## **1. 准备工作**

* 建议从[阿里云](https://dc.console.aliyun.com/next/index)购买域名
* 建议从[阿里云](https://ecs.console.aliyun.com/)购买ECS服务器
* 建议从[阿里云](https://bsn.console.aliyun.com/)进行备案

## **2. 配置ECS服务器**

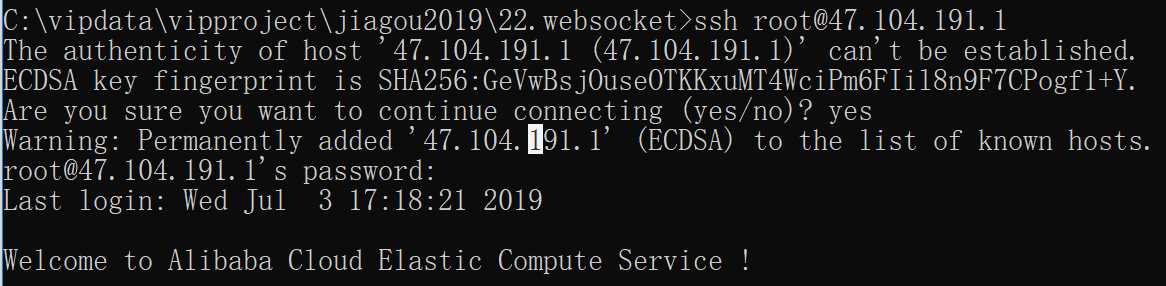
* 配置root密码
* 配置安全规则

## **3. 登录服务器**

### **3.1 命令行登录**

ssh root@47.104.191.1

* 当本机获得服务器公钥指纹，但是无法确认服务器安全性的时候会提示你是否要继续连接



### **3.2 SSH登录**

#### **3.2.1 新建用户**

adduser devops

#### **3.2.1 授与sudo权限**

visudo

devops ALL=(ALL:ALL) ALL

* 1 ALL 为允许使用sudo命令的主机
* 2 ALL devops可以以任意用户身份来执行命令
* 3 ALL devops可以以任意组身份来执行命令
* 4 ALL devops可以执行任意命令

以下命令表示允许test用户从任何主机登录，以root的身份执行/usr/sbin/useradd命令

test ALL=(root) /usr/sbin/useradd

### **3.3 配置无密码登录**

#### **3.3.1 客户端**

ssh-keygen -t rsa -b 4096 -C "83687401@qq.com"

cat .ssh/id\_rsa.pub

#### **3.3.2 服务器端**

ssh-keygen -t rsa -b 4096 -C "83687401@qq.com"

vi ~/.ssh/authorized\_keys

chmod 644 ~/.ssh

chmod 600 ~/.ssh/authorized\_keys

### **3.4 修改SSH端口**

/etc/ssh/sshd\_config

Port 22222

systemctl restart sshd.service



* 出方向: 是指ECS实例访问内网中其它实例或者公网的资源
* 入方向: 是指内网中的其它ECS实例 或公网上的资源访问ECS实例

## **4. docker**

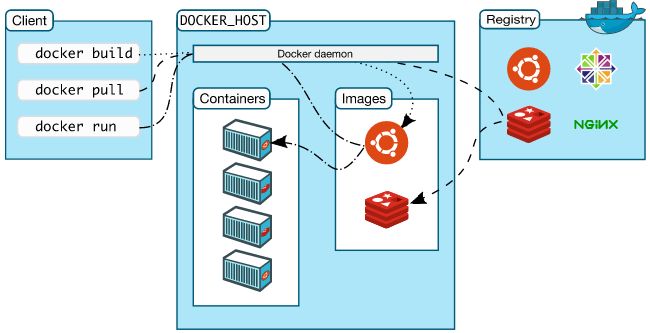
### **4.1 为什么使用docker?**

* 境部署是所有团队都必须面对的问题，随着系统越来越大，依赖的服务也越来越多，例如：Web服务器 + MySql数据库 + Redis缓存等
* 依赖服务很多，本地搭建一套环境成本越来越高，初级人员很难解决环境部署中的一些问题
* 服务的版本差异及OS的差异都可能导致线上环境BUG，项目引入新的服务时所有人的环境需要重新配置
* 任何安装过Docker的机器都可以运行这个容器可以获得同样的结果,从而完全消除了不同环境，不同版本可能引起的各种问题

