# LAB 1. Montando un nodo con un Broker.

## I. INTRODUCCIÓN.

En este laboratorio trataremos de montar una arquitectura simple compuesta de un solo nodo con un único Broker. Nos servirá de base para ver en la práctica algunos de los conceptos teóricos explicados durante la teoría.

El esquema del laboratorio que montaremos será el siguiente:

***Recursos utilizados para el laboratorio***:

- Máquina virtual 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 de Kafka. Veremos más adelante que nos proporciona una serie de ventajas e integración con diferentes servicios y conectores. El software lo podemos encontrar en el siguiente enlace:

https://www.confluent.io/

La instalación se puede llevar a cabo de varias formas:

- Utilizando paquetería propia del sistema operativo.
- Utilizando el software descomprimido sin integración con el sistema operativo.

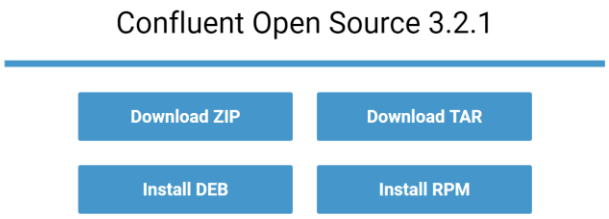


*Ilustración 1 Posibilidades de descarga de la suite Confluent.*

Nosotros vamos a utilizar la segunda opción ya que así conseguimos desligar el producto del sistema operativo. Permitiéndonos migrar de máquina simplemente copiando la carpeta. Eso sí, asegurándonos que existe java en la máquina de destino.

De todas formas, para los que quieran probar la instalación desde paquetería os dejo los pasos a ejecutar:

http://docs.confluent.io/current/installation.html#installation

```
$ wget -qO - http://packages.confluent.io/deb/3.2/archive.key | sudo apt-key add -
$ sudo add-apt-repository "deb [arch=amd64] http://packages.confluent.io/deb/3.2 stable main"
$ sudo apt-get update && sudo apt-get install confluent-platform-oss-2.11
```

Al optar por la segunda opción de instalación creemos conveniente la creación de un usuario para la gestión del producto:

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

Una vez creado el usuario podemos cambiarnos a él y descargar el software:

```
$ 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
```

Antes de continuar vamos a entender un poco la estructura del producto descomprimido. El esquema general es el siguiente:

```
confluent
├── bin
├── etc
│   ├── camus
│   ├── confluent-common
│   ├── kafka
│   ├── kafka-connect-elasticsearch
│   ├── kafka-connect-hdfs
│   ├── kafka-connect-jdbc
│   ├── kafka-connect-s3
│   ├── kafka-connect-storage-common
│   ├── kafka-rest
│   ├── rest-utils
│   └── schema-registry
├── logs
├── share
│   ├── confluent-common
│   ├── doc
│   │   ├── camus
│   │   ├── confluent-common
│   │   ├── kafka
│   │   │   └── confluent-support-metrics-client
│   │   ├── kafka-connect-elasticsearch
│   │   ├── kafka-connect-hdfs
│   │   ├── kafka-connect-jdbc
│   │   ├── kafka-connect-s3
│   │   ├── kafka-connect-storage-common
│   │   ├── kafka-rest
│   │   ├── kafka-serde-tools
│   │   ├── rest-utils
│   │   └── schema-registry
│   ├── java
│   │   ├── camus
│   │   ├── confluent-common
│   │   ├── kafka
│   │   ├── kafka-connect-elasticsearch
│   │   ├── kafka-connect-hdfs
│   │   ├── kafka-connect-jdbc
│   │   ├── kafka-connect-s3
│   │   ├── kafka-connect-storage-common
│   │   ├── kafka-rest
│   │   ├── kafka-serde-tools
│   │   ├── rest-utils
│   │   └── schema-registry
│   ├── kafka-rest
│   └── rest-utils
└── src
```

*Ilustración 2 Estructura de directorios Confluent Plataform.*

- ***bin.*** Encontramos todos los ejecutables necesarios para levantar los diferentes servicios. Algunos que usaremos en este Lab son:
    - zookeeper-server-start
    - zookeeper-server-stop
    - zookeeper-shell
    - kafka-server-start
    - kafka-server-stop

- ***etc.*** Configuraciones de los diferentes componentes. En este lab nos centraremos en la carpeta kafka que contiene los principales ficheros de configuración.

```
kafka
├── connect-console-sink.properties
├── connect-console-source.properties
├── connect-distributed.properties
├── connect-file-sink.properties
├── connect-file-source.properties
├── connect-log4j.properties
├── connect-standalone.properties
├── consumer.properties
├── log4j.properties
├── producer.properties
├── server.properties
├── tools-log4j.properties
└── zookeeper.properties
```

*Ilustración 3 Contenido etc/kafka.*

- ***logs***. Contiene todos los logs de las diferentes aplicaciones.

