目　录

关于版本

关于**kafka**

Kafka是由Apache软件基金会开发的一个开源流处理平台，由Scala和Java编写。该项目的目标是为处理实时数据提供一个统一、高吞吐、低延迟的平台。其持久化层本质上是一个“按照分布式事务日志架构的大规模发布/订阅消息队列”，这使它作为企业级基础设施来处理流式数据非常有价值。此外，Kafka可以通过Kafka Connect连接到外部系统（用于数据输入/输出），并提供了Kafka Streams——一个Java流式处理库。

Kafka存储的消息来自任意多被称为“生产者”（Producer）的进程。数据从而可以被分配到不同的“分区”（Partition）、不同的“Topic”下。在一个分区内，这些消息被索引并连同时间戳存储在一起。其它被称为“消费者”（Consumer）的进程可以从分区查询消息。Kafka运行在一个由一台或多台服务器组成的集群上，并且分区可以跨集群结点分布。

Kafka高效地处理实时流式数据，可以实现与Storm、HBase和Spark的集成。作为聚类部署到多台服务器上，Kafka处理它所有的发布和订阅消息系统使用了四个API，即生产者API、消费者API、Stream API和Connector API。它能够传递大规模流式消息，自带容错功能，已经取代了一些传统消息系统，如JMS、AMQP等。

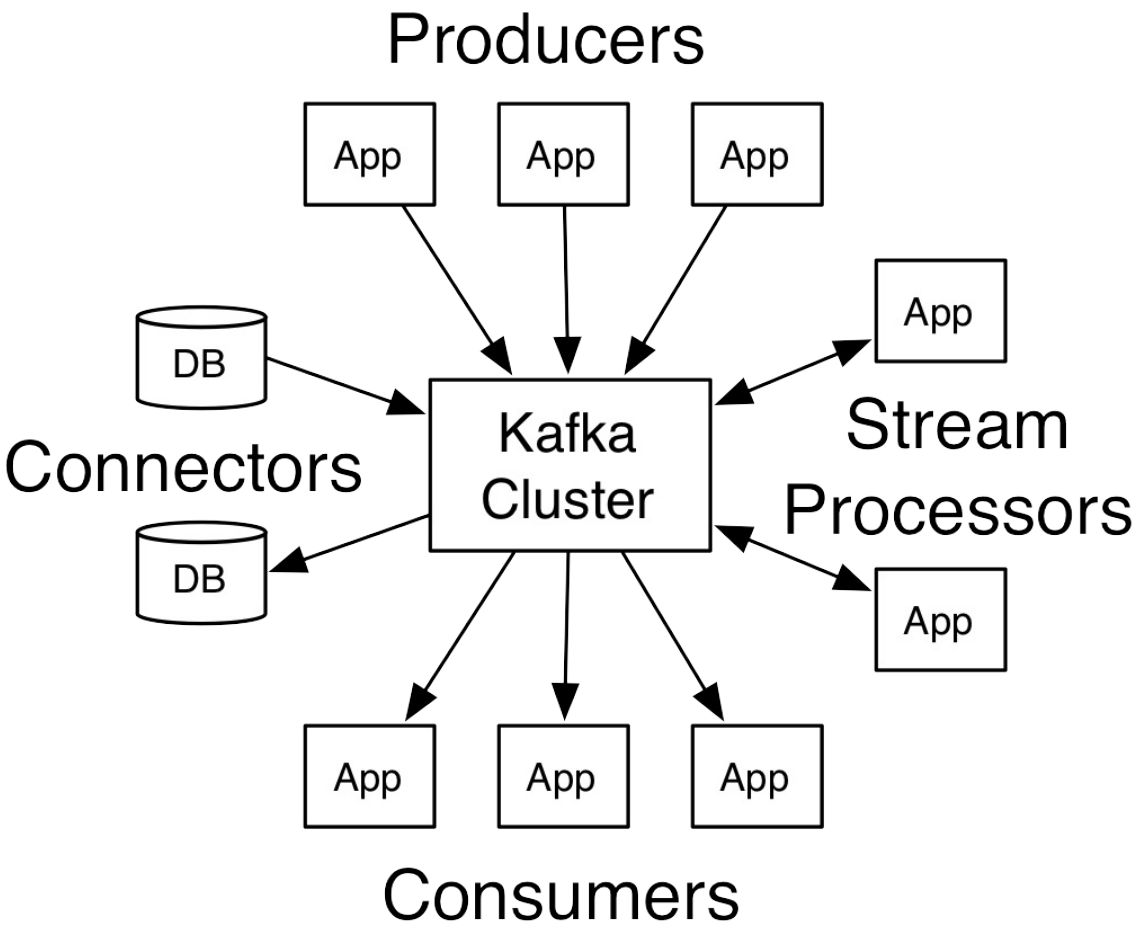
Kafka架构的主要术语包括Topic、Record和Broker。Topic由Record组成，Record持有不同的信息，而Broker则负责复制消息。Kafka有四个主要API：

