《Docker环境下的前后端分离部署与运维》课程脚本

一、Docker虚拟机常用命令

1. 先更新软件包

```
yum -y update
```

2. 安装Docker虚拟机

```
yum install -y docker
```

3. 运行、重启、关闭Docker虚拟机

```
service docker start
service docker start
service docker stop
```

4. 搜索镜像

```
docker search 镜像名称
```

5. 下载镜像

```
docker pull 镜像名称
```

6. 查看镜像

```
docker images
```

7. 删除镜像

```
docker rmi 镜像名称
```

8. 运行容器

```
docker run 启动参数  镜像名称
```

9. 查看容器列表

```
docker ps -a
```

10. 停止、挂起、恢复容器

```
docker stop 容器ID
docker pause 容器ID
docker unpase 容器ID
```

11. 查看容器信息

```
docker inspect 容器ID
```

12. 删除容器

```
docker rm 容器ID
```

13. 数据卷管理

```
docker volume create 数据卷名称  #创建数据卷
docker volume rm 数据卷名称  #删除数据卷
docker volume inspect 数据卷名称  #查看数据卷
```

14. 网络管理

```
docker network ls 查看网络信息
docker network create --subnet=网段 网络名称
docker network rm 网络名称
```

15. 避免VM虚拟机挂起恢复之后，Docker虚拟机断网

```
vi /etc/sysctl.conf
```

文件中添加`net.ipv4.ip_forward=1`这个配置

```shell
#重启网络服务
systemctl  restart network
```

二、安装PXC集群，负载均衡，双机热备

1. 安装PXC镜像

```
docker pull percona/percona-xtradb-cluster:5.7.21
```

强烈推荐同学们安装5.7.21版本的PXC镜像，兼容性最好，在容器内可以执行apt-get安装各种程序包。最新版的PXC镜像内，无法执行apt-get，也就没法安装热备份工具了。

2. 为PXC镜像改名

```
docker tag percona/percona-xtradb-cluster pxc
```

3. 创建net1网段

```
docker network create --subnet=172.18.0.0/16 net1
```

4. 创建5个数据卷

```
docker volume create --name v1
docker volume create --name v2
docker volume create --name v3
docker volume create --name v4
docker volume create --name v5
```

5. 创建备份数据卷（用于热备份数据）

```
docker volume create --name backup
```

6. 创建5节点的PXC集群

   注意，每个MySQL容器创建之后，因为要执行PXC的初始化和加入集群等工作，耐心等待1分钟左右再用客户端连接MySQL。另外，必须第1个MySQL节点启动成功，用MySQL客户端能连接上之后，再去创建其他MySQL节点。

```
#创建第1个MySQL节点
docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=abc123456 -e CLUSTER_NAME=PXC -e
XTRABACKUP_PASSWORD=abc123456 -v v1:/var/lib/mysql -v backup:/data --privileged --
name=node1 --net=net1 --ip 172.18.0.2 pxc
#创建第2个MySQL节点
docker run -d -p 3307:3306 -e MYSQL_ROOT_PASSWORD=abc123456 -e CLUSTER_NAME=PXC -e
XTRABACKUP_PASSWORD=abc123456 -e CLUSTER_JOIN=node1 -v v2:/var/lib/mysql -v
backup:/data --privileged --name=node2 --net=net1 --ip 172.18.0.3 pxc
#创建第3个MySQL节点
docker run -d -p 3308:3306 -e MYSQL_ROOT_PASSWORD=abc123456 -e CLUSTER_NAME=PXC -e
XTRABACKUP_PASSWORD=abc123456 -e CLUSTER_JOIN=node1 -v v3:/var/lib/mysql --
privileged --name=node3 --net=net1 --ip 172.18.0.4 pxc
#创建第4个MySQL节点
docker run -d -p 3309:3306 -e MYSQL_ROOT_PASSWORD=abc123456 -e CLUSTER_NAME=PXC -e
XTRABACKUP_PASSWORD=abc123456 -e CLUSTER_JOIN=node1 -v v4:/var/lib/mysql --
privileged --name=node4 --net=net1 --ip 172.18.0.5 pxc
#创建第5个MySQL节点
docker run -d -p 3310:3306 -e MYSQL_ROOT_PASSWORD=abc123456 -e CLUSTER_NAME=PXC -e
XTRABACKUP_PASSWORD=abc123456 -e CLUSTER_JOIN=node1 -v v5:/var/lib/mysql -v
backup:/data --privileged --name=node5 --net=net1 --ip 172.18.0.6 pxc
```