## IV. Levantando el entorno.

Una vez realizada la sencilla preparación pasamos a levantar los diferentes productos. Primeramente, levantaremos ***Apache ZooKeeper*** que gestionará la coordinación del entorno distribuido. Como hemos decidido hacerlo todo auto contenido en nuestra carpeta deberemos crear un directorio tmp que almacenará los datos y modificar el archivo de configuración de ZooKeeper:

```
$ mkdir /kafka/tmp
$ vi etc/kafka/zookeeper.properties
```

Modificaremos la variable ***"dataDir"*** dejando el fichero de configuración así:

```
kafkauser@SERVER1:~/confluent$ cat etc/kafka/zookeeper.properties|grep -v ^#

dataDir=/kafka/tmp/zookeeper
clientPort=2181
maxClientCnxns=0
```

Ahora estamos en disposición de levantar el ZooKeeper de la siguiente forma:

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

Si todo ha ido bien veremos por pantalla algo parecido a esto:

```
[2017-05-31 10:45:19,977] INFO Server environment:java.library.path=/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:19,977] INFO Server environment:java.io.tmpdir=/tmp (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:19,977] INFO Server environment:java.compiler=<NA> (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:19,977] INFO Server environment:os.name=Linux (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:19,977] INFO Server environment:os.arch=amd64 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:19,977] INFO Server environment:os.version=4.10.0-19-generic (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:19,978] INFO Server environment:user.name=kafkauser (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:19,978] INFO Server environment:user.home=/kafka (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:19,979] INFO Server environment:user.dir=/kafka/confluent (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:19,995] INFO tickTime set to 3000 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:19,995] INFO minSessionTimeout set to -1 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:19,995] INFO maxSessionTimeout set to -1 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-05-31 10:45:20,019] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
```

*Ilustración 4 Registro parcial arranque ZooKeeper.*

Podemos comprobar que el proceso está en máquina con un simple ***ps***:

```
kafkauser@SERVER1:~$ ps -ef|grep -i zookeeper
kafkaus+  14815  14141  0 10:45 pts/0    00:00:01 /usr/lib/jvm/java-8-o
racle/bin/java -Xmx512M -Xms512M -server -XX:+UseG1GC -XX:MaxGCPauseMil
lis=20 -XX:InitiatingHeapOccupancyPercent=35 -XX:+DisableExplicitGC -Dj
ava.awt.headless=true -Xloggc:/kafka/confluent/bin/../logs/zookeeper-gc
.log -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintG
CTimeStamps -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremo
te.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -Dkafka.
logs.dir=/kafka/confluent/bin/../logs -Dlog4j.configuration=file:./bin/
../etc/kafka/log4j.properties -cp :/kafka/confluent/bin/../share/java/k
afka/*:/kafka/confluent/bin/../share/java/confluent-support-metrics/*:/
usr/share/java/confluent-support-metrics/* org.apache.zookeeper.server.
quorum.QuorumPeerMain ./etc/kafka/zookeeper.properties
```

También podemos comprobar que el proceso está escuchando en el puerto 2181:

```
kafkauser@SERVER1:~$ netstat -putan|grep 2181
(No todos los procesos pueden ser identificados, no hay información de propiedad del proceso
 no se mostrarán, necesita ser superusuario para verlos todos.)
tcp6       0      0 :::2181                 :::*                    ESCUCHAR    14815/java
```

Con estas comprobaciones nos aseguraremos de que el proceso está levantado, pero si queremos comprobar en qué estado está funcionando Zookeeper lo podemos hacer mediante:

```
kafkauser@SERVER1:~$ echo ruok|nc localhost 2181
imok
```

Si nos devuelve "imok" es que está funcionando correctamente. Esta forma de interactuar con el servidor nos abre la posibilidad de ejecutar una serie de parámetros interesantes:

