

准备工作

安装Docker

下载redis源码包

mkdir /usr/local/redis\_docker

cd /usr/local/redis\_docker

wget http://download.redis.io/releases/redis-4.0.1.tar.gz

1

2

3

解压

tar -xzf redis-4.0.1.tar.gz

1

安装gcc

yum install -y gcc-c++

1

编译

make

#如果报错，可以尝试:make MALLOC=libc

1

2

配置redis.conf

vim /usr/local/redis\_docker/redis-4.0.1/redis.conf

以下是配置文件中相关的点，请一一对着修改即可

ip绑定

bind 0.0.0.0

0.0.0.0为不绑定ip

设置密码

requirepass 123456789

123456789为密码

主从复制的验证密码

masterauth 123456789

123456789为验证密码

日志文件

logfile “/usr/local/redis/logs/redis-server.log”

/usr/local/redis/logs/redis-server.log为日志文件的路径

开启集群配置

cluster-enabled yes

集群配置文件

cluster-config-file nodes-6379.conf

集群超时时间

cluster-node-timeout 15000

Redis基础镜像创建

编写镜像生成文件(Dockerfile)

cd /usr/local/redis\_docker

vim Dockerfile

1

2

在文件下拷贝一下内容

#指定基础镜像

FROM centos:7

#镜像作者

MAINTAINER lupf "397729842@qq.com"

#定义路径的环境变量

ENV REDIS\_HOME /usr/local

#将Dockerfile同级目录下的redis-4.0.1.tar.gz复制到镜像的根目录

ADD redis-4.0.1.tar.gz /

#创建安装目录，根据环境变量信息，实际的创建目录为:/usr/local/redis

RUN mkdir -p $REDIS\_HOME/redis

#将Dockerfile同级目录下redis-4.0.1中的redis配置文件拷贝到容器的/usr/local/redis目录中

ADD redis-4.0.1/redis.conf $REDIS\_HOME/redis/

#更新镜像的yum

RUN yum -y update

#安装gcc相关编译工具

RUN yum install -y gcc make

#指定工作目录

WORKDIR /redis-4.0.1

#执行编译

RUN make

#移动编译后的redis-server到容器相关的目录

RUN mv /redis-4.0.1/src/redis-server $REDIS\_HOME/redis/

#移动到上一级

WORKDIR /

#删除解压文件

RUN rm -rf /redis-4.0.1

#删除对于的工具

RUN yum remove -y gcc make

#添加数据卷

#/usr/local/redis/logs/redis-server.log

#此目录需要和redis.conf中logfile一致

VOLUME ["/usr/local/redis/logs"]

#暴露6379的端口

EXPOSE 6379

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

编译生成镜像

docker build -t lupf/redis-cluster .

1

注意，后面还有一个. 这个.表示当前目录下的Dockerfile

创建Redis节点镜像

创建目录

mkdir /usr/local/redis\_docker/redis-node

cd /usr/local/redis\_docker/redis-node

1

2

编写节点镜像文件

vim Dockerfile

FROM lupf/redis-cluster:latest

MAINTAINER lupf "397729842@qq.com"

ENTRYPOINT ["/usr/local/redis/redis-server", "/usr/local/redis/redis.conf"]

1

2

3

4

5

编译生成节点镜像

docker build -t lupf/redis-node .

1

查看镜像目录

docker images

配置集群

集群操作说明

CLUSTER INFO 打印集群的信息

CLUSTER NODES 列出集群当前已知的所有节点（node），以及这些节点的相关信息。

//节点(node)

CLUSTER MEET <ip> <port> 将 ip 和 port 所指定的节点添加到集群当中，让它成为集群的一份子。

CLUSTER FORGET <node\_id> 从集群中移除 node\_id 指定的节点。

CLUSTER REPLICATE <node\_id> 将当前节点设置为 node\_id 指定的节点的从节点。

CLUSTER SAVECONFIG 将节点的配置文件保存到硬盘里面。

//槽(slot)