7. 安装Haproxy镜像

```
docker pull haproxy
```

8. 宿主机上编写Haproxy配置文件

```
vi /home/soft/haproxy/haproxy.cfg
```

配置文件如下：

```
global
    #工作目录
    chroot /usr/local/etc/haproxy
    #日志文件，使用rsyslog服务中local5日志设备（/var/log/local5），等级info
    log 127.0.0.1 local5 info
    #守护进程运行
    daemon

defaults

    log global
```

```
    mode    http
    #日志格式
    option  httplog
    #日志中不记录负载均衡的心跳检测记录
    option  dontlognull
    #连接超时（毫秒）
    timeout connect 5000
    #客户端超时（毫秒）
    timeout client  50000
    #服务器超时（毫秒）
    timeout server  50000

#监控界面
listen  admin_stats
    #监控界面的访问的IP和端口
    bind  0.0.0.0:8888
    #访问协议
    mode        http
    #URI相对地址
    stats uri   /dbs
    #统计报告格式
    stats realm     Global\ statistics
    #登陆帐户信息
    stats auth  admin:abc123456
#数据库负载均衡
listen  proxy-mysql
    #访问的IP和端口
    bind  0.0.0.0:3306
    #网络协议
    mode  tcp
    #负载均衡算法（轮询算法）
    #轮询算法：roundrobin
    #权重算法：static-rr
    #最少连接算法：leastconn
    #请求源IP算法：source
    balance  roundrobin
    #日志格式
    option  tcplog
    #在MySQL中创建一个没有权限的haproxy用户，密码为空。Haproxy使用这个账户对MySQL数据库心跳检测
    option  mysql-check user haproxy
    server  MySQL_1 172.18.0.2:3306 check weight 1 maxconn 2000
```

```
    server  MySQL_2 172.18.0.3:3306 check weight 1 maxconn 2000
    server  MySQL_3 172.18.0.4:3306 check weight 1 maxconn 2000
    server  MySQL_4 172.18.0.5:3306 check weight 1 maxconn 2000
    server  MySQL_5 172.18.0.6:3306 check weight 1 maxconn 2000
    #使用keepalive检测死链
    option  tcpka
```

9. 创建两个Haproxy容器

```
#创建第1个Haproxy负载均衡服务器
docker run -it -d -p 4001:8888 -p 4002:3306 -v
/home/soft/haproxy:/usr/local/etc/haproxy --name h1 --privileged --net=net1 --ip
172.18.0.7 haproxy
#进入h1容器，启动Haproxy
docker exec -it h1 bash
haproxy -f /usr/local/etc/haproxy/haproxy.cfg
#创建第2个Haproxy负载均衡服务器
docker run -it -d -p 4003:8888 -p 4004:3306 -v
/home/soft/haproxy:/usr/local/etc/haproxy --name h2 --privileged --net=net1 --ip
172.18.0.8 haproxy
#进入h2容器，启动Haproxy
docker exec -it h2 bash
haproxy -f /usr/local/etc/haproxy/haproxy.cfg
```

10. Haproxy容器内安装Keepalived，设置虚拟IP

    注意事项：云主机不支持虚拟IP，另外很多公司的网络禁止创建虚拟IP（回家创建）,还有宿主机一定要关闭防火墙和SELINUX，很多同学都因为这个而失败的，切记切记

```
#进入h1容器
docker exec -it h1 bash
#更新软件包
apt-get update
#安装VIM
apt-get install vim
#安装Keepalived
apt-get install keepalived
#编辑Keepalived配置文件（参考下方配置文件）
vim /etc/keepalived/keepalived.conf
#启动Keepalived
service keepalived start

#宿主机执行ping命令
```

```
ping 172.18.0.201
```

配置文件内容如下：

```
vrrp_instance  VI_1 {
    state  MASTER
    interface  eth0
    virtual_router_id  51
    priority  100
    advert_int  1
    authentication {
        auth_type  PASS
        auth_pass  123456
    }
    virtual_ipaddress {
        172.18.0.201
    }
}
```

```
#进入h2容器
docker exec -it h2 bash
#更新软件包
apt-get update
#安装VIM
apt-get install vim
#安装Keepalived
apt-get install keepalived
#编辑Keepalived配置文件
vim /etc/keepalived/keepalived.conf
#启动Keepalived
service keepalived start
#宿主机执行ping命令
ping 172.18.0.201
```

配置文件内容如下：

```
vrrp_instance  VI_1 {
    state  MASTER
    interface  eth0
    virtual_router_id  51

    priority  100
```

```
    advert_int  1
    authentication {
        auth_type  PASS
        auth_pass  123456
    }
    virtual_ipaddress {
        172.18.0.201
    }
}
```