生产者API：支持应用程序发布Record流。
消费者API：支持应用程序订阅Topic和处理Record流。
Stream API：将输入流转换为输出流，并产生结果。
Connector API：执行可重用的生产者和消费者API，可将Topic链接到现有应用程序。

Topic 用来对消息进行分类，每个进入到Kafka的信息都会被放到一个Topic下
Broker 用来实现数据存储的主机服务器
Partition 每个Topic中的消息会被分为若干个Partition，以提高消息的处理效率

作为参考，以下是LinkedIn最繁忙的一个集群（最高峰）的统计数据：
60 brokers
50k partitions (replication factor 2)
800k messages/sec in
300 MB/sec inbound, 1 GB/sec+ outbound

架构图如下：



更多请参考：kafka文档

本版块维护人员

版主：子木

QQ：1242119478

交流Q群：526749756

安装部署

基础环境

服务器：
test11 192.168.37.11
test12 192.168.37.12
test13 192.168.37.13
系统版本：
centos 7.6

安装

1、下载安装包：wget http://mirror-hk.koddos.net/apache/kafka/2.3.0/kafka_2.12-2.3.0.tgz
2、解压并进入目录：
tar zxf kafka_2.12-2.3.0.tgz -C /usr/local/
cd /usr/local/kafka_2.12-2.3.0/
3、只需要修改server.properties文件的部分配置，如下：

```
test11:
vim config/server.properties
broker.id=1
log.dirs=/tmp/kafka-logs-1
zookeeper.connect=192.168.37.11:2181,192.168.37.12:2181,192.168.37.13:21
81

test12:
vim config/server.properties
broker.id=2
log.dirs=/tmp/kafka-logs-2
zookeeper.connect=192.168.37.11:2181,192.168.37.12:2181,192.168.37.13:21
81

test13:
vim config/server.properties
broker.id=3
log.dirs=/tmp/kafka-logs-3
zookeeper.connect=192.168.37.11:2181,192.168.37.12:2181,192.168.37.13:21
81
```

4、修改三台服务器上的hosts文件使用ip与主机名对应，如下：

```
192.168.37.11 test11
192.168.37.12 test12
```

```
192.168.37.13 test13
```

4、分别启动test11,12,13服务器上的kafka，命令如下：

`nohup ./bin/kafka-server-start.sh ./config/server.properties >/dev/null 2>&1 &`

5、创建topic，例：

`./bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 1 --topic my-replicated-topic`

当然，你可以连接到指定服务器创建，例：

`./bin/kafka-topics.sh --create --bootstrap-server test12:9092 --replication-factor 3 --partitions 1 --topic test22`