### **4.2 docker中的概念**

* Docker有三个基本概念：镜像(image),容器(container),仓库(repository)

| **概念** | **说明** |
| --- | --- |
| 镜像(image) | 镜像中包含有需要运行的文件。镜像用来创建container，一个镜像可以运行多个container；镜像可以通过Dockerfile创建，也可以从Docker hub/registry上下载 |
| 容器(container) | 容器是Docker的运行组件，启动一个镜像就是一个容器，容器是一个隔离环境，多个容器之间不会相互影响，保证容器中的程序运行在一个相对安全的环境中 |
| 仓库(repository) | 共享和管理Docker镜像，用户可以上传或者下载上面的镜像，官方地址为 registry.hub.docker.com/ （类似于github对源代码的管理），也可以搭建自己私有的Docker registry |



### **4.3 常见docker命令**

| **概念** | **说明** |
| --- | --- |
| 拉取镜像 | docker pull centos |
| 创建新容器并运行 | docker run --name mynginx -d nginx:latest |
| 启动容器 | docker start container\_name/container\_id |
| 停止容器 | docker stop container\_name/container\_id |
| 重启容器 | docker restart container\_name/container\_id |
| 在容器中开启交互终端 | docker exec -i -t container\_id /bin/bash |
| 使用当前目录Dockerfile创建镜像,标签为xxx:v1 | docker build -t xxx:v1 |

### **4.4 安装docker**

* docker分为企业版(EE)和社区版(CE)

$ yum install -y yum-utils device-mapper-persistent-data lvm2

$ yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

$ yum install -y docker-ce docker-ce-cli containerd.io

### **4.5 启动docker**

$ systemctl start docker

### **4.6 查看docker版本**

$ docker version

$ docker info

### **4.7 docker-compose**

* 实际项目不可能只单单依赖于一个服务，例如一个常见的Web项目可能依赖于: 静态文件服务器，应用服务器，Mysql数据库等
* 我们可以通过分别启动单个镜像，并把镜像绑定到本地对应端口的形式进行部署，达到容器可通信的目的
* 但是为了更方便的管理多容器的情况，官方提供了docker-compose的方式
* docker-compose是Docker的一种编排服务，是一个用于在 Docker 上定义并运行复杂应用的工具，可以让用户在集群中部署分布式应用
* 一个项目可以由多个服务（容器）关联而成，compose 面向项目进行管理，通过子命令对项目中的一组容器进行便捷地生命周期管理
* compose中有两个重要的概念
  + 服务 (service)：一个应用的容器，实际上可以包括若干运行相同镜像的容器实例
  + 项目 (project)：由一组关联的应用容器组成的一个完整业务单元，在docker-compose.yml 文件中定义

curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

docker-compose --version

### **4.8 阿里云加速**

sudo mkdir -p /etc/docker

sudo tee /etc/docker/daemon.json <<-'EOF'

{

"registry-mirrors": ["https://fwvjnv59.mirror.aliyuncs.com"]