11. 宿主机安装Keepalived，实现双机热备

```
#宿主机执行安装Keepalived
yum -y install keepalived
#修改Keepalived配置文件
vi /etc/keepalived/keepalived.conf
#启动Keepalived
service keepalived start
```

Keepalived配置文件如下：

```
vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.99.150
    }
}

virtual_server 192.168.99.150 8888 {
    delay_loop 3
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
```

```
    real_server 172.18.0.201 8888 {
        weight 1
    }
}

virtual_server 192.168.99.150 3306 {
    delay_loop 3
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP

    real_server 172.18.0.201 3306 {
        weight 1
    }
}
```

12. 热备份数据

```
#进入node1容器
docker exec -it node1 bash
#更新软件包
apt-get update
#安装热备工具
apt-get install percona-xtrabackup-24
#全量热备
innobackupex --user=root --password=abc123456 /data/backup/full
```

13. 冷还原数据 停止其余4个节点，并删除节点

```
docker stop node2
docker stop node3
docker stop node4
docker stop node5
docker rm node2
docker rm node3
docker rm node4
docker rm node5
```

node1容器中删除MySQL的数据

```
#删除数据
rm -rf /var/lib/mysql/*
#清空事务
innobackupex --user=root --password=abc123456 --apply-back /data/backup/full/2018-04-15_05-09-07/
#还原数据
innobackupex --user=root --password=abc123456 --copy-back  /data/backup/full/2018-04-15_05-09-07/
```

重新创建其余4个节点，组件PXC集群

三、PXC 特别注意事项

PXC的主节点和从节点分别代表什么意义?

PXC中的主节点和从节点跟Replication主从节点是有巨大差别的。

首先Replication集群的数据同步只能是从主节点到从节点，而且节点的身份是固定的，主节点永远是Master，从节点永远是Slave，不能互换。

但是PXC上的主节点指的是第一个启动的节点，它不仅要启动MySQL服务，还要用Galera创建PXC集群。这些工作完成之后，主节点自动降级成普通节点。其他节点启动的时候只需要启动MySQL服务，然后再加入到PXC集群即可，所以这些节点从启动到关闭，身份一直都是普通节点。

为什么Node1能启动，而其他的PXC节点启动就闪退呢?

这是因为Node1启动的时候要做跟多工作，上面已经提及了。所以你没等node1把PXC集群创建出来，你就飞快的启动其他PXC节点，它们找不到Node1启动的PXC集群，所以就自动闪退了。

正确的办法是启动Node1之后，等待10秒钟，然后用Navicat访问一下，能访问了，再去启动其他PXC节点

如果PXC集群在运行的状态下，在宿主机上直接关机，或者停止Docker服务，为什么下次启动哪个PXC节点都会闪退?

这个要从PXC集群的节点管理说起，PXC节点的数据目录是/var/lib/mysql，好在这个目录被我们映射到数据卷上了。比如你访问v1数据卷就能看到node1的数据目录。这其中有个grastate.dat的文件，它里面有个safe_to_bootstrap参数被PXC用来记载谁是最后退出PXC集群的节点。比如node1是最后关闭的节点，那么PXC就会在把safe_to_bootstrap设置成1，代表node1节点最后退出，它的数据是最新的。下次启动必须先启动node1，然后其他节点与node1同步。

如果你在PXC节点都正常运行的状态下关闭宿主机Docker服务或者电源，那么PXC来不及判断谁是最后退出的节点，所有PXC节点一瞬间就都关上了，哪个节点的safe_to_boostrap参数就都是0。解决这个故障也很好办，那就是挑node1，把该参数改成1，然后正常启动node1，再启动其他节点就行了。

PXC集群只有一个节点，关闭了这个节点的容器，下次还能启动起来吗?

当然是可以的，因为PXC里只有一个节点，那么这个节点一定是按照主节点启动的，所以启动它的时候，它会启动MySQL服务，还创建出PXC集群。即便关闭了容器，下次再启动还是这个步骤，不会出现启动故障。如果说PXC集群是由多个节点组成的，node1停掉了，其他节点都正常运行。这时候启动node1是会出现闪退的，node1刚启动几秒钟就挂了。这是因为node2等一些节点正在现有的PXC中运行，这时候你启动node1，再创建一个同名的PXC集群，肯定会引发冲突啊。所以node1就闪退了。

遇到这种情况，正确的处理办法是，把node1容器删除。别紧张，没让你删除v1数据卷，所以数据丢不了。然后用从节点的命令方式创建一个node1，启动参数中与某个节点同步的设置就随便选择一个现在运行的PXC节点，然后Node1就能启动了。