6、查看创建主题的状态，如下：

```
[root@test11 kafka_2.12-2.3.0]# ./bin/kafka-topics.sh --describe --boots
trap-server localhost:9092 --topic test22
Topic:test22    PartitionCount:1        ReplicationFactor:3     Configs:
segment.bytes=1073741824
        Topic: test22   Partition: 0    Leader: 2       Replicas: 2,1,3
Isr: 2,1,3

#当然用下面这个命令也是可以的
[root@test11 kafka_2.12-2.3.0]# ./bin/kafka-topics.sh --describe --zooke
eper test11:2181 --topic test22
```

leader：是负责给定分区的所有读写的节点。每个节点将成为随机选择的分区部分的领导者。上面节点2是leader节点

replicas：是复制此分区日志的节点列表，无论它们是否为领导者，或者即使它们当前处于活动状态。

isr：是“同步”复制品的集合。这是副本列表的子集，该列表当前处于活跃状态并且已经被领导者捕获

7、下面我们验证一下，向我们的新主题发送消息

```
[root@test11 kafka_2.12-2.3.0]# ./bin/kafka-console-producer.sh --broker
-list localhost:9092 --topic test22
>I send info in node 1

#在test13上接收消息
[root@test13 kafka_2.12-2.3.0]# ./bin/kafka-console-consumer.sh --bootst
rap-server localhost:9092 --from-beginning --topic test22
I send info in node 1
```

8、因为上面我们清楚节点2是leader节点，我们这时把节点2的kafka kill掉，看一下集群有没有受影响

```
[root@test12 kafka_2.12-2.3.0]# ps aux|grep kafka
root      5724  1.4 16.6 3657172 313304 pts/0  Sl   7月04  15:41 java -X
```

```
mx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingH
eapOccupancyPercent=35 -XX:+ExplicitGCInvokesConcurrent -Djava.awt.headl
ess=true -Xloggc:/usr/local/kafka_2.12-2.3.0/bin/.
......
[root@test12 kafka_2.12-2.3.0]# kill -9 5724
[root@test12 kafka_2.12-2.3.0]# ps aux|grep kafka
root     15860  0.0  0.0 112724   984 pts/0    R+   14:37   0:00 grep --
color=auto kafka
[1]+  已杀死               nohup ./bin/kafka-server-start.sh ./config/ser
ver.properties > /dev/null 2>&1
[root@test12 kafka_2.12-2.3.0]# ./bin/kafka-topics.sh --describe --boots
trap-server test13:9092 --topic test22
Topic:test22    PartitionCount:1        ReplicationFactor:3     Configs:
segment.bytes=1073741824
        Topic: test22   Partition: 0    Leader: 1       Replicas: 2,1,3
Isr: 1,3
#这时因为原来的leader节点2已经被kill，kafka重新选举出leader节点1来，已经不同步到
节点2了。
#我们再看下在test22主题上发消息有没有受到影响
test11发消息:
>I send info after leader 2 is kill
>[2019-07-05 14:38:32,519] WARN [Producer clientId=console-producer] Con
nection to node 2 (test12/192.168.37.12:9092) could not be established.
Broker may not be available. (org.apache.kafka.clients.NetworkClient)

test13接收消息:
[2019-07-05 14:37:44,032] WARN [Consumer clientId=consumer-1, groupId=co
nsole-consumer-19650] Connection to node 2 (test12/192.168.37.12:9092) c
ould not be established. Broker may not be available. (org.apache.kafka.
clients.NetworkClient)
I send info after leader 2 is kill

#从上面看这些消息仍可供消费的,只是会有node 2不能用的警告
```

管理kafka

管理topic

在本章的【安装部署】上已经说明了如何去创建topic,下面我们看一下其它管理topic的命令：
查看：

```
[root@test12 kafka_2.12-2.3.0]# ./bin/kafka-topics.sh --list --zookeeper
 test11:2181
__consumer_offsets
my-replicated-topic
test1
test22
```

删除：

```
[root@test12 kafka_2.12-2.3.0]# ./bin/kafka-topics.sh --delete --zookeep
er test11:2181 --topic my-replicated-topic
Topic my-replicated-topic is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true
.
```

