-4c4a-9895-839736e8234c	/192.168.56.1	rdkafka
Tweets	0	300	300	0	rdkafka-bf695dbb-d33d-43c4-827f-b838ea10110a	/192.168.56.1	rdkafka
Tweets	1	298	299	1	rdkafka-bf695dbb-d33d-43c4-827f-b838ea10110a	/192.168.56.1	rdkafka
ClasiPos	0	101	101	0	rdkafka-dcf2ee8c-d622-437f-aaf8-c279f1588799	/192.168.56.1	rdkafka
ClasiPos	1	96	96	0	rdkafka-dcf2ee8c-d622-437f-aaf8-c279f1588799	/192.168.56.1	rdkafka

Figura 10: Descripción final de los topics por cada broker

Por último, si se deseara extender este trabajo se podría implementar, en los tres consumidores que muestran la información de los tweets por pantalla en función de su polaridad, un guardado en una base de datos como por ejemplo MongoDB. La idea sería crear una colección para cada polaridad. Así, aunque cada semana se realizase una limpieza de los tópicos por la política de eliminación de mensajes configurado en nuestros servidores, mantendríamos los datos a lo largo del tiempo. Para ello se añadirían las siguientes líneas a los consumidores de los tweets clasificados:

```
from pymongo import MongoClient
client = MongoClient('localhost:27017')
collection = client.mytopic.mytopic
collection.insert_one(message)
```

Donde message sería el mensaje que ha generado el consumer (información tweet) del tópic al cual esté suscrito.