Kafka

刘磊

2020 年 11 月

安装

下载地址：http://kafka.apache.org/downloads

DOWNLOAD

2.6.0 is the latest release. The current stable version is 2.6.0.

You can verify your download by following these procedures and using these KEYS.

2.6.0

- Released Aug 3, 2020
- Release Notes
- Source download: kafka-2.6.0-src.tgz (asc, sha512)
- Binary downloads:
  - Scala 2.12 - kafka_2.12-2.6.0.tgz (asc, sha512)
  - Scala 2.13 - kafka_2.13-2.6.0.tgz (asc, sha512)

下载 kafka_2.12-2.6.0.tgz 和 kafka_2.12-2.6.0.tgz.sha512 到~/big_data_tools 并验证压缩包的完成性。

复制 kafka_2.12-2.6.0.tgz 到/apps 下，并进行解压

```
cp ~/big_data_tools/kafka_2.12-2.6.0.tgz /apps/
cd /apps
tar zxvf kafka_2.12-2.6.0.tgz
```

重命名 kafka_2.12-2.6.0 为 kafka，删除压缩包 `kafka_2.12-2.6.0.tgz`

```
mv kafka_2.12-2.6.0 kafka
rm kafka_2.12-2.6.0.tgz
```

使用 Kafka

启动 ZooKeeper 服务

Kafka 自带了 ZooKeeper，直接使用脚本启动单节点的 ZooKeeper 即可。

```
cd /apps/kafka
bin/zookeeper-server-start.sh -daemon config/zookeeper.properties
```

加-daemon 参数，可以在后台启动 Zookeeper，输出的信息保存在执行目录的 logs/zookeeper.out 文件中。

启动 Kafka 服务

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

创建一个主题

另外打开一个终端，创建一个名为“first_topic”的主题，只包含一个分区，只有一个副本，命令如下：

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

```
lei@ubuntu:/apps/kafka$ bin/kafka-topics.sh --create --bootstrap-server localhos
t:9092 --replication-factor 1 --partitions 1 --topic first_topic
WARNING: Due to limitations in metric names, topics with a period ('.') or under
score ('_') could collide. To avoid issues it is best to use either, but not bot
h.
Created topic first_topic.
```

查看创建的主题

查看 Kafka 中有哪些已创建的主题，可以用以下命令

```
bin/kafka-topics.sh --list \
--bootstrap-server localhost:9092
```

```
lei@ubuntu:/apps/kafka$ bin/kafka-topics.sh --list --bootstrap-server localhost:
9092
__consumer_offsets
first_topic
```

删除主题（跳过这一步，需要时再删除）

```
bin/kafka-topics.sh --delete \
--bootstrap-server localhost:9092 \
--topic first_topic
```

发送消息到服务中

运行生产者，然后输入一些消息发送到服务器。默认情况下，每行将作为单独的消息发送。

```
bin/kafka-console-producer.sh \
--broker-list localhost:9092 \
--topic first_topic
```

```
lei@ubuntu:/apps/kafka$ bin/kafka-console-producer.sh --broker-list localhost:90
92 --topic first_topic
>Hello kafka!
>This is a message!
```

从服务中获取消息

再打开一个终端运行消费者，从服务中获取已有的消息。

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

```
lei@ubuntu:/apps/kafka$ bin/kafka-console-consumer.sh --bootstrap-server localho
st:9092 --topic first_topic --from-beginning
Hello kafka!
This is a message!
```

查看运行信息

另外再打开一个终端，运行以下命令查看运行信息

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

```
lei@ubuntu:/apps/kafka$ bin/kafka-topics.sh --describe --bootstrap-server localh
ost:9092 --topic first_topic
Topic: first_topic      PartitionCount: 1       ReplicationFactor: 1    Configs:
 segment.bytes=1073741824
        Topic: first_topic      Partition: 0    Leader: 0       Replicas: 0    I
sr: 0
```

第一个行显示所有 Partition 的一个总结，以下每一行给出一个 Partition 中的信息，我们只有一个 Partition，所以只显示一行。

Leader：是在给出的所有 Partitons 中负责读写的节点，每个节点都有可能成为 Leader

Replicas：显示给定 Partiton 所有副本所存储节点的节点列表，不管该节点是否是 Leader 或者是否存活。

Isr：副本都已同步的的节点集合，这个集合中的所有节点都是存活状态，并且跟 Leader 同步

下面是一个具有三个分区的例子

```
Topic:test    PartitionCount:3    ReplicationFactor:3    Configs:
    Topic: test    Partition: 0    Leader: 0    Replicas: 0,1,2    Isr: 0,2,1
    Topic: test    Partition: 1    Leader: 1    Replicas: 1,2,0    Isr: 1,2,0
    Topic: test    Partition: 2    Leader: 2    Replicas: 2,0,1    Isr: 2,0,1
```

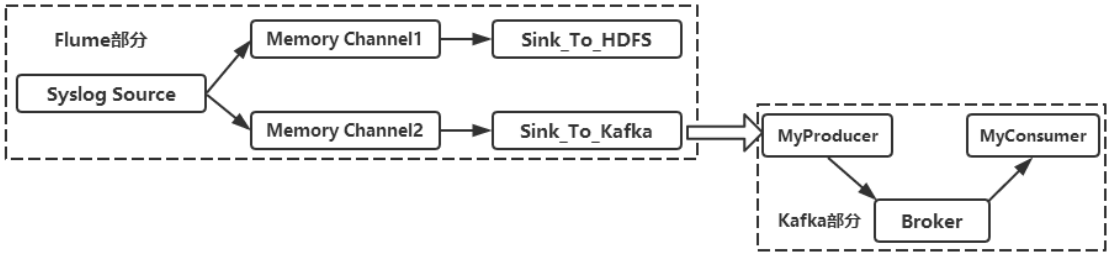
关闭 Zookeeper

需要时再关闭，这里可以先不关闭。

```
bin/zookeeper-server-stop.sh -daemon config/zookeeper.properties
```

