LAB 3. Implementando un clúster con Multi-Broker.

I. INTRODUCCIÓN.

En este laboratorio trataremos de montar una arquitectura un poco más compleja en la que implementaremos un clúster de tres nodos con un broker en cada nodo.

El esquema del laboratorio que montaremos será el siguiente:

Realizar Esquema

Ilustración 1 Esquema general de un Broker en un sólo nodo.

Recursos utilizados para el laboratorio:

- 3 Máquinas virtuales Ubuntu 17.04 (2G RAM y 40 HD en thin provisioning).
- Java 1.8.0.131
- Confluent-oss-3.2.1-2.11

II. Primeros pasos. Preparando el entorno.

Lo primero que vamos a hacer es instalar java ya que es el requisito principal que deberemos cumplir para poder levantar nuestro entorno de pruebas. En mi caso he optado por instalar la variante de java de Oracle, se puede usar cualquier variante siempre que sea > 1.7.

```
# add-apt-repository ppa:webupd8team/java
# apt-get update
# apt-get install Oracle-java8-installer
```

Comprobamos que todo ha ido bien mediante:

```
# java -version
java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
```

III. Instalación inicial.

Para llevar a cabo la instalación vamos a utilizar un empaquetado de Apache Kafka creado por Confluent, empresa fundada por los desarrolladores originales. Todas las operatorias que indicamos a continuación, junto con la instalación de java deberán realizarse en los tres nodos. también podéis optar por preparar una de las máquinas y clonarla.

Si no lo tenemos creado procedemos a crear el usuario "***kafkauser"***.

```
$ sudo adduser --home /kafka kafkauser
$ sudo passwd kafkauser
```

Una vez creado el usuario, descargamos el software de Confluent:

```
$ wget http://packages.confluent.io/archive/3.2/confluent-oss-3.2.1-2.11.tar.gz
```

Una vez descargado, vamos a descomprimir el producto:

```
kafkauser@SERVER1:~$ tar xvf confluent-oss-3.2.1-2.11.tar.gz
kafkauser@SERVER1:~$ mv  confluent-3.2.1 confluent
```

Con estos sencillos pasos ya estamos preparados para comenzar a jugar.

IV. Levantando el entorno.

Suponiendo que ya tenemos todos los requisitos implementados, pasamos a realizar la implementación. Vamos a dividirla en dos partes para no liarnos: Clúster Zookeeper y Clúster Apache kafka.

Clúster ZooKeeper.

Empezamos editando en los tres nodos el fichero de configuración de ZooKeeper:

```
/kafka/confluent/etc/kafka/zookeeper.properties
```

La configuración que dejaremos en nuestro caso será la siguiente:

```
dataDir=/kafka/tmp/zookeeper
clientPort=2181
maxClientCnxns=0
server.1=192.168.175.180:2888:3888
server.2=192.168.175.181:2888:3888
server.3=192.168.175.182:2888:3888
initLimit=5
syncLimit=2
```

Básicamente le indicamos a ZooKeeper que servidores formarán parte del clúster y redirigimos el directorio de trabajo a la carpeta de ***/kafka/tmp/zookeeper***.

Aparecen también dos nuevos parámetros:

- ***initLimit***. Es el tiempo de espera que utilizaran los nodos de ZooKeeper para conectarse a un líder.
- ***syncLimit***. Es el tiempo límite de desfase que puede un nodo respecto al líder.

Podemos usar otra variable "***tickTime***" para definir la unidad. Por defecto un tick es 2000 milisegundos. Siguiendo esto hemos definido 10 segundos para initLimit y 4 segundos para el syncimit.

El rango 2888:3888 será el que utilicen los nodos para comunicarse.

Ahora deberemos configurar los ***"myid"***, de cada ZooKeeper. Este parámetro sirve de identificador único para cada servidor. En nuestro caso los dejaremos asi:

192.168.175.180
mkdir -p /kafka/tmp/zookeeper/
echo "1" > /kafka/tmp/zookeeper/myid

192.168.175.181
mkdir -p /kafka/tmp/zookeeper/
echo "2" > /kafka/tmp/zookeeper/myid

192.168.175.182
mkdir -p /kafka/tmp/zookeeper/
echo "3" > /kafka/tmp/zookeeper/myid

Con todo configurado pasamos a levantar los tres nodos:

NODO1(192.168.175.180)

kafkauser@SERVER1:~/confluent$ ./bin/zookeeper-server-start -daemon ./etc/kafka/zookeeper.properties
kafkauser@SERVER1:~/confluent$ echo "ruok"|nc localhost 2181
imok

NODO2(192.168.175.180)

kafkauser@SERVER2:~/confluent$ ./bin/zookeeper-server-start -daemon ./etc/kafka/zookeeper.properties
kafkauser@SERVER2:~/confluent$ echo "ruok"|nc localhost 2181
imok

NODO3(192.168.175.180)

kafkauser@SERVER3:~/confluent$ ./bin/zookeeper-server-start -daemon ./etc/kafka/zookeeper.properties
kafkauser@SERVER3:~/confluent$ echo "ruok"|nc localhost 2181
imok

En este punto podemos usar la herramienta "zktop.py" para conectarnos a los tres nodos y ver lo que está pasando. De momento no veremos mucho ya que no tenemos kafka conectado a ellos. Más adelante veremos la diferencia.