}

EOF

# 重载所有修改过的配置文件

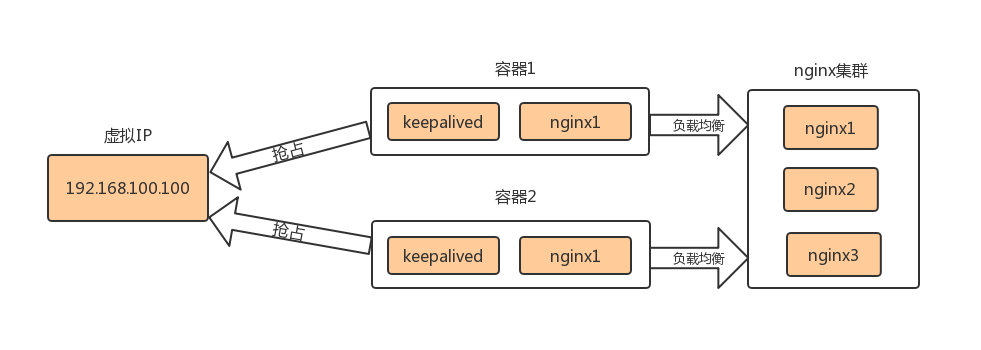
sudo systemctl daemon-reload

sudo systemctl restart docker

## **5.整体架构**

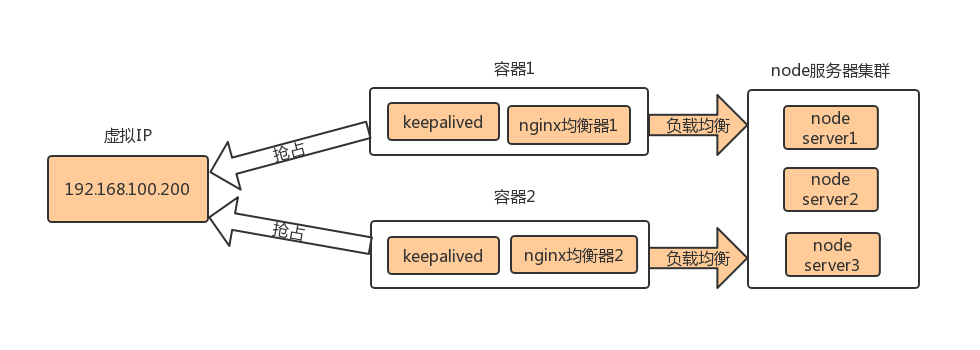
### **5.1 前端架构**

* 用户在浏览器里输入前端项目的虚拟IP地址
* 这个虚拟IP可能会被某个keepalived容器抢占
* 这个keepalived容器会让负载均衡的nginx服务器请求前端项目的nginx集群
* 前端项目调用的接口是后端项目的虚拟IP



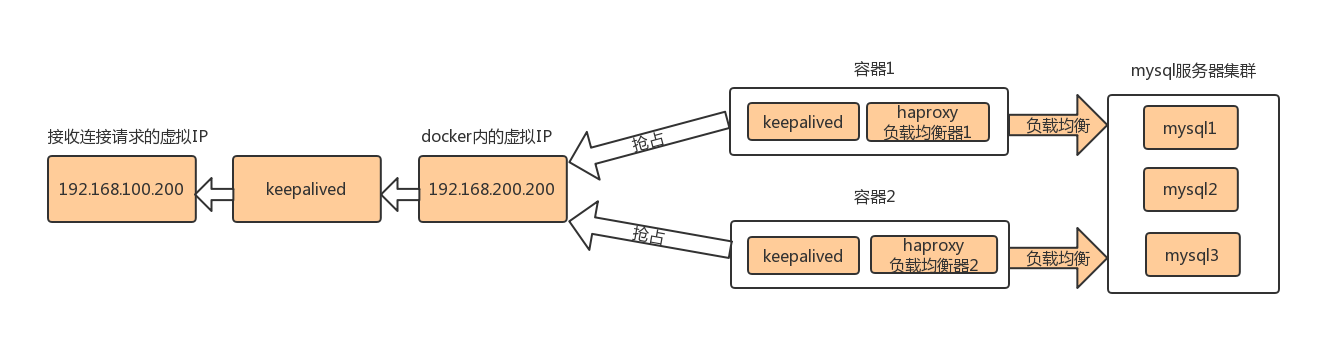
### **5.2 后端架构**

* 前端项目会访问这个后端的虚拟IP
* 这个虚拟IP可能会被某个keepalived容器抢占
* 这个请求会转发到keepalived容器上的负载均衡节点上
* 负载均衡节点会把请求转发的node集群的某个节点上
* node服务器可能需要访问mysql、mongodb、redis服务器



### **5.3 mysql数据库集群**

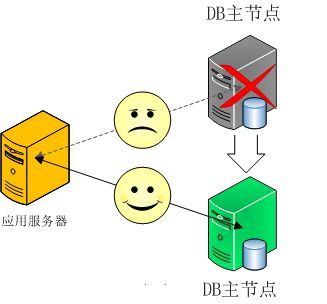
* 数据库的虚拟IP是192.168.100.200,web服务器如果想访问数据库需要连接这个IP
* 虚拟IP收到请求后会把请求转交给docker容器内的一个虚拟IP192.168.200.200上
* Docker内的虚拟IP不能被外网使用，所以需要借助宿主机keepalived映射成外网可以访问的虚拟IP
* 此处配置了双机热备方案，如果第一个容器抢占了这个虚拟IP192.168.100.200
* 这个虚拟IP会把请求转发给此容器内的haproxy节点上
* haproxy节点会把请求转发给MYSQL数据库集群中的某个节点上