- ***dump***. Enumerar las sesiones y los nodos efímeros. Esto sólo funciona en el nodo líder.
  Si lanzamos después de levantar el ZooKeeper:

  ```
  kafkauser@SERVER1:~$ echo dump|nc localhost 2181
  SessionTracker dump:
  Session Sets (0):
  ephemeral nodes dump:
  Sessions with Ephemerals (0):
  ```

  Después de levantar el proceso de Kafka veremos algo parecido a esto:

  ```
  kafkauser@SERVER1:~$ echo dump|nc localhost 2181
  SessionTracker dump:
  Session Sets (2):
  0 expire at Wed May 31 12:20:15 CEST 2017:
  1 expire at Wed May 31 12:20:18 CEST 2017:
      0x15c5dae412e0000
  ephemeral nodes dump:
  Sessions with Ephemerals (1):
  0x15c5dae412e0000:
      /controller
      /brokers/ids/0
  ```

- ***envi***. Muestra las variables de entorno.
- ***Kill***. Para el servidor de ZooKeeper. En principio esta función no funciona correctamente.
- ***reqs.*** Lista las peticiones excepcionales.
- ***ruok***. Sirve para comprobar el estado del servidor. Si responde "imok", está funcionando correctamente.
- ***srst***. Resetea las estadísticas devueltas por stat.
- ***stat***. Lista las estadísticas de rendimientos y conexiones de los clientes.

  ```
  kafkauser@SERVER1:~$ echo stat|nc localhost 2181
  Zookeeper version: 3.4.9-1757313, built on 08/23/2016 06:50 GMT
  Clients:
   /0:0:0:0:0:0:0:1:33132[0](queued=0,recved=1,sent=0)
   /127.0.0.1:54872[1](queued=0,recved=220,sent=222)

  Latency min/avg/max: 0/0/8
  Received: 221
  Sent: 222
  Connections: 2
  Outstanding: 0
  Zxid: 0x2c
  Mode: standalone
  Node count: 26
  ```

Para poder monitorizar lo que pasa en ZooKeeper en tiempo real, podemos utilizar multitud de herramientas. Una muy interesante es la siguiente:

https://github.com/phunt/zktop.git

Nos la descargamos mediante:

```
kafkauser@SERVER1:~$ git clone https://github.com/phunt/zktop.git
Clonar en «zktop»...
remote: Counting objects: 117, done.
remote: Total 117 (delta 0), reused 0 (delta 0), pack-reused 117
Receiving objects: 100% (117/117), 29.41 KiB | 0 bytes/s, done.
Resolving deltas: 100% (60/60), done.
```

O simplemente nos bajamos el fichero ***zktop.py***. Las opciones son sencillas:

```
kafkauser@SERVER1:~/zktop$ ./zktop.py --help
Usage: zktop.py [options]

Options:
  -h, --help            show this help message and exit
  --servers=SERVERS     comma separated list of host:port (default
                        localhost:2181)
  -n, --names           resolve session name from ip (default False)
  --fix_330             workaround for a bug in ZK 3.3.0
  -v VERBOSITY, --verbosity=VERBOSITY
                        log level verbosity (DEBUG, INFO, WARN(ING), ERROR,
                        CRITICAL/FATAL)
  -l LOGFILE, --logfile=LOGFILE
                        directory in which to place log file, or empty for
                        none
  -c CONFIGFILE, --config=CONFIGFILE
                        zookeeper configuration file to lookup servers from
  -t TIMEOUT, --timeout=TIMEOUT
                        connection timeout to zookeeper instance
```

*Ilustración 5 opciones zktop.py*

Para lanzar el monitor usaremos algo parecido a esto:

```
kafkauser@SERVER1:~/zktop$ ./zktop.py --servers=localhost:2181
```

```
Ensemble -- nodecount:26 zxid:0x2c sessions:2

ID SERVER          PORT M    OUTST     RECVD      SENT CONNS MINLAT AVGLAT MAXLAT
0  localhost       2181 S        0       627       628     2      0      0      8

CLIENT           PORT S I   QUEUED    RECVD      SENT
127.0.0.1       54890 0 0        0        1         0
127.0.0.1       54872 0 1        0      621       623
```

