TP3_apartado_1_adrian_zapater

January 11, 2020

1 TP3 - Adrián José Zapater Reig

Apartado 1 - Kafka

1.0.1 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?

[1]:
```
# Vamos a usar el zookeeper que incorpora por defecto kafka
!cat /usr/local/kafka/config/zookeeper.properties
```

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# the directory where the snapshot is stored.
dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181
# disable the per-ip limit on the number of connections since this is a non-
production config
maxClientCnxns=0
# Disable the adminserver by default to avoid port conflicts.
# Set the port to something non-conflicting if choosing to enable this
admin.enableServer=false
# admin.serverPort=8080
```

```
# Enable snapshot.trust.empty config if the ZK upgrade from 3.4.X to 3.5.6 is
failing
# with "java.io.IOException: No snapshot found, but there are log entries"
error.
# Check upgrade docs for more details.
# snapshot.trust.empty=true
```

El puerto por defecto es 2181. Lo podemos localizar en la propiedad clientPort.

[ ]:
```
# Levantamos zookeeper:
! /usr/local/kafka/bin/zookeeper-server-start.sh /usr/local/kafka/config/
 ↪zookeeper.properties
```

1.1 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?

[8]:
```
! cat /usr/local/kafka/config/server.properties
```

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# see kafka.server.KafkaConfig for additional details and defaults

############################# Server Basics #############################

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=0

############################# Socket Server Settings
#############################

# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
```

```
#     listeners = PLAINTEXT://your.host.name:9092
#listeners=PLAINTEXT://:9092

# Hostname and port the broker will advertise to producers and consumers. If not
set,
# it uses the value for "listeners" if configured.  Otherwise, it will use the
value
# returned from java.net.InetAddress.getCanonicalHostName().
#advertised.listeners=PLAINTEXT://your.host.name:9092

# Maps listener names to security protocols, the default is for them to be the
same. See the config documentation for more details
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_
PLAINTEXT,SASL_SSL:SASL_SSL

# The number of threads that the server uses for receiving requests from the
network and sending responses to the network
num.network.threads=3

# The number of threads that the server uses for processing requests, which may
include disk I/O
num.io.threads=8

# The send buffer (SO_SNDBUF) used by the socket server
socket.send.buffer.bytes=102400

# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will accept (protection
against OOM)
socket.request.max.bytes=104857600


############################# Log Basics #############################

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

# The default number of log partitions per topic. More partitions allow greater
# parallelism for consumption, but this will also result in more files across
# the brokers.
num.partitions=1

# The number of threads per data directory to be used for log recovery at
startup and flushing at shutdown.
# This value is recommended to be increased for installations with data dirs
located in RAID array.
```

```
num.recovery.threads.per.data.dir=1

############################# Internal Topic Settings
#############################
# The replication factor for the group metadata internal topics
"__consumer_offsets" and "__transaction_state"
# For anything other than development testing, a value greater than 1 is
recommended to ensure availability such as 3.
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1

############################# Log Flush Policy #############################

# Messages are immediately written to the filesystem but by default we only
fsync() to sync
# the OS cache lazily. The following configurations control the flush of data to
disk.
# There are a few important trade-offs here:
#    1. Durability: Unflushed data may be lost if you are not using replication.
#    2. Latency: Very large flush intervals may lead to latency spikes when the
flush does occur as there will be a lot of data to flush.
#    3. Throughput: The flush is generally the most expensive operation, and a
small flush interval may lead to excessive seeks.
# The settings below allow one to configure the flush policy to flush data after
a period of time or
# every N messages (or both). This can be done globally and overridden on a per-
topic basis.

# The number of messages to accept before forcing a flush of data to disk
#log.flush.interval.messages=10000

# The maximum amount of time a message can sit in a log before we force a flush
#log.flush.interval.ms=1000

############################# Log Retention Policy #############################

# The following configurations control the disposal of log segments. The policy
can
# be set to delete segments after a period of time, or after a given size has
accumulated.
# A segment will be deleted whenever *either* of these criteria are met.
Deletion always happens
# from the end of the log.

# The minimum age of a log file to be eligible for deletion due to age
log.retention.hours=168
```

```
# A size-based retention policy for logs. Segments are pruned from the log
unless the remaining
# segments drop below log.retention.bytes. Functions independently of
log.retention.hours.
#log.retention.bytes=1073741824

# The maximum size of a log segment file. When this size is reached a new log
segment will be created.
log.segment.bytes=1073741824

# The interval at which log segments are checked to see if they can be deleted
according
# to the retention policies
log.retention.check.interval.ms=300000

############################# Zookeeper #############################

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=localhost:2181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=6000


############################# Group Coordinator Settings
#############################

# The following configuration specifies the time, in milliseconds, that the
GroupCoordinator will delay the initial consumer rebalance.
# The rebalance will be further delayed by the value of
group.initial.rebalance.delay.ms as new members join the group, up to a maximum
of max.poll.interval.ms.
# The default value for this is 3 seconds.
# We override this to 0 here as it makes for a better out-of-the-box experience
for development and testing.
# However, in production environments the default value of 3 seconds is more
suitable as this will help to avoid unnecessary, and potentially expensive,
rebalances during application startup.
group.initial.rebalance.delay.ms=0
```

[ ]:
```
! /usr/local/kafka/bin/kafka-server-start.sh /usr/local/kafka/config/server.
↪properties
```

Puerto por defecto: 9092. Lo podemos obtener de la traza de la shell al levantar el broker.