关于搭建技术体系，深入学习方面的感言

本门课程的数据库部分只讲到数据库集群怎么搭建，如果想要完全驾驭数据库，仅凭这点知识还是远远不够的。像我在PXC免费课中讲到的，我们首先要有一个明确的知识体系，下一步就是做知识分解，一点点把技术拼图给完成。我见过非常多，东学一下，西学一下的人，即便有10年的工作经验，依然难成大器。比如说前些年VR挺火的，现在区块链技术也挺火的。我是个快要毕业的大学生，我想学这些技术帮我找一份好的工作。其实这种想法不是不对，但是过于急功近利。我先把话题说的远一点，咱们一会儿再回来说技术规划的事情。

在2013年，雷军和董明珠打了个赌，如果小米在5年内营业额超过格力，董明珠输给雷军10个亿，反之也是如此。两家企业相比较，大家更看好雷军的小米。这是因为中国的制造业利润太薄，制造业盈利存在上限的天花板，所以我们能准确预测出一家制造业企业未来的盈利空间。但是互联网公司就不是这样，想象空间太巨大。比如说马云的阿里巴巴，从1999年创建，到2014年上市，用了15年时间。刘强东的京东商城，差不多也用了15年时间上市。但是到了黄铮的拼多多这里，只用了3年时间就完成了上市，他还一举超过刘强东，成为中国第六富豪。IT界一夜成名的还有滴滴打车的程维、美团的王兴等等。

也正是IT互联网公司造富神话太多，也就越来越多的资本涌入IT圈。有的资本公司做长线，但是更多的资本公司炒短线。先投资一家小公司，然后再制造行业舆论和技术导向，当人们被技术忽悠的疯狂的时候，有人肯接盘了，这时候再把投资的这家小公司卖掉，割一波韭菜就撤。于是我们看到太多太多被热炒的技术，没过1年时间就无人问津了。比如2014年的iOS技术，2015年的VR技术等等。当年VR技术被捧到天上，王雪红一度被业界认为会用VR让HTC翻身，可是到现在我们再也听不到HTC这家公司的新闻了。活着还是倒闭了，我们都不知道，大家也不关心。

所以我们选择一个技术方向的时候，首先要看这个领域是不是有太多的水分，是不是有太多的热钱。比如今年3月的时候，真格基金的徐小平发了一个微博说自己非常看好以太坊技术，于是短短时间，中国各家资本公司纷纷涌入区块链领域。甚至还出现了2万块钱的区块链讲师速成班，不管懂不懂计算机，都能给你在短期内培养成区块链专家，然后你再去培训机构忽悠人，月薪两万，割学技术大学生的韭菜。我想，徐小平在微博上把区块链捧到天上，何尝不是一种资本炒作的手法呢。

最后说回到技术领域这块。比如你是一个大学生，想投身一个有发展的技术方向，那么不妨拿起手机看看那些常用的APP上都用到了什么技术，这才是能落地能普及的。比如说刷抖音很容易上瘾，上划一条是你喜欢的视频，再上划一下，又是你感兴趣的视频。还有每个人打开淘宝APP，首页的内容都不一样，都是你喜欢的商品，看什么东西都想卖。淘宝也很绝，即使你不花钱买东西，只要你点击看了一个你喜欢的商品，从此以后，你各项爱好都被淘宝所感知了。这种技术是大数据上的协同过滤。说简单一点，就是掌握了你很少资料的情况下，去分析跟你有相同行为的人的喜好，于是推算出你的喜好。比如你点击了某个牌子的运动鞋，那么淘宝的协同过滤就会计算购买了这个运动鞋的用户，还买了什么东西，这些用户共性的东西，就是你的爱好。所以你再打开淘宝APP，看到的都是你喜欢的东西。抖音啊，今日头条啊，都是这样的技术。所以你学大数据，这个技术是能落地的，太多产品在用这个技术了。相比较而言，区块链和人工智能就显得太高冷了。不是说技术不好，但是距离大规模普及还是有很远的距离。比如说区块链吧，上半年好多企业都在炒作，不跟区块链挂钩好像都不是科技企业，甚至南方有家茶叶公司，把名字都改成了带有区块链字眼。这就像荷兰的郁金香泡沫，当人们都意识到郁金香根本就不值那么高价格的时候，于是泡沫就破裂了。比如极路由公司，把自己的路由器搞成了能挖矿，每天平均赚4块钱。我们知道1万块钱存到支付宝上，一天也产生不了1块钱的利益。现在几百块钱的极路由每天产生4块钱的收益，不就相当你花几百块钱享受几万块钱的利息收益吗，多让人疯狂啊。极路由一边忽悠消费者，一边又去忽悠投资人，还跟互联网金融公司合作，弄了多种套餐玩法。投资人和消费者都被忽悠热血沸腾，拿出十几万块钱，买极路由器，幻想自己在家当地主，天天几百块钱收入。但是不到半年时间，极路由的技术+金融戏法就玩不下去了，老板欠了3.5个亿跑路了。同学们也不妨搜索一下因为区块链破产的消费者和炒作者。你投身这样的领域觉得靠谱吗?