修改：

```
[root@test12 kafka_2.12-2.3.0]# ./bin/kafka-topics.sh --describe --zooke
eper test11:2181 --topic test22
Topic:test22    PartitionCount:1        ReplicationFactor:3     Configs:
        Topic: test22   Partition: 0    Leader: 2       Replicas: 2,1,3
Isr: 1,3,2
[root@test12 kafka_2.12-2.3.0]# ./bin/kafka-topics.sh --bootstrap-server
 localhost:9092 --alter --topic test22 --partitions 3           #把topi
c test22原来的一个分区修改为3个分区
[root@test12 kafka_2.12-2.3.0]# ./bin/kafka-topics.sh --describe --zooke
eper test11:2181 --topic test22
Topic:test22    PartitionCount:3        ReplicationFactor:3     Configs:
        Topic: test22   Partition: 0    Leader: 2       Replicas: 2,1,3
Isr: 1,3,2
        Topic: test22   Partition: 1    Leader: 3       Replicas: 3,2,1
Isr: 3,2,1
        Topic: test22   Partition: 2    Leader: 1       Replicas: 1,3,2
Isr: 1,3,2
```

kafka工具

kafka监控工具—Burrow

因为Burrow是用go写的，所以要先安装go环境，如下：
1、下载go安装包：`wget https://dl.google.com/go/go1.12.6.linux-amd64.tar.gz`
2、解压到指定目录：`tar -zxf go1.12.6.linux-amd64.tar.gz -C /usr/local/`
3、添加环境变量：

```
[root@test11 go]# vim /etc/profile.d/go.sh
export PATH=$PATH:/usr/local/go/bin
[root@test11 go]# source /etc/profile.d/go.sh
```

构建和安装
go get github.com/linkedin/Burrow
cd /root/go/src/github.com/linkedin/Burrow
dep ensure（如果没有安装的执行curl https://raw.githubusercontent.com/golang/dep/master/install.sh | sh，dep命令会生成在/root/go/bin下）
go install
4、编辑burrow.toml如下：

```
[root@test11 ~]# cat go/src/github.com/linkedin/Burrow/config/burrow.toml
[general]
pidfile="burrow.pid"
stdout-logfile="burrow.out"
access-control-allow-origin="mysite.example.com"

[logging]
filename="logs/burrow.log"
level="info"
maxsize=100
maxbackups=30
maxage=10
use-localtime=false
use-compression=true

[zookeeper]
servers=[ "test11:2181", "test12:2181", "test13:2181" ]
timeout=6
root-path="/burrow"

[client-profile.test]
```

```
client-id="burrow-test"
kafka-version="0.10.0"

[cluster.local]
class-name="kafka"
servers=[ "test11:9092", "test12:9092", "test13:9092" ]
client-profile="test"
topic-refresh=120
offset-refresh=30

[consumer.local]
class-name="kafka"
cluster="local"
servers=[ "test11:9092", "test12:9092", "test13:9092" ]
client-profile="test"
group-blacklist="^(console-consumer-|python-kafka-consumer-|quick-).*$"
group-whitelist=""

[consumer.local_zk]
class-name="kafka_zk"
cluster="local"
servers=[ "test11:2181", "test12:2181", "test13:2181" ]
#zookeeper-path="/kafka-cluster"
zookeeper-timeout=30
group-blacklist="^(console-consumer-|python-kafka-consumer-|quick-).*$"
group-whitelist=""

[httpserver.default]
address=":8000"

[storage.default]
class-name="inmemory"
workers=20
intervals=15
expire-group=604800
min-distance=1
```

启动：`/root/go/bin/Burrow --config-dir /root/go/src/github.com/linkedin/Burrow/config`

Kafka Manager

1、下载安装包：`wget https://github.com/yahoo/kafka-manager/archive/2.0.0.2.tar.gz`
2、解压安装包到指定目录：`tar zxf 2.0.0.2.tar.gz -C /usr/local/`
3、yum安装sbt(因为kafka-manager需要sbt编译)

```
[root@test11 Burrow]# curl https://bintray.com/sbt/rpm/rpm > bintray-sbt
-rpm.repo
[root@test11 Burrow]# mv bintray-sbt-rpm.repo /etc/yum.repos.d/
[root@test11 Burrow]# yum install sbt -y
```

4、编译kafka-manager

`[root@test11 kafka-manager-2.0.0.2]# ./sbt clean dist`

提示如下，证明编译完成：

```
[success] All package validations passed
[info] Your package is ready in /usr/local/kafka-manager-2.0.0.2/target/
universal/kafka-manager-2.0.0.2.zip
```