### **5.4 mongodb数据库集群**

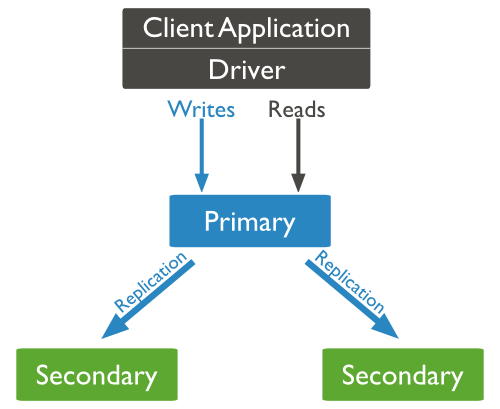
#### **5.4.1 主从复制**

* 主从复制是一个简单的数据库同步备份的集群技术
* 在数据库集群中要明确知道谁是主服务器，主服务器只有一台
* 从服务器要知道自己的数据源也就是知道自己的主服务器是谁



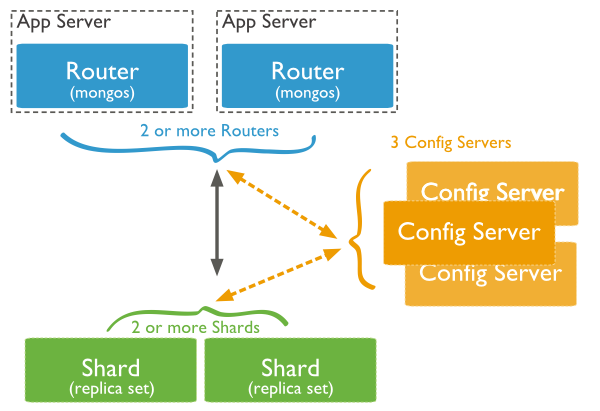
#### **5.4.2 副本集**

* MongoDB复制是将数据同步在多个服务器的过程。
* 复制提供了数据的冗余备份，并在多个服务器上存储数据副本，提高了数据的可用性， 并可以保证数据的安全性。
* 复制还允许您从硬件故障和服务中断中恢复数据。



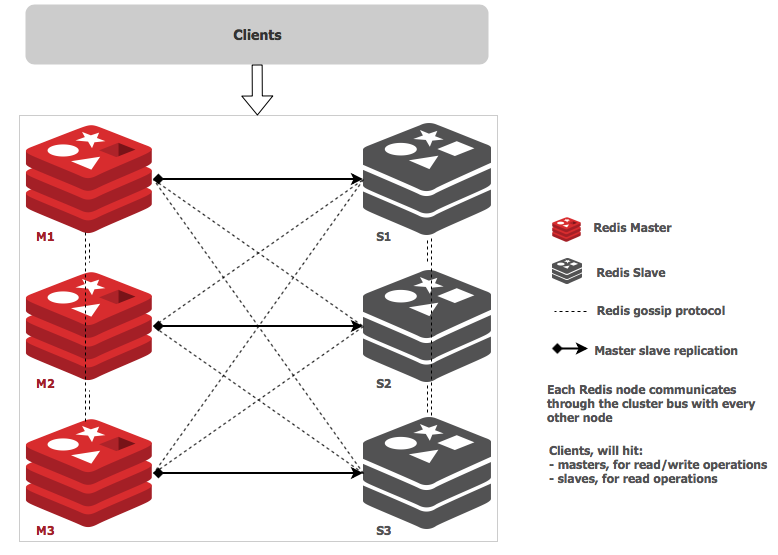
#### **5.4.3 分片**

* 在Mongodb里面存在另一种集群，就是分片技术,可以满足MongoDB数据量大量增长的需求
* 当MongoDB存储海量的数据时，一台机器可能不足以存储数据，也可能不足以提供可接受的读写吞吐量。这时，我们就可以通过在多台机器上分割数据，使得数据库系统能存储和处理更多的数据