所以选择技术规划一定要挑选成熟的，应用范围广泛的。比如大数据、Java体系、前端体系、Python体系等等。所以就像我在免费课里说到的，你选择好一个发力的方向，剩下的就是如何分解知识点了。比如说，你学了Java的一部分，没去深挖微服务架构和集群架构。如果你只停留在做单体项目，比如说SSM实现一个管理项目，VUE做一个网站等等，那我可以保证，你在IT开发这个领域很难坚持10年。经常听说，做开发30岁要转行，公司里几乎看不见40岁还做开发的人。其中的原因很多人没讲到关键点，作为创业者和企业家的角度来看，如果一个开发者，技术只停留在用SSM架构套各种业务，今天做一个管理系统，明天做一个管理系统。因为单体项目很难支撑高并发，所以你做的项目基本都是在小公司内部使用。随着你工作年限增加，对企业来说用人成本也就变高了。用一个2年经验的开发者，也会使用SSM做项目，成本还低，那为什么不用这样的人呢。所以你就被企业无情的抛弃了，根本原因是技术停滞，成本太高。这时候你再想跳槽到大企业，技术和年龄都达不到要求了。所以我建议年轻人，参加工作以后，虽然岗位有界限，但是技术无界限。前端、后台、数据库、原型设计、项目管理，多少都要学一些，没人教，就自己总结。只有技术全才，才能胜任技术团队的管理者，才不会被企业淘汰。

其实我见过很多项目都是被不称职的管理者给弄黄的。也许你也有同感，这个项目A公司做过，B公司做过，现在客户找到我们又要重做一版，这是为什么呢？项目开发的失败率也太高了吧。这通常是项目管理者技术不过关导致的，比如说A客户去南方考察，觉得某公司的管理系统特别好，我想也做一套。于是找到了你，A客户是大老板，下面具体的业务细节说不太清楚，脑子里也没有清晰的想法，反正想到什么就东一句、西一句的，然后你贸然选用了瀑布模型开发。经过了需求分析、文档设计，总算进入到了开发。经过两个月的开发，终于可以拿出半成品给A客户了，这时候A客户说你做的不是他想要的东西，要不你推到重做把，要不我找别的公司去做。然后别的公司做了，A客户依然不满意，于是开启了无限的重做模式。

究其原因还是管理者用错了开发模型，瀑布模型看似合理，但是问题在于编码阶段太靠后，等拿出半成品已经小半年时间过去了。因此瀑布模型适用于那种能提出明确需求，而且到每一处细节的客户。像A客户这样的人，我们应该用螺旋模型去开发，小规模快速迭代，只开发最核心模块，短期内就拿出演示品跟客户确认。这种小步快跑的策略适合提不出具体需求和需求不明确的场合。所以说，同学，以后你想做管理者，是不是得了解“软件工程”的知识啊!

这几年我在教育部举办的互联网+创新创业大赛预赛和决赛当评委的时候，跟很多大学生创业的同学讲到了，你的创业想法很好，但是很可能项目会作废了。起初他们不相信，觉得自己拿到风投的钱，几个学软件的同学会编程，这就足够了。于是次年的时候，我又见到了这些同学，参赛的题目又换了。我就好奇的问他们，去年的创业项目做成了吗？他们跟我说，创业项目做砸了。呵呵，果然是这样。因为他们的团队里缺少了技术方面的全才。仅靠专才是不能完成项目开发的。我再举一个例子，某公司的开发者老张，他是做后端比较擅长，因为工作年头比较长了，于是就被提拔成了项目经理。有一天，日方客户发来一封邮件，说某模块的业务流程要修改，看看中方这边多长时间能实现，费用是多少。于是老张就召集团队的人开了一次会。老张擅长后台开发，这次需求觉得没什么难度，然后问了问数据库的哥们和前端的哥们，他们说也好实现，然后老张就承诺日方两天就能做完。但是过于乐观的态度，很容易引来噩梦般的后果。数据库的哥们和前端的哥们，写代码的时候才发现实现难度很大，最后花了半个月才做完，浪费了大量的加班费用。因为老张不懂前端，也不懂数据库集群，觉得底下人说什么就代表什么。由此看出，管理者技术必须全面，不能人云亦云，必须保持自己的判断力。