5、重新解压上面编译好的kafka-manager-2.0.0.2.zip

```
cd /usr/local/kafka-manager-2.0.0.2/target/universal/
unzip kafka-manager-2.0.0.2.zip
mv kafka-manager-2.0.0.2 /usr/local/kafka-manager-2
[root@test11 local]# cd /usr/local/kafka-manager-2
[root@test11 kafka-manager-2]# ls
bin  conf  lib  README.md  share
```

6、更改配件文件的kafka-manager.zkhosts列表为自己的zk节点

```
[root@test11 kafka-manager-2]# vim conf/application.conf
kafka-manager.zkhosts="test11:2181,test12:2181,test13:2181"
```

7、启动服务，注意请先启动zk,kafka。bin/kafka-manager 默认的端口是9000，可通过 -Dhttp.port，指定端口; -Dconfig.file=conf/application.conf指定配置文件:

`nohup bin/kafka-manager -Dconfig.file=conf/application.conf -Dhttp.port=8088 &`

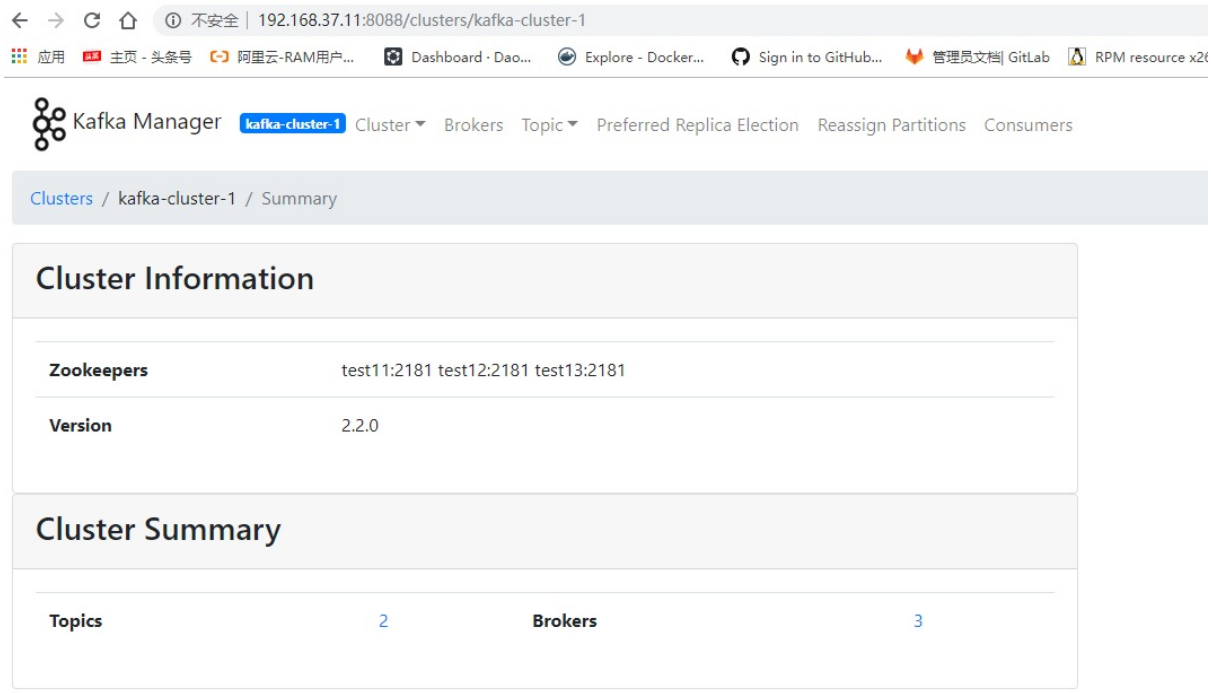
登录kafka-manager 的web界面，并且添加集群，如下：

网址：http://192.168.37.11:8088