CLUSTER ADDSLOTS <slot> [slot ...] 将一个或多个槽（slot）指派（assign）给当前节点。

CLUSTER DELSLOTS <slot> [slot ...] 移除一个或多个槽对当前节点的指派。

CLUSTER FLUSHSLOTS 移除指派给当前节点的所有槽，让当前节点变成一个没有指派任何槽的节点。

CLUSTER SETSLOT <slot> NODE <node\_id> 将槽 slot 指派给 node\_id 指定的节点，如果槽已经指派给另一个节点，那么先让另一个节点删除该槽>，然后再进行指派。

CLUSTER SETSLOT <slot> MIGRATING <node\_id> 将本节点的槽 slot 迁移到 node\_id 指定的节点中。

CLUSTER SETSLOT <slot> IMPORTING <node\_id> 从 node\_id 指定的节点中导入槽 slot 到本节点。

CLUSTER SETSLOT <slot> STABLE 取消对槽 slot 的导入（import）或者迁移（migrate）。

//键 (key)

CLUSTER KEYSLOT <key> 计算键 key 应该被放置在哪个槽上。

CLUSTER COUNTKEYSINSLOT <slot> 返回槽 slot 目前包含的键值对数量。

CLUSTER GETKEYSINSLOT <slot> <count> 返回 count 个 slot 槽中的键。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

启动镜像

启动各个节点容器

docker run -d --name redis-node-7000 -p 7000:6379 lupf/redis-node

docker run -d --name redis-node-7001 -p 7001:6379 lupf/redis-node

docker run -d --name redis-node-7002 -p 7002:6379 lupf/redis-node

docker run -d --name redis-node-7003 -p 7003:6379 lupf/redis-node

docker run -d --name redis-node-7004 -p 7004:6379 lupf/redis-node

docker run -d --name redis-node-7005 -p 7005:6379 lupf/redis-node

1

2

3

4

5

6

宿主机7000-7005 6个端口分别映射到对应的6个容器

查看docker下容器的状态

docker ps

查看容器内网下关联的IP

docker inspect 容器ID(根据自己的实际情况填) | grep IPAddress

docker inspect ff0dcb60a2f8 6cc62feebc45 a1dc7363e199 f4948f478ca8 cc27bf11c74f 251760674d99 | grep IPAddress

1

其中ff0dcb60a2f8 6cc62feebc45 a1dc7363e199 f4948f478ca8 cc27bf11c74f 251760674d99为容器的ID，请根据个人的实际情况填写

解析宿主机与容器的ip映射关系

宿主机(ip 端口) 容器(ip 端口)

192.168.1.208:7000 172.17.0.4:6379

192.168.1.208:7001 172.17.0.5:6379

192.168.1.208:7002 172.17.0.6:6379

192.168.1.208:7003 172.17.0.7:6379

192.168.1.208:7004 172.17.0.8:6379

192.168.1.208:7005 172.17.0.9:6379

节点配置

任意进入一个redis

redis-cli -p 7000

auth 123456789

1

2

添加节点

将以下所有的redis服务都添加到redis集群的节点

CLUSTER MEET 172.17.0.4 6379

CLUSTER MEET 172.17.0.5 6379

CLUSTER MEET 172.17.0.6 6379

CLUSTER MEET 172.17.0.7 6379

CLUSTER MEET 172.17.0.8 6379

CLUSTER MEET 172.17.0.9 6379

1

2

3

4

5

6

查看节点

CLUSTER NODES

由于当前只是添加了节点，没有分配槽点，也没有配置主从关联关系，所以集群并没有生效

配置槽点

配置添加的脚本

cd /usr/local/redis\_docker

vim addslots.sh

1

2

添加以下配置

#!/bin/bash

#将0-5461的槽点配置在172.17.0.4:6379（宿主机127.0.0.1:7000）的redis上

n=0

for ((i=n;i<=5461;i++))

do

/usr/local/bin/redis-cli -h 127.0.0.1 -p 7000 -a 123456789 CLUSTER ADDSLOTS $i

done

#将5462-10922的槽点配置在172.17.0.5:6379（宿主机127.0.0.1:7001）的redis上

n=5462

for ((i=n;i<=10922;i++))