说了这么多，也非常感谢大家在慕课网上支持我的课程，后续我还会在慕课网上录制更多的实战课，将我们的技术体系拼图给逐步完成。目前正在录制Python的课程和其他的开发课程，请大家稍加等待。下面列出来我在慕课网上所有课程的连接，以供大家学习。

《MySQL数据库集群-PXC方案》

https://coding.imooc.com/class/274.html

《Docker环境下的前后端分离项目部署与运维》

https://coding.imooc.com/class/219.html

安装Redis，配置RedisCluster集群

1. 安装Redis镜像

```
docker pull yyyytttwwww/redis
```

2. 创建net2网段

```
docker network create --subnet=172.19.0.0/16 net2
```

3. 创建6节点Redis容器

```
docker run -it -d --name r1 -p 5001:6379 --net=net2 --ip 172.19.0.2 redis bash
docker run -it -d --name r2 -p 5002:6379 --net=net2 --ip 172.19.0.3 redis bash
docker run -it -d --name r3 -p 5003:6379 --net=net2 --ip 172.19.0.4 redis bash
docker run -it -d --name r4 -p 5004:6379 --net=net2 --ip 172.19.0.5 redis bash
docker run -it -d --name r5 -p 5005:6379 --net=net2 --ip 172.19.0.6 redis bash
```

注意：redis配置文件里必须要设置bind 0.0.0.0，这是允许其他IP可以访问当前redis。如果不设置这个参数，就不能组建Redis集群。

4. 启动6节点Redis服务器

```
#进入r1节点
docker exec -it r1 bash
cp /home/redis/redis.conf /usr/redis/redis.conf
cd /usr/redis/src
./redis-server ../redis.conf
#进入r2节点
docker exec -it r2 bash
cp /home/redis/redis.conf /usr/redis/redis.conf
cd /usr/redis/src
./redis-server ../redis.conf
#进入r3节点
docker exec -it r3 bash
cp /home/redis/redis.conf /usr/redis/redis.conf
cd /usr/redis/src
./redis-server ../redis.conf
#进入r4节点
docker exec -it r4 bash
cp /home/redis/redis.conf /usr/redis/redis.conf
cd /usr/redis/src
./redis-server ../redis.conf
#进入r5节点
docker exec -it r5 bash
cp /home/redis/redis.conf /usr/redis/redis.conf
cd /usr/redis/src
./redis-server ../redis.conf
#进入r6节点
docker exec -it r6 bash
cp /home/redis/redis.conf /usr/redis/redis.conf
cd /usr/redis/src
./redis-server ../redis.conf
```

5. 创建Cluster集群

```
#在r1节点上执行下面的指令
cd /usr/redis/src
mkdir -p ../cluster
cp redis-trib.rb ../cluster/
cd ../cluster
#创建Cluster集群
./redis-trib.rb create --replicas 1 172.19.0.2:6379 172.19.0.3:6379
172.19.0.4:6379 172.19.0.5:6379 172.19.0.6:6379 172.19.0.7:6379
```

打包部署后端项目

1. 进入人人开源后端项目，执行打包（修改配置文件，更改端口，打包三次生成三个JAR文件）

```
mvn clean install -Dmaven.test.skip=true
```

2. 安装Java镜像

```
docker pull java
```

3. 创建3节点Java容器

```
#创建数据卷，上传JAR文件
docker volume create j1
#启动容器
docker run -it -d --name j1 -v j1:/home/soft --net=host java
#进入j1容器
docker exec -it j1 bash
#启动Java项目
nohup java -jar /home/soft/renren-fast.jar

#创建数据卷，上传JAR文件
docker volume create j2
#启动容器
docker run -it -d --name j2 -v j2:/home/soft --net=host java
#进入j1容器
docker exec -it j2 bash
#启动Java项目
nohup java -jar /home/soft/renren-fast.jar


#创建数据卷，上传JAR文件
```

```
docker volume create j3
#启动容器
docker run -it -d --name j3 -v j3:/home/soft --net=host java
#进入j1容器
docker exec -it j3 bash
#启动Java项目
nohup java -jar /home/soft/renren-fast.jar
```

4. 安装Nginx镜像

```
docker pull nginx
```

5. 创建Nginx容器，配置负载均衡

   宿主机上/home/n1/nginx.conf配置文件内容如下：

```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;
```

```
    proxy_redirect          off;
    proxy_set_header        Host $host;
    proxy_set_header        X-Real-IP $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    client_max_body_size    10m;
    client_body_buffer_size   128k;
    proxy_connect_timeout   5s;
    proxy_send_timeout      5s;
    proxy_read_timeout      5s;
    proxy_buffer_size        4k;
    proxy_buffers           4 32k;
    proxy_busy_buffers_size  64k;
    proxy_temp_file_write_size 64k;

    upstream tomcat {
        server 192.168.99.104:6001;
        server 192.168.99.104:6002;
        server 192.168.99.104:6003;
    }
    server {
        listen       6101;
        server_name  192.168.99.104;
        location / {
            proxy_pass   http://tomcat;
            index  index.html index.htm;
        }
    }
}
```