Flume 传输数据给 Kafka

新建 Flume 的配置文件/apps/flume/conf/syslog_mem_hdfskafka.conf，使用 Flume 抓取 syslog 端口的日志数据，使用 mem 作为 Channel，一个输出是将数据存储到 HDFS 中的/myflume/目录下，作为持久存储；另一个输出是将数据传递给 Kafka 进行使用，Kafka 端启用 console-consumer 来消费数据，并输出到屏幕上。



创建 Flume 的配置文件/apps/flume/conf/syslog_mem_hdfsandkafka.conf

```
#定义各个组件
agent1.sources = src
agent1.channels = ch_hdfs ch_kafka
agent1.sinks = des_hdfs des_kafka

#把组件关联起来
agent1.sources.src.channels = ch_hdfs ch_kafka
agent1.sinks.des_hdfs.channel = ch_hdfs
agent1.sinks.des_kafka.channel = ch_kafka

#配置 source
agent1.sources.src.type = syslogtcp
agent1.sources.src.bind = localhost
agent1.sources.src.port = 6666

#配置 channel
agent1.channels.ch_hdfs.type = memory
agent1.channels.ch_kafka.type = memory

#配置 hdfs sink
```

```
agent1.sinks.des_hdfs.type = hdfs
agent1.sinks.des_hdfs.hdfs.path = hdfs://localhost:9000/myflume/
agent1.sinks.des_hdfs.hdfs.useLocalTimeStamp = true
#设置 flume 临时文件的前缀为_
agent1.sinks.des_hdfs.hdfs.inUsePrefix=_
#设置 flume 写入文件的前缀
agent1.sinks.des_hdfs.hdfs.filePrefix = q7
agent1.sinks.des_hdfs.hdfs.fileType = DataStream
agent1.sinks.des_hdfs.hdfs.writeFormat = Text

#配置 kafka sink
agent1.sinks.des_kafka.type = org.apache.flume.sink.kafka.KafkaSink
agent1.sinks.des_kafka.brokerList = localhost:9092
agent1.sinks.des_kafka.topic = flumekafka
agent1.sinks.des_kafka.batchSize=100
agent1.sinks.des_kafka.requiredAcks=1
```

启动 Hadoop

```
/apps/hadoop/sbin/start-all.sh
```

启动 Zookeeper

```
cd /apps/kafka
/bin/zookeeper-server-start.sh \
-daemon config/zookeeper.properties
```

启动 kafka-server

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

创建 topic

再打开一个终端，在 Kafka 中创建名为 flumekafka 的 topic

```
cd /apps/kafka
bin/kafka-topics.sh --create \
--bootstrap-server localhost:9092 \
--replication-factor 1 \
--partitions 1 \
--topic flumekafka
```

```
lei@ubuntu:/apps/kafka$ bin/kafka-topics.sh --create --bootstrap-server localhos
t:9092 --replication-factor 1 --partitions 1 --topic flumekafka
Created topic flumekafka.
```

启动 Flume

```
cd /apps/flume
flume-ng agent --conf conf \
--conf-file /apps/flume/conf/syslog_mem_hdfsandkafka.conf \
--name agent1 \
-Dflume.root.logger=DEBUG,console
```

启动 consumer

另外打开一个终端，启动 Kafka 的 console consumer 来消费数据。

```
cd /apps/kafka
bin/kafka-console-consumer.sh \
--bootstrap-server localhost:9092 \
--topic flumekafka \
--from-beginning
```

发送数据

再打开一个终端使用 nc 命令向 6666 端口发送数据

```
echo "hello can you hear me?" | nc localhost 6666
```

查看 consumer 输出

查看 Kafka 的 console consumer 是否有内容输出

```
lei@ubuntu:/apps/kafka$ bin/kafka-console-consumer.sh \
> --bootstrap-server localhost:9092 \
> --topic flumekafka \
> --from-beginning
hello can you hear me?
```

查看 Flume-ng 输出

```
2020-11-20 21:51:54,333 INFO hdfs.BucketWriter: Creating hdfs://localhost:9000/m
yflume//_q7.1605880313955.tmp
2020-11-20 21:51:54,341 INFO clients.Metadata: Cluster ID: ecDPtJ5NR5y1isjQVMvBv
A
2020-11-20 21:52:26,996 INFO hdfs.HDFSEventSink: Writer callback called.
2020-11-20 21:52:26,996 INFO hdfs.BucketWriter: Closing hdfs://localhost:9000/my
flume//_q7.1605880313955.tmp
2020-11-20 21:52:27,065 INFO hdfs.BucketWriter: Renaming hdfs://localhost:9000/m
yflume/_q7.1605880313955.tmp to hdfs://localhost:9000/myflume/q7.1605880313955
```

查看 HDFS 结果

```
hadoop fs -cat /myflume/*
```

```
lei@ubuntu:/apps/kafka$ hadoop fs -cat /myflume/*
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/apps/hadoop/share/hadoop/common/lib/slf4j-log
4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/apps/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org
/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
hello can you hear me?
```