链滴

手把手教你于 docker 搭建 redis-cluster 集群并验证

作者：yxw839841231

原文链接：https://ld246.com/article/1632640491583

来源网站：链滴

文本介绍基于 docker 搭建 redis-cluster 集群环境，准备前提是安装了 docker 的 Linux环境，这里centos为背景。

介绍正文内容之前，有必要先梳理下redis的集中集群方案，从早到晚说来，redis集群方案演进了三方案，最开始是大家熟知的主从复制的集群，到哨兵模式，到最新的redis-cluster。

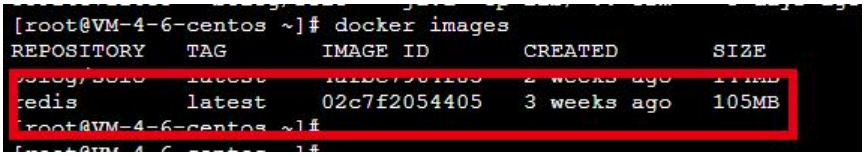
具体区别和优缺点，网上的资料很多，大家自行查询。需要说明的是，redis-cluster需要redis3.0以版本。

准备工作

1、为了能正常走完所有步骤，首先执行docker -v 命令，确认已经正常安装了docker。

2、使用 docker 是因为能快速部署redis实例，那么现在我们需要一个redis3.0以上的版本，本文使最新版本，那么你可以执行如下命令：docker pull redis。

正常获取完一个redis官方镜像后，执行docker images命令，你应该看到如下界面。



如果顺利，准备工作就告一段落。

创建redis实例

如果你对docker略知一二，那应该没什么问题。如果你不懂docker为何物也不碍事，因为我也不懂你要做的是按顺序执行好每一个命令。

完成准备操作后，在完成redis集群构建之前，我们需要3个redis实例，假设我们给取名为 r-node1、r node2和r-node3。同时，为了方便起见，我们将每个实例暴露的端口号定义为从6381开始。

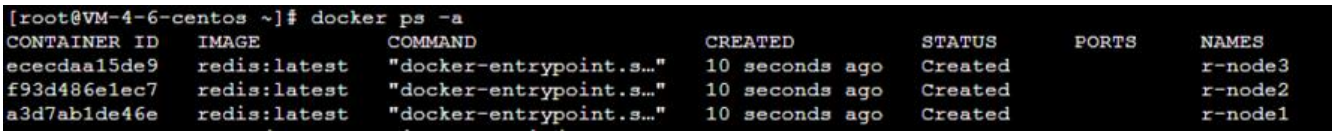
那么，你需要一次执行如下命令（不强制按顺序）：

```
docker create --name r-node1 --net host -v /data/redis-data/node1:/data redis:latest --cluste
-enabled yes --cluster-config-file node-1.conf --port 6379

docker create --name r-node2 --net host -v /data/redis-data/node2:/data redis:latest --cluste
-enabled yes --cluster-config-file node-2.conf --port 6380

docker create --name r-node3 --net host -v /data/redis-data/node3:/data redis:latest --cluste
-enabled yes --cluster-config-file node-3.conf --port 6381
```

每一行命令执行完毕，都能看到一串随机字符串，你可以忽略。然后通过docker ps -a命令，你能看如下三个容器。



当我们用命令docker start r-node1 r-node2 r-node3 启动容器后，就相当于启动了三个redis实例这个时候如果再次执行命令docker ps -a，会看到容器状态发生了变化。

```
[root@VM-4-6-centos ~]# docker start r-node1 r-node2 r-node3
r-node1
r-node2
r-node3
[root@VM-4-6-centos ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS     NAMES
ecedaa15de9    redis:latest   "docker-entrypoint.s…"   3 minutes ago    Up 10 seconds             r-node3
f93d486e1ec7   redis:latest   "docker-entrypoint.s…"   3 minutes ago    Up 10 seconds             r-node2
a3d7ab1de46e   redis:latest   "docker-entrypoint.s…"   3 minutes ago    Up 10 seconds             r-node1
```

构建集群

有了redis实例后，是时候搭建我们的集群了。

执行命令 docker exec -it r-node1 /bin/bash 这个时候你会进入到容器r-node1中。请注意，执行该命令，你的pwd已经发生了变化，如果你要退出请执行 exit ，执行前述命令可以再次进入。