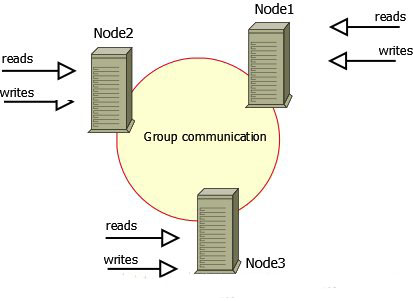
### **5.5 redis数据库集群**

* Redis-Cluster采用无中心结构，每个节点保存数据和整个集群状态,每个节点都和其他所有节点连接
* 所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽
* 客户端与redis节点直连,不需要中间proxy层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可



## **6.mysql集群**

* [percona-xtradb-cluster](https://hub.docker.com/r/percona/percona-xtradb-cluster/)
* [percona-xtradb-cluster官方文档](https://www.percona.com/doc/percona-xtradb-cluster/LATEST/install/docker.html)
* PXC的数据是强一致性的，要么所有节点都提交，要么都不提交



| **端口** | **描述** |
| --- | --- |
| 3306 | MYSQL服务端口 |
| 4444 | 请求全量同步(SST)接口 |
| 4567 | 数据库节点之间的通信接口 |
| 4568 | 请求增量同步(IST)端口 |

### **6.1 安装集群**

#### **6.1.1 下载镜像**

docker pull percona/percona-xtradb-cluster:5.6

docker tag percona/percona-xtradb-cluster:5.6 pxc

docker image rm percona/percona-xtradb-cluster:5.6

#### **6.1.2 创建内部网络**

docker network create --subnet=172.18.0.0/24 znet

docker network inspect znet

docker network rm znet

#### **6.1.3 创建docker卷**

docker volume create --name v1

docker volume create --name v2

docker volume create --name v3

#### **6.1.4 创建pxc容器**

docker run -d \

-p 3306:3306 \

-e MYSQL\_ROOT\_PASSWORD=123456 \

-e CLUSTER\_NAME=PXC \

-e XTRABACKUP\_PASSWORD=123456 \

-v v1:/var/lib/mysql \

--privileged \

--name=mysql1 \

--net=znet \

--ip 172.18.0.2 \

pxc

docker exec -it mysql1 bash

docker logs mysql1

docker run -d \

-p 3307:3306 \

-e MYSQL\_ROOT\_PASSWORD=123456 \

-e CLUSTER\_NAME=PXC \

-e XTRABACKUP\_PASSWORD=123456 \

-e CLUSTER\_JOIN=mysql1 \

-v v2:/var/lib/mysql \

--privileged \

--name=mysql2 \

--net=znet \

--ip 172.18.0.3 \

pxc

docker run -d \

-p 3308:3306 \

-e MYSQL\_ROOT\_PASSWORD=123456 \

-e CLUSTER\_NAME=PXC \

-e XTRABACKUP\_PASSWORD=123456 \

-e CLUSTER\_JOIN=mysql1 \

-v v3:/var/lib/mysql \

--privileged \

--name=mysql3 \

--net=znet \

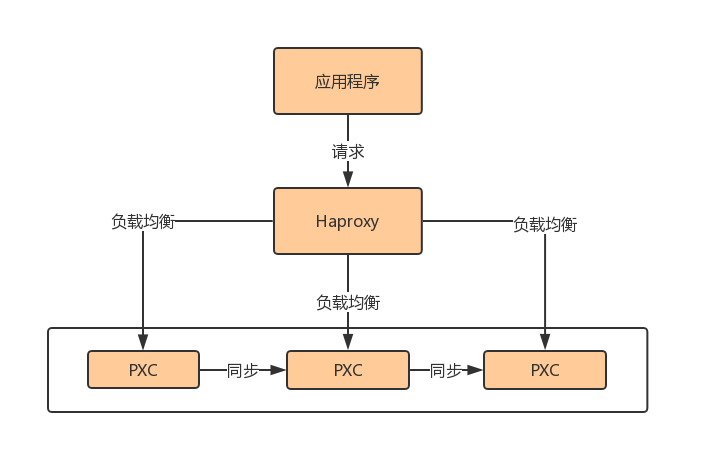
--ip 172.18.0.4 \

pxc

| **参数** | **含义** |
| --- | --- |
| -d | 服务后台运行 |
| -p | 映射端口号 |
| -e MYSQL\_ROOT\_PASSWORD=123456 | 指定容器内的数据库的root密码 |
| -e CLUSTER\_NAME=PXC | 集群的名称 |
| -e XTRABACKUP\_PASSWORD=123456 | 备份密码 |
| -v v1:/var/lib/mysql | 把容器内的/var/lib/mysql目录映射为宿主机的数据卷 |
| --privileged | 自动获取权限 |
| --name | 指定容器的名称 |