*Ilustración 6 zktop.py console*

Quedan muchas cosas pendientes sobre ZooKeeper, pero las iremos viendo en los siguientes laboratorios. Para terminar, vemos como arrancar ZooKeeper en modo demonio:

$ ./bin/zookeeper-server-start -daemon ./etc/kafka/zookeeper.properties

Una vez levantado ZooKeeper pasamos a levantar el ***Apache kafka***. Antes de hacerlo deberemos editar el fichero de configuración para que apunte al directorio "*tmp"*. Para ello:

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

Y editamos la variable ***"log.dirs"*** para que apunte a nuestro directorio:

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

```
############################ Log Basics ############################

# A comma seperated list of directories under which to store log files
log.dirs=/kafka/tmp/kafka-logs
```

Una vez editado pasamos a arrancar Apache Kafka de la siguiente forma:

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

Si lo queremos arrancar en modo demonio:

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

En el arranque nos mostrará todas las configuraciones por defecto. Las pongo sólo como curiosidad ya que abordar todos los parámetros excede el propósito de este documento. Pero es bueno que sepamos que existen por si las tenemos que modificar. Las encontramos en APENDICE I.

Para comprobar que se ha levantado correctamente valdrá con ejecutar:

$ ps -ef|grep server.properties|grep -v grep

```
kafkauser@SERVER1:~/tmp$ ps -ef|grep server.properties|grep -v grep
kafkaus+  15636  14996  1 13:34 pts/1    00:00:08 /usr/lib/jvm/java-8-oracle/bin/ja
va -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOcc
upancyPercent=35 -XX:+DisableExplicitGC -Djava.awt.headless=true -Xloggc:/kafka/con
fluent/bin/../logs/kafkaServer-gc.log -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCD
ateStamps -XX:+PrintGCTimeStamps -Dcom.sun.management.jmxremote -Dcom.sun.managemen
t.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -Dkafka.log
s.dir=/kafka/confluent/bin/../logs -Dlog4j.configuration=file:./bin/../etc/kafka/lo
g4j.properties -cp :/kafka/confluent/bin/../share/java/kafka/*:/kafka/confluent/bin
/../share/java/confluent-support-metrics/*:/usr/share/java/confluent-support-metric
s/* io.confluent.support.metrics.SupportedKafka ./etc/kafka/server.properties
kafkauser@SERVER1:~/tmp$
```

También podemos ver que el puerto por defecto, 9092, está levantado:

kafkauser@SERVER1:~/tmp$ netstat -putan|grep 9092

```
kafkauser@SERVER1:~/tmp$ netstat -putan|grep 9092
(No todos los procesos pueden ser identificados, no hay información de propiedad del proceso
 no se mostrarán, necesita ser superusuario para verlos todos.)
tcp6       0      0 :::9092                 :::*                    ESCUCHAR    15636/java
tcp6       1      0 127.0.0.1:53314         127.0.0.1:9092          CLOSE_WAIT  15636/java
```

El script de arranque de Kafka es muy sencillo;

```
if [ $# -lt 1 ];
then
	echo "USAGE: $0 [-daemon] server.properties [--override property=value]*"
	exit 1
fi
base_dir=$(dirname $0)

if [ "x$KAFKA_LOG4J_OPTS" = "x" ]; then

LOG4J_CONFIG_NORMAL_INSTALL="/etc/kafka/log4j.properties"

LOG4J_CONFIG_ZIP_INSTALL="$base_dir/../etc/kafka/log4j.properties"
 if [ -e "$LOG4J_CONFIG_NORMAL_INSTALL" ]; then # Normal install layout
   KAFKA_LOG4J_OPTS="-Dlog4j.configuration=file:${LOG4J_CONFIG_NORMAL_INSTALL}"
 elif [ -e "${LOG4J_CONFIG_ZIP_INSTALL}" ]; then # Simple zip file layout
   KAFKA_LOG4J_OPTS="-Dlog4j.configuration=file:${LOG4J_CONFIG_ZIP_INSTALL}"
 else # Fallback to normal default
   KAFKA_LOG4J_OPTS="-Dlog4j.configuration=file:$base_dir/../config/log4j.properties"
 fi
fi
export KAFKA_LOG4J_OPTS

if [ "x$KAFKA_HEAP_OPTS" = "x" ]; then
   export KAFKA_HEAP_OPTS="-Xmx1G -Xms1G"
fi

EXTRA_ARGS=${EXTRA_ARGS-'-name kafkaServer -loggc'}

COMMAND=$1
case $COMMAND in
 -daemon)
   EXTRA_ARGS="-daemon "$EXTRA_ARGS
   shift
   ;;
 *)
   ;;
esac

exec $base_dir/kafka-run-class $EXTRA_ARGS io.confluent.support.metrics.SupportedKafka "$@"
```

Podemos destacar la variable ***KAFKA_HEAP_ARGS***, donde podremos modificar los parámetros de memoria de la máquina virtual de java.