./zktop.py --servers=192.168.175.180:2181,192.168.175.181:2181,192.168.175.182:2181

```
Ensemble -- nodecount:4 zxid:0x200000000 sessions:3

ID SERVER           PORT M    OUTST    RECVD     SENT CONNS MINLAT AVGLAT MAXLAT
0  192.168.175.180  2181 F        0       28       27     1      0      0      0
1  192.168.175.181  2181 L        0       29       28     1      0      0      0
2  192.168.175.182  2181 F        0       29       28     1      0      0      0

CLIENT           PORT S I   QUEUED    RECVD     SENT
192.168.175.180 41330 0 0        0        1        0
192.168.175.180 52624 1 0        0        1        0
192.168.175.180 34078 2 0        0        1        0
```

Ilustración 2 zktop.py contra el clúster de ZooKeeper.

Clúster Apache Kafka.

Una vez configurado el clúster de ZooKeeper pasamos a configurar el clúster de Apache kafka. Lo primero que vamos a realizar es editar el archivo de configuración:

kafkauser@SERVER1:~/confluent$ vi ./etc/kafka/server.properties

Para nuestras necesidades básicas dejaremos la siguiente configuración:

```
broker.id=0
delete.topic.enable=true
num.network.threads=3
num.io.threads=8
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
log.dirs=/kafka/tmp/kafka-logs
num.partitions=1
num.recovery.threads.per.data.dir=1
log.retention.hours=168
log.segment.bytes=1073741824
log.retention.check.interval.ms=300000
zookeeper.connect=192.168.175.180:2181,192.168.175.181:2181,192.168.175.182:2181
zookeeper.connection.timeout.ms=6000
confluent.support.metrics.enable=true
confluent.support.customer.id=anonymous
```

El parámetro ***broker.id*** deberá ser configurado en cada nodo. En nuestro caso:

NODO1(192.168.175.180) broker.id=0
NODO1(192.168.175.181) broker.id=1
NODO1(192.168.175.182) broker.id=2

La mayoría del resto de parámetros lo hemos dejado por defecto, pero hemos configurado las siguientes:

- ***Delete.topic.enable=true***. Permite borrar Topics del broker. Lo hemos habilitado para hacer más ágil la administración, pero en entornos cerrador no sería conveniente ya que desde programa nos las pueden borrar, e incluso si levantamos el acceso rest, nos las pueden borrar vía una petición simple. Eso sí porque no tenemos claves de acceso a las colas.

- ***Log.dirs***. Lugar donde se guarda el fichero de persistencia de Apache Kafka. Interesante saber dónde está porque podemos hacer un dump para ver el contenido.

- ***Zookeeper.connect***. Referencia al clúster de Zookeeper que hemos configurado.

Los parámetros relacionados con la retención de los mensajes en las Topic y demás los dejaremos para futuros laboratorios.

Una vez configurado en los tres equipos podemos levantar el clúster mediante:

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-server-start -daemon ./etc/kafka/server.properties

kafkauser@SERVER2:~/confluent$ ./bin/kafka-server-start -daemon ./etc/kafka/server.properties

kafkauser@SERVER3:~/confluent$ ./bin/kafka-server-start -daemon ./etc/kafka/server.properties
```

Con esto ya tendremos nuestro entorno levantado para empezar a jugar.

V. Probando nuestro nuevo clúster.

Vamos a probar el funcionamiento del clúster. Como el en los LABs anteriores empezaremos creando una Topic y haciendo pruebas de alta disponibilidad. Hay que resaltar que el diseño de las Topic con sus particiones y réplicas es un tema complejo que no abordaremos en este LAB pero que debemos saber que no es trivial. Para hacer nuestras pruebas, vamos a crear una cola TestLAB3 con replicación 3 y 1 partición.

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-topics --create --zookeeper 192.168.175.180:2181,192.168.175.181:2181,192.168.175.182:2181 --replication-factor 3 --partition 1 --topic TestLAB3
Created topic "TestLAB3".
```

Vemos como se ha creado con:

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-topics --describe --zookeeper 192.168.175.180:2181,192.168.175.181:2181,192.168.175.182:2181 --topic TestLAB3
Topic:TestLAB3  PartitionCount:1  ReplicationFactor:3  Configs:
        Topic: TestLAB3  Partition: 0  Leader: 2  Replicas: 2,3,1  Isr: 2,3,1
```

El número de particiones lo hemos puesto a 1 y determinará el paralelismo que se puede alcanzar en el lado del consumidor. Por esta razón, deberemos escoger el número de particiones en base a la forma en que nuestros datos serán consumidos.

Para hacer las pruebas del clúster vamos a cambiar un poco el método de los otros LAB y vamos utilizar herramientas para simular carga y analizar un poco el rendimiento.

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-producer-perf-test --num-records 1000000 --record-size 100 --topic TestLAB3 --throughput 10000 --producer-props bootstrap.servers=192.168.175.180:9092,192.168.175.181:9092,192.168.175.182:9092 max.in.flight.requests.per.connection=1 batch.size=10000
49842 records sent, 9966,4 records/sec (0,95 MB/sec), 19,9 ms avg latency, 177,0 max latency.
50210 records sent, 10042,0 records/sec (0,96 MB/sec), 3,6 ms avg latency, 52,0 max latency.
50300 records sent, 10060,0 records/sec (0,96 MB/sec), 3,6 ms avg latency, 53,0 max latency.
50000 records sent, 10000,0 records/sec (0,95 MB/sec), 2,8 ms avg latency, 65,0 max latency.
49594 records sent, 9918,8 records/sec (0,95 MB/sec), 4,8 ms avg latency, 74,0 max latency.
50466 records sent, 10093,2 records/sec (0,96 MB/sec), 6,8 ms avg latency, 116,0 max latency.
50000 records sent, 10000,0 records/sec (0,95 MB/sec), 2,2 ms avg latency, 46,0 max latency.
50020 records sent, 10002,0 records/sec (0,95 MB/sec), 1,4 ms avg latency, 16,0 max latency.
50030 records sent, 10004,0 records/sec (0,95 MB/sec), 2,3 ms avg latency, 63,0 max latency.
50000 records sent, 10000,0 records/sec (0,95 MB/sec), 3,2 ms avg latency, 66,0 max latency.
50020 records sent, 10004,0 records/sec (0,95 MB/sec), 1,4 ms avg latency, 16,0 max latency.
50010 records sent, 10000,0 records/sec (0,95 MB/sec), 2,1 ms avg latency, 62,0 max latency.
50000 records sent, 10000,0 records/sec (0,95 MB/sec), 1,7 ms avg latency, 19,0 max latency.
50060 records sent, 9996,0 records/sec (0,95 MB/sec), 1,7 ms avg latency, 24,0 max latency.
50090 records sent, 10018,0 records/sec (0,96 MB/sec), 2,1 ms avg latency, 64,0 max latency.
50020 records sent, 10004,0 records/sec (0,95 MB/sec), 1,4 ms avg latency, 16,0 max latency.
50040 records sent, 10008,0 records/sec (0,95 MB/sec), 1,6 ms avg latency, 21,0 max latency.
50010 records sent, 10002,0 records/sec (0,95 MB/sec), 1,7 ms avg latency, 38,0 max latency.
50040 records sent, 10006,0 records/sec (0,95 MB/sec), 1,4 ms avg latency, 21,0 max latency.
1000000 records sent, 9998,500225 records/sec (0,95 MB/sec), 3,37 ms avg latency, 177,00 ms max latency, 1 ms 50th, 8 ms 95th, 54 ms 99th, 139 ms 99.9th.
```

Recogeremos los datos con el consumidor:

```
kafkauser@SERVER3:~/confluent$ ./bin/kafka-console-consumer --bootstrap-server 192.168.175.180:9092,192.168.175.181:9092,192.168.175.182:9092 --topic TestLAB3
……..
SSXVNJHPDQDXVCRASTVYBCWVMGNYKRXVZXKGX
TSPSJDGYLUEGQFLAQLOCFLJBEPOWFNSOMYARHAO
PUFOJHHDXEHXJBHWGSMZJGNL
SSXVNJHPDQDXVCRASTVYBCWVMGNYKRXVZXKGX
……..
```


Después de la prueba podemos comprobar el offset del consumidor mediante:

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-run-class kafka.tools.GetOffsetShell --broker-list 192.168.175.180:9092,192.168.175.181:9092,192.168.175.182:9092 -topic TestLAB3
TestLAB3:0:1000000
```

Hasta aquí el LAB. Más adelante veremos cómo funcionan los consumidores y herramientas interesantes para ver que esta pasando.