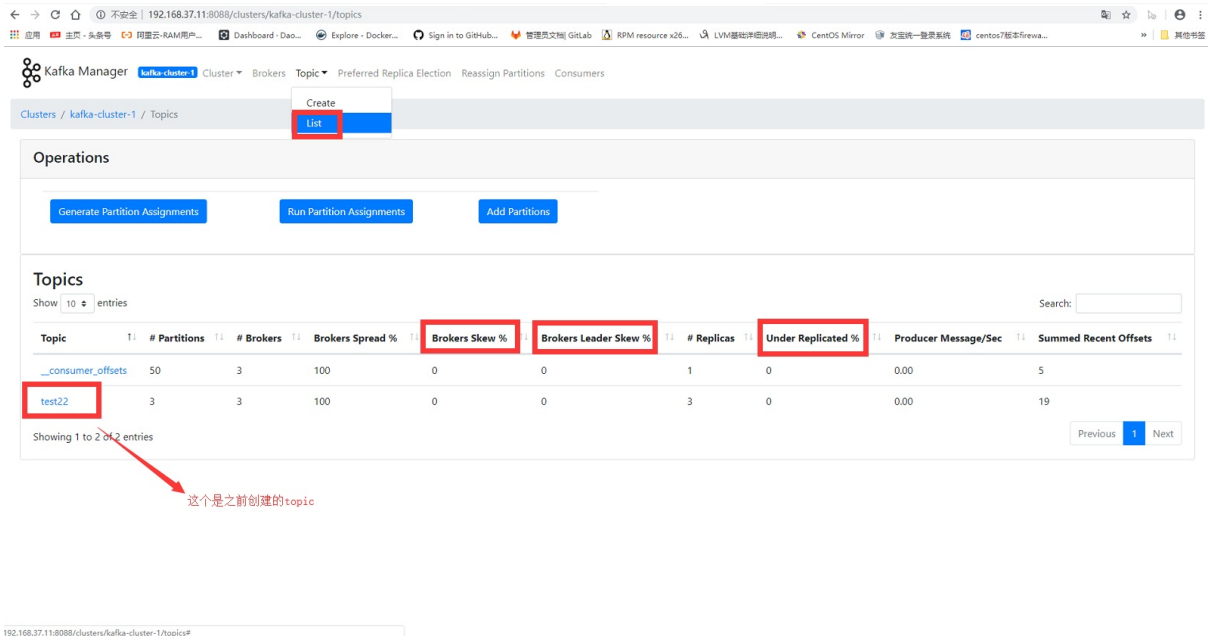
截图如下，


添加群集
群集名称
kafka-cluster-1
Cluster Zookeeper主机
test11:2181,test12:2181,test13:2181
卡夫卡版
2.2.0
启用JMX轮询（在启动kafka服务器之前设置JMX_PORT env变量）
JMX Auth用户名
JMX Auth密码
JMX与SSL
启用Logkafka
轮询消费者信息（如果ZK用于较旧的Kafka版本的偏移跟踪，则不推荐用于大量消费者）
过滤掉不活跃的消费者
启用Active OffsetCache（不建议大型消费者使用）
显示代理和主题大小（仅在应用此修补程序后有效）
brokerViewUpdatePeriodSeconds
30

创建完后截图如下：



这个kafka-manager界面很简单，基本自行熟悉一下就明白它的功能了，下面就截图主要讲一下其中比较值得注意的三个参数



注：

Broker Skew: 反映 broker 的 I/O 压力，broker 上有过多的副本时，相对于其他 broker ，该 broker 频繁的从 Leader 分区 fetch 抓取数据，磁盘操作相对于其他 broker 要多，如果该指标过高，说明 topic 的分区均不不好，topic 的稳定性弱；

Broker Leader Skew：数据的生产和消费进程都至于 Leader 分区打交道，如果 broker 的 Leader 分区过多，该 broker 的数据流入和流出相对于其他 broker 均要大，该指标过高，说明 topic 的分流做的不够好；

Under Replicated: 该指标过高时，表明 topic 的数据容易丢失，数据没有复制到足够的 broker 上。