KAFKA_HEAP_OPTS="-Xmx1G -Xms1G"

Para resumir un poco, para levantar el entorno bastará con ejecutar:

```
$ ./bin/zookeeper-server-start -daemon ./etc/kafka/zookeeper.properties

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

Y ya tendremos nuestro entorno de un nodo con un broker levantado.

## V. Jugando con el entorno. Primeras pruebas.

Ya tenemos todo levantado, así que pasamos a hacer una prueba simple. Para ello ejecutaremos los siguientes pasos:

1. Crearemos unas nueva Topic llamada "TestLAB1".
2. Lanzaremos el productor de prueba para insertar mensajes en la Topic.
3. Lanzaremos un consumidor de prueba para recoger los mensajes.

Para comenzar vamos a crear una Topic nueva llamada "TestLAB1" con una sola partición y una réplica. El tema del diseño de particiones y replicas es un tema complejo que abordaremos en otros laboratorios. Para que nos vaya sonando, el ***número de particiones*** determinará el paralelismo que se puede alcanzar en el lado del consumidor y el ***factor de replicación*** determinará el número de réplicas de la Topic presentes en el clúster. Como máximo sólo puede existir una por broker. En nuestro ejemplo podríamos tener un factor de replicación de uno.

Para crear la primera Topic ejecutaremos:

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-topics --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic TestLAB1
Created topic "TestLAB1"
```

Si vemos el log de Apache Kafka veremos algo parecido a esto:

```
[2017-05-31 14:20:56,724] INFO [ReplicaFetcherManager on broker 0] Removed fetcher for partitions TestLAB1-0 (kafka.server.ReplicaFetcherManager)
[2017-05-31 14:20:56,728] INFO Completed load of log TestLAB1-0 with 1 log segments and log end offset 0 in 1 ms (kafka.log.Log)
[2017-05-31 14:20:56,733] INFO Created log for partition [TestLAB1,0] in /kafka/tmp/kafka-logs with properties {compression.type -> producer, message.format.version -> 0.10.2-IV0, file.delete.delay.ms -> 60000, max.message.bytes -> 1000012, min.compaction.lag.ms -> 0, message.timestamp.type -> CreateTime, min.insync.replicas -> 1, segment.jitter.ms -> 0, preallocate -> false, min.cleanable.dirty.ratio -> 0.5, index.interval.bytes -> 4096, unclean.leader.election.enable -> true, retention.bytes -> -1, delete.retention.ms -> 86400000, cleanup.policy -> [delete], flush.ms -> 9223372036854775807, segment.ms -> 604800000, segment.bytes -> 1073741824, retention.ms -> 604800000, message.timestamp.difference.max.ms -> 9223372036854775807, segment.index.bytes -> 10485760, flush.messages -> 9223372036854775807}. (kafka.log.LogManager)
[2017-05-31 14:20:56,735] INFO Partition [TestLAB1,0] on broker 0: No checkpointed highwatermark is found for partition TestLAB1-0 (kafka.cluster.Partition)
```

Es muy interesante porque vemos los parámetros por defecto que está cogiendo y como está creando los logs de particiones en el disco. El resultado lo podemos ver de la siguiente forma:

```
kafkauser@SERVER1:~/tmp/kafka-logs$ tree
.
├── cleaner-offset-checkpoint
├── meta.properties
├── recovery-point-offset-checkpoint
├── replication-offset-checkpoint
└── TestLAB-0
    ├── 00000000000000000000.index
    ├── 00000000000000000000.log
    └── 00000000000000000000.timeindex
```

*Ilustración 7 Contendido /kafka/tmp/kafka-logs*

Podemos comprobar que la Topic se ha creado correctamente mediante:

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-topics --list --zookeeper localhost:2181
TestLAB
```

Y ver sus especificaciones mediante:

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-topics --describe --zookeeper localhost:2181 -topic TestLAB
Topic:TestLAB PartitionCount:1 ReplicationFactor:1
        Configs:
        Topic: TestLAB Partition: 0 Leader: 0
        Replicas: 0 Isr: 0
```

En la salida podemos ver algunos parámetros interesantes:

- ***"Leader"*** es el nodo responsable de todas las lecturas y escrituras para la partición dada. Cada nodo será el líder de una parte seleccionada al azar de las particiones.
- ***"Réplicas"*** es la lista de nodos que replican el registro para esta partición, independientemente de si son el líder o incluso si están vivos actualmente.
- ***"ISR"*** es el conjunto de réplicas "en sincronía". Este es el subconjunto de la lista de réplicas que se encuentra actualmente vivas.



Ahora que ya tenemos nuestra primera Topic creada vamos a levantar un productor de prueba para poder simular el envío de mensajes a "TestLAB".
Para ello bastará con ejecutar los siguientes comandos:

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-console-producer --broker-list localhost:9092 -topic TestLAB
```

Una vez levantado podemos empezar a mandar mensajes a la Topic:

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-console-producer
Prueba1
Prueba2
23
Prueba 3
```

Ahora falta recogerlos, para ello levantaremos un sencillo consumidor de la siguiente forma:

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-console-consumer --zookeeper localhost:2181 --topic TestLAB --from-beginning
```

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-console-consumer --zookeeper localhost
:2181 --topic TestLAB --from-beginning
Using the ConsoleConsumer with old consumer is deprecated and will be removed in
a future major release. Consider using the new consumer by passing [bootstrap-ser
ver] instead of [zookeeper].
Prueba1
Prueba2
23
Prueba 3
```

Hemos añadido el parámetro ***--from-begining*** para que lea todos los mensajes desde el principio. Si paramos el consumidor y volvemos a lanzar sin ese parámetro vemos que permanece a la espera ya que el offset ya se ha desplazado hasta la última posición. Si añadimos un nuevo mensaje "ultimo", lo recibirá sin problemas.

```
kafkauser@SERVER1: ~/confluent 81x8
kafkauser@SERVER1:~/confluent$ ./bin/kafka-console-producer --broker-list localho
Prueba1
Prueba2
23
Prueba 3
ultimo

kafkauser@SERVER1: ~/confluent 81x16
^CProcessed a total of 4 messages
kafkauser@SERVER1:~/confluent$ clear

kafkauser@SERVER1:~/confluent$ ./bin/kafka-console-consumer --zookeeper localhost
:2181 --topic TestLAB
Using the ConsoleConsumer with old consumer is deprecated and will be removed in
a future major release. Consider using the new consumer by passing [bootstrap-ser
ver] instead of [zookeeper].
ultimo
```

*Ilustración 8 Pruebas consumidor-productor.*

Ahora bien, si queremos poder ver el contenido de una cola sin sacar los mensajes con un consumidor podemos usar las siguientes herramientas:

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-run-class kafka.tools.DumpLogSegments --deep-iteration --files /kafka/tmp/kafka-logs/TestLAB-0/00000000000000000000.log --print-data-log
Dumping /kafka/tmp/kafka-logs/TestLAB-0/00000000000000000000.log
Starting offset: 0
offset: 0 position: 0 CreateTime: 1496235313452 isvalid: true payloadsize: 7 magic: 1 compresscodec: NONE crc: 3090365568 payload: Prueba1
offset: 1 position: 41 CreateTime: 1496235318421 isvalid: true payloadsize: 7 magic: 1 compresscodec: NONE crc: 2992356866 payload: Prueba2
offset: 2 position: 82 CreateTime: 1496235320137 isvalid: true payloadsize: 2 magic: 1 compresscodec: NONE crc: 3313128816 payload: 23
offset: 3 position: 118 CreateTime: 1496235324114 isvalid: true payloadsize: 8 magic: 1 compresscodec: NONE crc: 496809997 payload: Prueba 3
offset: 4 position: 160 CreateTime: 1496235773815 isvalid: true payloadsize: 6 magic: 1 compresscodec: NONE crc: 486099406 payload: ultimo
```

A partir de aquí, lo que nos queda es jugar con las diferentes opciones para ver cómo se comporta el entorno. Para resumir un poco lo que hemos hecho en esta sencilla pruebas es lo siguiente:

***Crear una Topic nueva***

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-topics --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic TestLAB1
```