创建第1个Nginx节点

```
docker run -it -d --name n1 -v /home/n1/nginx.conf:/etc/nginx/nginx.conf --
net=host --privileged nginx
```

宿主机上/home/n2/nginx.conf配置文件内容如下：

```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;
```

```
events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    proxy_redirect          off;
    proxy_set_header        Host $host;
    proxy_set_header        X-Real-IP $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    client_max_body_size    10m;
    client_body_buffer_size   128k;
    proxy_connect_timeout   5s;
    proxy_send_timeout      5s;
    proxy_read_timeout      5s;
    proxy_buffer_size        4k;
    proxy_buffers           4 32k;
    proxy_busy_buffers_size  64k;
    proxy_temp_file_write_size 64k;

    upstream tomcat {
        server 192.168.99.104:6001;
        server 192.168.99.104:6002;
        server 192.168.99.104:6003;
    }
    server {
        listen       6102;
```

```
        server_name  192.168.99.104;
        location / {
            proxy_pass   http://tomcat;
            index  index.html index.htm;
        }
    }
}
```

创建第2个Nginx节点

```
docker run -it -d --name n2 -v /home/n2/nginx.conf:/etc/nginx/nginx.conf --
net=host --privileged nginx
```

6. 在Nginx容器安装Keepalived

```
#进入n1节点
docker exec -it n1 bash
#更新软件包
apt-get update
#安装VIM
apt-get install vim
#安装Keepalived
apt-get install keepalived
#编辑Keepalived配置文件(如下)
vim /etc/keepalived/keepalived.conf
#启动Keepalived
service keepalived start
```

```
vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 123456
    }
    virtual_ipaddress {
        192.168.99.151
    }
```

```
}
virtual_server 192.168.99.151 6201 {
    delay_loop 3
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
    real_server 192.168.99.104 6101 {
        weight 1
    }
}
```

```
#进入n1节点
docker exec -it n2 bash
#更新软件包
apt-get update
#安装VIM
apt-get install vim
#安装Keepalived
apt-get install keepalived
#编辑Keepalived配置文件(如下)
vim /etc/keepalived/keepalived.conf
#启动Keepalived
service keepalived start
```

```
vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 123456
    }
    virtual_ipaddress {
        192.168.99.151
    }
}
virtual_server 192.168.99.151 6201 {

    delay_loop 3
```

```
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
    real_server 192.168.99.104 6102 {
        weight 1
    }
}
```

打包部署后端项目

1. 在前端项目路径下执行打包指令

```
npm run build
```

2. build目录的文件拷贝到宿主机的/home/fn1/renren-vue、/home/fn2/renren-vue、/home/fn3/renren-vue的目录下面
3. 创建3节点的Nginx，部署前端项目

   宿主机/home/fn1/nginx.conf的配置文件

```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
```

```
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

        proxy_redirect          off;
        proxy_set_header        Host $host;
        proxy_set_header        X-Real-IP $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        client_max_body_size    10m;
        client_body_buffer_size   128k;
        proxy_connect_timeout   5s;
        proxy_send_timeout      5s;
        proxy_read_timeout      5s;
        proxy_buffer_size        4k;
        proxy_buffers           4 32k;
        proxy_busy_buffers_size  64k;
        proxy_temp_file_write_size 64k;

        server {
                listen 6501;
                server_name  192.168.99.104;
                location  /  {
                        root  /home/fn1/renren-vue;
                        index  index.html;
                }
        }
}
```

```
#启动第fn1节点
docker run -it -d --name fn1 -v /home/fn1/nginx.conf:/etc/nginx/nginx.conf -v
/home/fn1/renren-vue:/home/fn1/renren-vue --privileged --net=host nginx
```

宿主机/home/fn2/nginx.conf的配置文件

```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;
```

```
events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    proxy_redirect          off;
    proxy_set_header        Host $host;
    proxy_set_header        X-Real-IP $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    client_max_body_size    10m;
    client_body_buffer_size   128k;
    proxy_connect_timeout   5s;
    proxy_send_timeout      5s;
    proxy_read_timeout      5s;
    proxy_buffer_size        4k;
    proxy_buffers           4 32k;
    proxy_busy_buffers_size  64k;
    proxy_temp_file_write_size 64k;

    server {
        listen 6502;
        server_name  192.168.99.104;
        location  /  {
            root  /home/fn2/renren-vue;
            index  index.html;
        }
```

```
    }
}
```

```
#启动第fn2节点
docker run -it -d --name fn2 -v /home/fn2/nginx.conf:/etc/nginx/nginx.conf -v
/home/fn2/renren-vue:/home/fn2/renren-vue --privileged --net=host nginx
```