接下来就是见证奇迹的时候，请执行命令：

redis-cli --cluster create 10.0.4.6:6379 10.0.4.6:6380 10.0.4.6:6381 --cluster-replicas 0

需要说明的是，10.0.4.6 应该是你的物理机IP，如果你是在widows系统下，可能会出现一些不可预的情况，如果是Linux系统，那么请放心根据ifconfig命令找到你的物理IP，替换即可。

如果不出意外，执行完上述命令，你需要输入yes，才能真正完成redis集群的搭建，然后你会看到如提示。

```
[root@VM-4-6-centos ~]# docker exec -it r-node1 /bin/bash
root@VM-4-6-centos:/data# redis-cli --cluster create 10.0.4.6:6379 10.0.4.6:6380 10.0.4.6:6381 --cluster-replicas 0
>>> Performing hash slots allocation on 3 nodes...
Master[0] -> Slots 0 - 5460
Master[1] -> Slots 5461 - 10922
Master[2] -> Slots 10923 - 16383
M: 04753ecf1b6bd08f72d86ae73b264ab0a9f6ee0b 10.0.4.6:6379
   slots:[0-5460] (5461 slots) master
M: 88e909b3b01ef691143616abe1519886f37ef8f1 10.0.4.6:6380
   slots:[5461-10922] (5462 slots) master
M: afc6142b124276b9539d6229a177cafc851853cb 10.0.4.6:6381
   slots:[10923-16383] (5461 slots) master
Can I set the above configuration? (type 'yes' to accept): yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join
.
>>> Performing Cluster Check (using node 10.0.4.6:6379)
M: 04753ecf1b6bd08f72d86ae73b264ab0a9f6ee0b 10.0.4.6:6379
   slots:[0-5460] (5461 slots) master
M: afc6142b124276b9539d6229a177cafc851853cb 10.0.4.6:6381
   slots:[10923-16383] (5461 slots) master
M: 88e909b3b01ef691143616abe1519886f37ef8f1 10.0.4.6:6380
   slots:[5461-10922] (5462 slots) master
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

All 16384 slots covered. 如果能看到这个提示，说明你的redis集群已经搭建完成，可以工作了。解redis的你，是不是对 16384 这个数字很敏感？你细品~

验证集群

我们使用 redis-cli 来执行键值对的操作，你可以执行命令 redis-cli -c 来连接集群。

接下来就可用过 set key value的方式来设置缓存，比如：从 k1:1 , k2:2 ..... 开始赋值，你会看到如情况：

```
root@VM-4-6-centos:/data# redis-cli -c
127.0.0.1:6379> set k1 1
-> Redirected to slot [12706] located at 10.0.4.6:6381
OK
10.0.4.6:6381> set k2 2
-> Redirected to slot [449] located at 10.0.4.6:6379
OK
10.0.4.6:6379> set k3 3
OK
10.0.4.6:6379> set k4 4
-> Redirected to slot [8455] located at 10.0.4.6:6380
OK
10.0.4.6:6380> set k5 5
-> Redirected to slot [12582] located at 10.0.4.6:6381
OK
10.0.4.6:6381> set k6 6
-> Redirected to slot [325] located at 10.0.4.6:6379
OK
10.0.4.6:6379> set k7 7
OK
10.0.4.6:6379> set k8 8
-> Redirected to slot [8331] located at 10.0.4.6:6380
OK
10.0.4.6:6380> set k9 9
-> Redirected to slot [12458] located at 10.0.4.6:6381
OK
10.0.4.6:6381> set k0 0
-> Redirected to slot [8579] located at 10.0.4.6:6380
OK
10.0.4.6:6380> keys *
1) "k0"
2) "k8"
3) "k4"
10.0.4.6:6380>
```

聪明的你肯定以及注意到了，在赋值的过程中，槽位发生了切换。命令提示符前端口号的不同也在告我们正在不同的实例下。

当我们在 6380 端口对应的实例下，尝试获取所有的key时，也只是拿到了该实例下的key，但如果我在 该实例下尝试查询k2会发生什么情况呢?

```
10.0.4.6:6380> get k2
-> Redirected to slot [449] located at 10.0.4.6:6379
"2"
10.0.4.6:6379>
```

我这边会自动切换到 6379 实例下，你的呢？请自行尝试。