***Ver las características de la nueva Topic.***

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-topics --describe --zookeeper localhost:2181 -topic TestLAB
```

***Lanzar el productor***

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-console-producer --broker-list localhost:9092 -topic TestLAB
```

***Lanzar el consumidor***

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-console-consumer --zookeeper localhost:2181 --topic TestLAB --from-beginning
```

***Leemos los datos directamente del log de la Topic.***

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-run-class kafka.tools.DumpLogSegments --deep-iteration --files /kafka/tmp/kafka-logs/TestLAB-0/00000000000000000000.log --print-data-log
```

Para terminar este primer laboratorio vamos a ver como se borra una Topic del Broker. Para poder realizarlo deberemos añadir un parámetro a nuestro servidor Kafka, ya que por defecto viene con protección para borrados. El parámetro que debemos añadir es el siguiente:

```
delete.topic.enable=true
```

Una vez reiniciado el servidor de kafka, pasamos a borrar la Topic de pruebas TestLAB.

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-topics --zookeeper localhost:2181 --list
TestLAB
__confluent.support.metrics

kafkauser@SERVER1:~/confluent$ ./bin/kafka-topics --zookeeper localhost:2181 --delete --topic TestLAB
Topic TestLAB is marked for deletion.
```



Note: This will have no impact if delete.topic.enable is not set to true.

Comprobamos y vemos que la Topic se ha borrado correctamente.

```
kafkauser@SERVER1:~/confluent$ ./bin/kafka-topics --zookeeper localhost:2181 --list
__confluent.support.metrics
```

Podemos ver también que ha borrado los ficheros de la carpeta *tmp*.

```
kafkauser@SERVER1:~/tmp/kafka-logs$ tree
.
├── cleaner-offset-checkpoint
├── __confluent.support.metrics-0
│   ├── 00000000000000000000.index
│   ├── 00000000000000000000.log
│   └── 00000000000000000000.timeindex
├── meta.properties
├── recovery-point-offset-checkpoint
└── replication-offset-checkpoint
```

## VI. APENDICE I. KafkaConfig values.