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

[22]:
```
! /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:
↪9092 --replication-factor 1 --partitions 1 --topic initial
```

[11]:
```
# Comprobamos que se ha creado:
! /usr/local/kafka/bin/kafka-topics.sh --list --bootstrap-server localhost:9092
```

```
initial
```

1.3 4. Describir el topic "initial" y comentar los resultados que se obtienen.

Tenemos un topic con una única partición, por lo que dicha partición tendrá todos los mensajes. Tenemos un único factor de replicación por lo que los mensajes solo se almacenarán en el master. No hay tolerancia a fallos.

1.4 5. Arrancar un productor sobre el topic "initial".

[ ]:
```
! /usr/local/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092␣
↪--topic initial
```

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

[ ]:
```
! /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:
↪9092 --topic initial --from-beginning
```

1.6 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.

Los mensajes que escribimos en el productor aparecen en el mismo orden en el consumidor con un pequeño delay. los mensajes que se han escrito en el productor antes de crear el consumidor también han sido recibido en el mismo orden en el consumidor.

1.7 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?

[ ]:
```
! /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:
↪9092 --replication-factor 2 --partitions 2 --topic testRep
```

RESULTADO: [2020-01-10 19:31:28,198] ERROR java.util.concurrent.ExecutionException: org.apache.kafka.common.errors.InvalidReplicationFactorException: Replication factor: 2 larger than available brokers: 1.

Nos devuelve un error. Esto se debe a que solo tenemos un broker levantado y no puede replicar 2 veces los mensajes si solo tenemos un broker.

1.8 9. Detener el broker levantado en el ejercicio 2.

1.9 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.

[17]:
```
#Fichero de configuración de broker 1 - id 0:
! cat /usr/local/kafka/config/broker_3.properties
```

```
broker.id=0
listeners=PLAINTEXT://:9092
log.dirs=/tmp/kafka-logs-0


num.network.threads=3
num.io.threads=8

socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600


num.partitions=1
num.recovery.threads.per.data.dir=1

offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1

log.retention.hours=168
log.segment.bytes=1073741824
log.retention.check.interval.ms=300000

zookeeper.connect=localhost:2181
zookeeper.connection.timeout.ms=6000

group.initial.rebalance.delay.ms=0
```

[18]:
```
#Fichero de configuración de broker 2 - id 1:
! cat /usr/local/kafka/config/broker_1.properties
```

```
broker.id=1
listeners=PLAINTEXT://:9093
log.dirs=/tmp/kafka-logs-1
```

```
num.network.threads=3
num.io.threads=8

socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600


num.partitions=1
num.recovery.threads.per.data.dir=1

offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1

log.retention.hours=168
log.segment.bytes=1073741824
log.retention.check.interval.ms=300000

zookeeper.connect=localhost:2181
zookeeper.connection.timeout.ms=6000

group.initial.rebalance.delay.ms=0
```

[19]:
```
#Fichero de configuración de broker 3 - id 2:
! cat /usr/local/kafka/config/broker_2.properties
```

```
broker.id=2
listeners=PLAINTEXT://:9094
log.dirs=/tmp/kafka-logs-2


num.network.threads=3
num.io.threads=8

socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600


num.partitions=1
num.recovery.threads.per.data.dir=1

offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1
```

```
log.retention.hours=168
log.segment.bytes=1073741824
log.retention.check.interval.ms=300000

zookeeper.connect=localhost:2181
zookeeper.connection.timeout.ms=6000

group.initial.rebalance.delay.ms=0
```

[ ]:
```
! /usr/local/kafka/bin/kafka-server-start.sh /usr/local/kafka/config/broker_1.
 ↪properties
! /usr/local/kafka/bin/kafka-server-start.sh /usr/local/kafka/config/broker_2.
 ↪properties
! /usr/local/kafka/bin/kafka-server-start.sh /usr/local/kafka/config/broker_3.
 ↪properties
```

Para poder levantar 3 brokers necesitamos declarar cada broker con un id único, un puerto libre y un fichero de log separado.

se han modificado las siguientes propiedades:

- broker.id=<$id>
- listeners=<$puerto>
- log.dirs=<$ruta_de_logs>

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

[23]:
```
! /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:
 ↪9092 --replication-factor 3 --partitions 1 --topic testOrder1
! /usr/local/kafka/bin/kafka-topics.sh --list --bootstrap-server localhost:9092
```

```
__consumer_offsets
initial
testOrder1
```

NOTA: Apuntamos al broker 0 como bootstrap-server para almacenar los metadatos ahí.