| --net | 指定加入的网络 |

### **6.2 负载均衡**

* 单节点处理所有请求负载高，性能差，所以我们要使用负载均衡
* 使用Haproxy做负载均衡，请求被均匀分发给每个节点，单节点负载低，性能好
* [Haproxy](https://zhangge/5125.html)只是一个转发器



#### **6.2.1 安装haproxy镜像**

docker pull haproxy

#### **6.2.2 创建配置文件**

touch /home/devops/haproxy/haproxy.cfg

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:123456

#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

#使用keepalive检测死链

option tcpka

#### **6.2.3 创建haproxy容器**

docker run -it -d -p 4001:8888 -p 4002:3306 -v /home/devops/haproxy:/usr/local/etc/haproxy --name haproxy1 --privileged --net=znet --ip 172.18.0.5 haproxy:2.0

haproxy -f /usr/local/etc/haproxy/haproxy.cfg

http://47.104.191.1:4001/dbs

admin 123456

### **6.3 keepalived双机热备**

* 虚拟IP在linux系统中一个网卡可以定义多个IP地址，然后把这些IP地址分配给对应的程序
* keepalived是用来强占虚拟IP的，在各自的haproxy容器中安装keepalived,用来强占虚拟IP
* 抢到虚拟IP的服务器叫做主服务器，没抢到的叫做备服务器
* 没抢到的就会处于等待的状态，然后通过心跳检测来检测主服务器是否正常，如果不正常则立刻抢占虚拟IP

### **6.3.1 安装keepalived**

* 安装keepalived必须要安装在haproxy所在的容器内

apt-**get** update

apt-**get** install -y keepalived

/etc/keepalived/keepalived.conf

rm /var/cache/apt/archives/lock

rm /var/lib/dpkg/lock

apt-**get** -y install vim

vim /etc/keepalived/keepalived.conf

docker cp keepalived.conf haproxy1:/etc/keepalived

vrrp\_instance VI\_1 {

state MASTER

interface eth0

virtual\_router\_id 100

priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 123456

}

virtual\_ipaddress {

172.18.0.201

}

}

| **字段** | **含义** |
| --- | --- |
| state | keepalived节点身份，master是主服务器，backup是备服务器，主服务器要抢点虚拟IP，备用服务器不抢占虚拟IP |
| interface | 网卡设备,docker 网卡在宿主机上可以访问，但其它地方访问不到,所以需要映射到局域网的虚拟IP上 |
| virtual\_router\_id | 虚拟路由标识，master和backup的虚拟路由标识必须一样，可以是0~255 |
| priority | master权重，权重越高越容易抢到虚拟IP |
| authentication | 主从服务器验证方式，主务必须使用相同的密码才能正常通信 |
| virual\_ipaddress | 虚拟IP，可以设置多个虚拟IP |

启动keepalive后，宿主机就可以ping通虚拟IP

service keepalived start

apt-**get** install -y inetutils-ping

apt-**get** install net-tools -y

ping 172.18.0.201

### **6.3.2 haproxy2**

docker run -it -d -p 4003:8888 -p 4004:3306 -v /home/devops/haproxy:/usr/local/etc/haproxy --name haproxy2 --privileged --net=znet --ip 172.18.0.6 haproxy:2.0

## **7. 布署Egg.js**

* [eggjs](https://eggjs.org/zh-cn/intro/quickstart.html)