do

/usr/local/bin/redis-cli -h 127.0.0.1 -p 7001 -a 123456789 CLUSTER ADDSLOTS $i

done

#将10923-16383的槽点配置在172.17.0.6:6379（宿主机127.0.0.1:7002）的redis上

n=10923

for ((i=n;i<=16383;i++))

do

/usr/local/bin/redis-cli -h 127.0.0.1 -p 7002 -a 123456789 CLUSTER ADDSLOTS $i

done

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

执行shell脚本并查看结果

sh vim addslots.sh

主从高可用

问题分析

根据上面的配置及下图分析；三主确实没有问题，16383个槽点也分别分散到各个redis服务中去了，集群的状态也没有问题，但是有没有发现，剩下的3个绿色的redis服务也都是主节点，切未分配槽点，也没有配置主从关联；因此现有的三个主redis服务只要有一个出现问题，整个集群奇偶失效了；下面来配置主从关联。

主从管理配置

主从关系关联表(左边为主，对应右边为从)

redis 主redis(宿主) 从redis 从redis(宿主)

172.17.0.4:6379 127.0.0.1:7000 172.17.0.7:6379 127.0.0.1:7003

172.17.0.5:6379 127.0.0.1:7001 172.17.0.8:6379 127.0.0.1:7004

172.17.0.6:6379 127.0.0.1:7002 172.17.0.9:6379 127.0.0.1:7005

编写配置关联脚本

映射关系请参考以上表格

vi addSlaveNodes.sh

#CLUSTER REPLICATE后面跟的是172.17.0.4:6379的集群ID

/usr/local/bin/redis-cli -h 127.0.0.1 -p 7003 -a 123456789 CLUSTER REPLICATE ae3f32ffe860ab89b7f7e8f3c33e3f83a9b49226

#CLUSTER REPLICATE后面跟的是172.17.0.5:6379的集群ID

/usr/local/bin/redis-cli -h 127.0.0.1 -p 7004 -a 123456789 CLUSTER REPLICATE b727870ce91ad7389ab62d9088d4146947002af9

#CLUSTER REPLICATE后面跟的是172.17.0.6:6379的集群ID

/usr/local/bin/redis-cli -h 127.0.0.1 -p 7005 -a 123456789 CLUSTER REPLICATE 6c19857b5bbf4cbd91f3b2b11e9d055e9e288f87

1

2

3

4

5

6

执行并查看接错

由此可见，主从关系已经创建成功

高可用测试

测试方案

测试结果

恢复刚刚测试的节点

读写测试

集群读写正常

测试连接不上的问题说明

按以上的创建方式，docker使用的是默认的bridge模式网络，即容器和宿主机有各自独立的网段，宿主机是192.168.0.1的网段，而虚拟机则是172.17.0.1的网段；由于所有的集群容器都是在一台宿主机上，使用的同一个网卡，因此，在宿主机下访问这个集群是没有任何问题的；但是(这里要翻车了)，如果我们使用其他局域网的ip（如192.168.1.100）访问，就会出现连接不上的情况；为什么会出现这样的情况呢，简单说明一下，上面我们添加集群节点的时候，是使用172.17.0.4 —172.17.0.9这个容器的ip加入的，虽然启动容器的时候，容器和宿主机是有做端口映射的，但实际在redis内部切换集群节点的时候是访问的172.17.0.1这个网段的ip，因此就会出现比如192.168.1.100这台访问集群的时候，其实连接的是172.17.0.4—172.17.0.9这个之间的ip，由于不是在一个局域网，因此就会出现连接不上的问题。

如何能解决这个问题呢，其实，我们根本不需要解决，这里只是学习，明白为什么出现这个情况就好了，实际的开发及使用中，如果我们按这样的方式搭建集群，其实一点作用都没有，既没有解决高可用，也没有解决负载的问题；肯定是使用不同的6台机器（如192.168.1.208—192.168.1.213）去做集群，然后我们添加集群节点的时候，就不用使用容器的ip端口去添加了，而是使用192.168.1.208—192.168.1.213这段ip去添加，这样就可以满足192.168.1.100这台机器的正常访问了。