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

[24]:
```
! /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:
 ↪9092 --replication-factor 3 --partitions 3 --topic testOrder2
! /usr/local/kafka/bin/kafka-topics.sh --list --bootstrap-server localhost:9092
```

```
__consumer_offsets
initial
```

```
testOrder1
testOrder2
```

1.12 14. Describir los topics "testOrder1" y "testOrder2" y comentar los resultados que se obtienen.

Se han creado 2 topics, ambos con 3 factor de replicación 3 por lo que cada mensaje se almacenará 3 veces. Un topic tiene 1 partición a la que irán todos los mensajes y el otro tiene 3 particiones. Los mensajes del segundo se distribuirán por id de partición, si tiene, entre las particiones. En caso de no tener id de partición se usará el módulo de la key del mensaje y si tampoco tiene eso, se utilizará el algoritmo round robin para repartir los mensajes.

1.13 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.

[ ]:
```
! /usr/local/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092␣
↪--topic testOrder1
```

adzarei@adzarei-mac:kafka $/usr/local/kafka/bin/kafka - console - producer.sh - -broker - listlocalhost : 9092 - -topictestOrder1 > hola > quetal > blablabla > adios$

[ ]:
```
! /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:
↪9092 --topic testOrder1 --from-beginning
```

adzarei@adzarei-mac:kafka $/usr/local/kafka/bin/kafka - console - consumer.sh - -bootstrap - serverlocalhost : 9092 - -topictestOrder1 - -from - beginningholaquetalblablablaadios$

1.14 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.

[ ]:
```
! /usr/local/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092␣
↪--topic testOrder2
```

adzarei@adzarei-mac:kafka $/usr/local/kafka/bin/kafka - console - producer.sh - -broker - listlocalhost : 9092 - -topictestOrder2 > hola > quetal > blBLABLA > adios >$

adzarei@adzarei-mac:kafka $/usr/local/kafka/bin/kafka - console - consumer.sh - -bootstrap - serverlocalhost : 9092 - -topictestOrder2 - -from - beginningholaquetalblBLABLAadios$

1.15 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.

[ ]:
```
! /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:
↪9092 --topic testOrder1
```

[ ]:
```
! seq 1 30 | /usr/local/kafka/bin/kafka-console-producer.sh --broker-list␣
↪localhost:9092 --topic testOrder1
```

adzarei@adzarei-mac:kafka */usr/local/kafka/bin/kafka — console — consumer.sh — —bootstrap — serverlocalhost* : 9092 — —*topictestOrder*1123456789101112131415161718192021222324252627282930

1.16 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.

[ ]:
```
! /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:
↪9092 --topic testOrder2
```

[ ]:
```
! seq 1 30 | /usr/local/kafka/bin/kafka-console-producer.sh --broker-list␣
↪localhost:9092 --topic testOrder2
```

adzarei@adzarei-mac:kafka */usr/local/kafka* — 2.3.0/*bin/kafka* — *console* — *consumer.sh* — —*bootstrap* — *serverlocalhost* : 9092 — —*topictestOrder*2147101316192225282581114172023262936912151821242730

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

Como se puede apreciar kafka sólo respeta el orden de los mensajes en una partición, pero no entre particiones.

Se puede apreciar que, como no se ha asignado una kay para los mensajes ha usado el algoritmo round roubin para enviar un mensaje (número) a cada partición:

1 -> partición 1,

2 -> partición 2,

3 -> partición 3,

4 -> partición 1,

....

1.18 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).

[ ]:
```
! /usr/local/kafka/bin/kafka-run-class.sh kafka.tools.GetOffsetShell␣
↪--broker-list localhost:9092 --topic testOrder1
```

testOrder1:0:30 -

[ ]:
```
! /usr/local/kafka/bin/kafka-run-class.sh kafka.tools.GetOffsetShell␣
↪--broker-list localhost:9092 --topic testOrder2
```

testOrder2:0:10 testOrder2:1:10 testOrder2:2:10 testOrder1 solo tiene 1 partición y por eso una única línea con 30 mensajes.

testOrder2 tiene 3 particiones y por eso tiene una línea con 10 mensajes por partición.

1.19 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?

[ ]:
```
! /usr/local/kafka/bin/kafka-topics.sh --describe --bootstrap-server localhost:
↪9092 --topic testOrder2
```

/usr/local/kafka-2.3.0/bin/kafka-topics.sh –describe –bootstrap-server localhost:9092 –topic testOrder2 Topic:testOrder2 PartitionCount:3 ReplicationFactor:3 Configs:segment.bytes=1073741824 Topic: testOrder2 Partition: 0 Leader: 1 Replicas: 1,0,2 Isr: 1,0 Topic: testOrder2 Partition: 1 Leader: 0 Replicas: 0,2,1 Isr: 0,1 Topic: testOrder2 Partition: 2 Leader: 1 Replicas: 2,1,0 Isr: 1,0 Como se puede apreciar, la partición 2 ha cambiado de líder, en este caso es el broker 1. En la columna ISR vemos que ahora sólo hay 2 replicas actualizadas, en los broker 0 y 1 (no aparece el broker 2)