```
[2017-05-31 13:34:45,956] INFO KafkaConfig values:
        advertised.host.name = null
        advertised.listeners = null
        advertised.port = null
        authorizer.class.name =
        auto.create.topics.enable = true
        auto.leader.rebalance.enable = true
        background.threads = 10
        broker.id = 0
        broker.id.generation.enable = true
        broker.rack = null
        compression.type = producer
        connections.max.idle.ms = 600000
        controlled.shutdown.enable = true
        controlled.shutdown.max.retries = 3
        controlled.shutdown.retry.backoff.ms = 5000
        controller.socket.timeout.ms = 30000
        create.topic.policy.class.name = null
        default.replication.factor = 1
        delete.topic.enable = false
        fetch.purgatory.purge.interval.requests = 1000
        group.max.session.timeout.ms = 300000
        group.min.session.timeout.ms = 6000
        host.name =
        inter.broker.listener.name = null
        inter.broker.protocol.version = 0.10.2-IV0
        leader.imbalance.check.interval.seconds = 300
        leader.imbalance.per.broker.percentage = 10
        listener.security.protocol.map = SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,TRACE:TRACE,SASL_SSL:SASL_SSL,PLAINTEXT:PLAINTEXT
        listeners = null
        log.cleaner.backoff.ms = 15000
        log.cleaner.dedupe.buffer.size = 134217728
        log.cleaner.delete.retention.ms = 86400000
        log.cleaner.enable = true
        log.cleaner.io.buffer.load.factor = 0.9
        log.cleaner.io.buffer.size = 524288
        log.cleaner.io.max.bytes.per.second = 1.7976931348623157E308
        log.cleaner.min.cleanable.ratio = 0.5
        log.cleaner.min.compaction.lag.ms = 0
        log.cleaner.threads = 1
        log.cleanup.policy = [delete]
        log.dir = /tmp/kafka-logs
        log.dirs = /kafka/tmp/kafka-logs
        log.flush.interval.messages = 9223372036854775807
        log.flush.interval.ms = null
        log.flush.offset.checkpoint.interval.ms = 60000
        log.flush.scheduler.interval.ms = 9223372036854775807
        log.index.interval.bytes = 4096
        log.index.size.max.bytes = 10485760
        log.message.format.version = 0.10.2-IV0
        log.message.timestamp.difference.max.ms = 9223372036854775807
        log.message.timestamp.type = CreateTime
        log.preallocate = false
        log.retention.bytes = -1
        log.retention.check.interval.ms = 300000
        log.retention.hours = 168
        log.retention.minutes = null
        log.retention.ms = null
        log.roll.hours = 168
        log.roll.jitter.hours = 0
        log.roll.jitter.ms = null
        log.roll.ms = null
        log.segment.bytes = 1073741824
        log.segment.delete.delay.ms = 60000
        max.connections.per.ip = 2147483647
        max.connections.per.ip.overrides =
        message.max.bytes = 1000012
        metric.reporters = []
        metrics.num.samples = 2
        metrics.recording.level = INFO
        metrics.sample.window.ms = 30000
        min.insync.replicas = 1
        num.io.threads = 8
        num.network.threads = 3
        num.partitions = 1
        num.recovery.threads.per.data.dir = 1
        num.replica.fetchers = 1
        offset.metadata.max.bytes = 4096
        offsets.commit.required.acks = -1
        offsets.commit.timeout.ms = 5000
        offsets.load.buffer.size = 5242880
        offsets.retention.check.interval.ms = 600000
        offsets.retention.minutes = 1440
        offsets.topic.compression.codec = 0
        offsets.topic.num.partitions = 50
        offsets.topic.replication.factor = 3
        offsets.topic.segment.bytes = 104857600
        port = 9092
        principal.builder.class = class org.apache.kafka.common.security.auth.DefaultPrincipalBuilder
        producer.purgatory.purge.interval.requests = 1000
        queued.max.requests = 500
        quota.consumer.default = 9223372036854775807
        quota.producer.default = 9223372036854775807
        quota.window.num = 11
        quota.window.size.seconds = 1
```

```
        replica.fetch.backoff.ms = 1000
        replica.fetch.max.bytes = 1048576
        replica.fetch.min.bytes = 1
        replica.fetch.response.max.bytes = 10485760
        replica.fetch.wait.max.ms = 500
        replica.high.watermark.checkpoint.interval.ms = 5000
        replica.lag.time.max.ms = 10000
        replica.socket.receive.buffer.bytes = 65536
        replica.socket.timeout.ms = 30000
        replication.quota.window.num = 11
        replication.quota.window.size.seconds = 1
        request.timeout.ms = 30000
        reserved.broker.max.id = 1000
        sasl.enabled.mechanisms = [GSSAPI]
        sasl.kerberos.kinit.cmd = /usr/bin/kinit
        sasl.kerberos.min.time.before.relogin = 60000
        sasl.kerberos.principal.to.local.rules = [DEFAULT]
        sasl.kerberos.service.name = null
        sasl.kerberos.ticket.renew.jitter = 0.05
        sasl.kerberos.ticket.renew.window.factor = 0.8
        sasl.mechanism.inter.broker.protocol = GSSAPI
        security.inter.broker.protocol = PLAINTEXT
        socket.receive.buffer.bytes = 102400
        socket.request.max.bytes = 104857600
        socket.send.buffer.bytes = 102400
        ssl.cipher.suites = null
        ssl.client.auth = none
        ssl.enabled.protocols = [TLSv1.2, TLSv1.1, TLSv1]
        ssl.endpoint.identification.algorithm = null
        ssl.key.password = null
        ssl.keymanager.algorithm = SunX509
        ssl.keystore.location = null
        ssl.keystore.password = null
        ssl.keystore.type = JKS
        ssl.protocol = TLS
        ssl.provider = null
        ssl.secure.random.implementation = null
        ssl.trustmanager.algorithm = PKIX
        ssl.truststore.location = null
        ssl.truststore.password = null
        ssl.truststore.type = JKS
        unclean.leader.election.enable = true
        zookeeper.connect = localhost:2181
        zookeeper.connection.timeout.ms = 6000
        zookeeper.session.timeout.ms = 6000
        zookeeper.set.acl = false
        zookeeper.sync.time.ms = 2000
(kafka.server.KafkaConfig)
```