### **7.1 编写项目**

mkdir zhufeng\_egg.js

cnpm init egg --type=simple

cnpm install

cnpm start / npm run dev / npm test

### **7.2 部署项目**

#### **7.2.1 package.json**

* 把package.json中start这行里命令里的--daemon去掉,在Docker里eggjs应用要在前台运行

#### **7.2.2 Dockerfile**

* 在本地应用的根目录下建一个名为Dockerfile的文件

# 设置基础镜像,如果本地没有该镜像，会从Docker.io服务器pull镜像

FROM node:12

# 创建app目录

RUN mkdir -p /usr/src/egg\_server

# 设置工作目录

WORKDIR /usr/src/egg\_server

# 拷贝package.json文件到工作目录

COPY package.json /usr/src/egg\_server/package.json

# 安装npm依赖(使用淘宝的镜像源)

RUN npm install --registry=https://registry.npm.taobao.org

# 拷贝所有源代码到工作目录

COPY . /usr/src/egg\_server

# 暴露容器端口

EXPOSE 7001

# 启动node应用

CMD npm start

* 1.拉取docker镜像
* 2.创建docker工作目录，并将package.json拷贝到docker里
* 3.安装npm依赖
* 4.将服务器上的应用拷贝到docker里
* 5.暴露docker容器的端口，然后启动node应用

#### **7.2.3 上传服务器**

* 使用ftp工具或git工具将整个应用上传到生产环境服务器
* 并使用终端连接到服务器，进入到服务器应用的目录下

#### **7.2.4 编译docker镜像**

docker build -t egg\_server .

#### **7.2.4 启动docker容器**

docker run -d --name egg\_server1 -p 7001:7001 --net=znet --ip 172.18.0.7 egg\_server

docker run -d --name egg\_server2 -p 7002:7001 --net=znet --ip 172.18.0.8 egg\_server

docker run -d --name egg\_server3 -p 7003:7001 --net=znet --ip 172.18.0.9 egg\_server

docker ps

curl -i localhost:7001

curl -i localhost:7002

curl -i localhost:7003

## **8. 布署nginx**

### **8.1 拉取官方的镜像**

docker pull nginx

### **8.2 启动nginx**

docker run -d --name nginx1 -p 80:80 nginx

docker ps

### **8.3 创建目录**

$ mkdir -p ~/nginx/www ~/nginx/logs ~/nginx/conf

| **目录名** | **含义** |
| --- | --- |
| www | 目录将映射为 nginx 容器配置的虚拟目录 |
| logs | 目录将映射为 nginx 容器的日志目录 |
| conf | 目录里的配置文件将映射为 nginx 容器的配置文件 |

### **8.4 拷贝配置文件**

$ docker cp 09ffe6a26871:/etc/nginx/nginx.conf ~/nginx/conf

### **8.5 部署**

docker run -d -p 80:80 --name nginx1 -v ~/nginx/www:/usr/share/nginx/html -v ~/nginx/conf/nginx.conf:/etc/nginx/nginx.conf -v ~/nginx/conf/conf.d:/etc/nginx/conf.d -v ~/nginx/logs:/var/log/nginx --net=znet --ip 172.18.0.10 nginx

| **目录名** | **含义** |
| --- | --- |
| -p 80:80 | 将容器的 80 端口映射到主机的 80 端口 |
| --name nginx1 | 将容器命名为nginx1 |
| -v ~/nginx/www:/usr/share/nginx/html | 将我们自己创建的 www 目录挂载到容器的 /usr/share/nginx/html |
| -v ~/nginx/conf/nginx.conf:/etc/nginx/nginx.conf | 将我们自己创建的 nginx.conf 挂载到容器的 /etc/nginx/nginx.conf |
| -v ~/nginx/logs:/var/log/nginx | 将我们自己创建的 logs 挂载到容器的 /var/log/nginx |

/root/nginx/conf/conf.d/default.conf

upstream nodeservers {

server 172.18.0.7:7001;

server 172.18.0.8:7001;

server 172.18.0.9:7001;

}

server {

listen 80;

server\_name 47.104.191.1;

location / {

root /usr/share/nginx/html;

index index.html index.htm;

}

location /api {

proxy\_pass http://nodeservers;