宿主机/home/fn3/nginx.conf的配置文件

```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    proxy_redirect          off;
    proxy_set_header        Host $host;
    proxy_set_header        X-Real-IP $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    client_max_body_size    10m;

    client_body_buffer_size   128k;
```

```
    proxy_connect_timeout   5s;
    proxy_send_timeout      5s;
    proxy_read_timeout      5s;
    proxy_buffer_size        4k;
    proxy_buffers           4 32k;
    proxy_busy_buffers_size  64k;
    proxy_temp_file_write_size 64k;

    server {
        listen 6503;
        server_name  192.168.99.104;
        location  /  {
            root  /home/fn3/renren-vue;
            index  index.html;
        }
    }
}
```

启动fn3节点

```
#启动第fn3节点
docker run -it -d --name fn3 -v /home/fn3/nginx.conf:/etc/nginx/nginx.conf -v
/home/fn3/renren-vue:/home/fn3/renren-vue --privileged --net=host nginx
```

4. 配置负载均衡

宿主机/home/ff1/nginx.conf配置文件

```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;


    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
```

```
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    proxy_redirect          off;
    proxy_set_header        Host $host;
    proxy_set_header        X-Real-IP $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    client_max_body_size    10m;
    client_body_buffer_size   128k;
    proxy_connect_timeout   5s;
    proxy_send_timeout      5s;
    proxy_read_timeout      5s;
    proxy_buffer_size        4k;
    proxy_buffers           4 32k;
    proxy_busy_buffers_size  64k;
    proxy_temp_file_write_size 64k;

    upstream fn {
        server 192.168.99.104:6501;
        server 192.168.99.104:6502;
        server 192.168.99.104:6503;
    }
    server {
        listen       6601;
        server_name  192.168.99.104;
        location / {
            proxy_pass   http://fn;
            index  index.html index.htm;
        }
    }
}
```

```
#启动ff1节点
docker run -it -d --name ff1 -v /home/ff1/nginx.conf:/etc/nginx/nginx.conf --
net=host --privileged nginx
```

宿主机/home/ff2/nginx.conf配置文件

```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    proxy_redirect          off;
    proxy_set_header        Host $host;
    proxy_set_header        X-Real-IP $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    client_max_body_size    10m;
    client_body_buffer_size   128k;
    proxy_connect_timeout   5s;
    proxy_send_timeout      5s;

    proxy_read_timeout      5s;
```

```
    proxy_buffer_size        4k;
    proxy_buffers           4 32k;
    proxy_busy_buffers_size  64k;
    proxy_temp_file_write_size 64k;

    upstream fn {
        server 192.168.99.104:6501;
        server 192.168.99.104:6502;
        server 192.168.99.104:6503;
    }
    server {
        listen       6602;
        server_name  192.168.99.104;
        location / {
            proxy_pass   http://fn;
            index  index.html index.htm;
        }
    }
}
```

```
#启动ff2节点
docker run -it -d --name ff2 -v /home/ff2/nginx.conf:/etc/nginx/nginx.conf --
net=host --privileged nginx
```

5. 配置双机热备

```
#进入ff1节点
docker exec -it ff1 bash
#更新软件包
apt-get update
#安装VIM
apt-get install vim
#安装Keepalived
apt-get install keepalived
#编辑Keepalived配置文件(如下)
vim /etc/keepalived/keepalived.conf
#启动Keepalived
service keepalived start
```

```
vrrp_instance VI_1 {
    state MASTER
```

```
    interface ens33
    virtual_router_id 52
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 123456
    }
    virtual_ipaddress {
        192.168.99.152
    }
}
virtual_server 192.168.99.151 6701 {
    delay_loop 3
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
    real_server 192.168.99.104 6601 {
        weight 1
    }
}
```

```
#进入ff1节点
docker exec -it ff2 bash
#更新软件包
apt-get update
#安装VIM
apt-get install vim
#安装Keepalived
apt-get install keepalived
#编辑Keepalived配置文件(如下)
vim /etc/keepalived/keepalived.conf
#启动Keepalived
service keepalived start
```

```
vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 52
    priority 100
```

```
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 123456
    }
    virtual_ipaddress {
        192.168.99.152
    }
}
virtual_server 192.168.99.151 6701 {
    delay_loop 3
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
    real_server 192.168.99.104 6602 {
        weight 1
    }
}
```