}

}

$ cd ~/nginx/www

$ docker kill -s HUP nginx1

$ docker restart nginx1

curl http://47.104.191.1/api/login

curl http://47.104.191.1/

## **9. Ant Design Pro**

* [)Ant Design Pro]([https://github.com/ant-design/ant-design-pro是一个企业级中后台解决方案，在Ant](https://github.com/ant-design/ant-design-pro%E6%98%AF%E4%B8%80%E4%B8%AA%E4%BC%81%E4%B8%9A%E7%BA%A7%E4%B8%AD%E5%90%8E%E5%8F%B0%E8%A7%A3%E5%86%B3%E6%96%B9%E6%A1%88%EF%BC%8C%E5%9C%A8Ant) Design组件库的基础上，提炼出典型模板/业务组件/通用页等，在此基础上能够使开发者快速的完成中后台应用的开发

git clone https://github.com/ant-design/ant-design-pro.git --depth=1

cd ant-design-pro

cnpm i

cnpm run docker-prod:dev

## **10.mongodb集群**

* 在Docker环境上搭建一个MongoDB集群

### **10.1 容器**

| **集群角色** | **ContainerName** | **IP:port** |
| --- | --- | --- |
| Config Server | cfg\_1 | 172.18.0.11:27019 |
| Config Server | cfg\_2 | 172.18.0.12:27019 |
| Config Server | cfg\_3 | 172.18.0.13:27019 |
| Shard Server | shard1\_1 | 172.18.0.14:27018 |
| Shard Server | shard1\_2 | 172.18.0.15:27018 |
| Shard Server | shard1\_3 | 172.18.0.16:27018 |
| Shard Server | shard2\_1 | 172.18.0.17:27018 |
| Shard Server | shard2\_2 | 172.18.0.18:27018 |
| Shard Server | shard2\_3 | 172.18.0.19:27018 |
| Shard Server | shard3\_1 | 172.18.0.20:27018 |
| Shard Server | shard3\_2 | 172.18.0.21:27018 |
| Shard Server | shard3\_3 | 172.18.0.22:27018 |
| Mongos mongos\_1 | 172.18.0.23:27020 |  |
| Mongos mongos\_2 | 172.18.0.24:27020 |  |
| Mongos mongos\_3 | 172.18.0.25:27020 |  |

### **10.2 拉取镜像**

docker pull mongo:4.0.0

### **10.2 集群配置文件**

mkdir -p /home/devops/configsvr

mkdir -p /home/devops/shard1

mkdir -p /home/devops/shard2

mkdir -p /home/devops/shard3

mkdir -p /home/devops/mongos

#### **10.2.1 Config-Server 配置文件**

* 路径：/home/devops/configsvr/mongod.conf
* 说明：MongoDB v3.4 之后要求Config-Server也需要组成副本集形式

storage:

dbPath: /data/dbjournal:

enabled: truesystemLog:

destination: filelogAppend: truepath: /var/log/mongodb/mongod.lognet:

bindIp: 127.0.0.1processManagement:

timeZoneInfo: /usr/share/zoneinforeplication:

replSetName: cfgsharding:

clusterRole: configsvr

#### **10.2.3 Shard-Server 配置文件**

* 路径：/home/devops/shard1/mongod.conf
* 说明：此处配置3个分片为shard1,shard2,shard3;每个分片都需要组成副本集。
* shard2,shard3目录下配置文件同名，修改replSetName字段的值分别为’shard2’和’shard3’

storage:

dbPath: /data/db

journal:

enabled: truesystemLog:

destination: file

logAppend: true

path: /var/log/mongodb/mongod.lognet:

bindIp: 127.0.0.1processManagement:

timeZoneInfo: /usr/share/zoneinforeplication:

replSetName: shard1sharding:

clusterRole: shardsvr

#### **10.2.4 Mongos 配置文件**

* 路径：/home/dmc/mongos/mongos.conf
* 说明：mongos不需要存储因此去掉storage字段；可任意配置net.port字段，需要指定processManagement.fork为true以–fork方式启动
* sharding.configDB字段用于指定Config-Server集群地址，格式为[replSetName]/[config-server1:port],[config-server2:port]

systemLog:

destination: file

logAppend: true

path: /var/log/mongodb/mongos.lognet:

port: 27020

bindIp: 127.0.0.1processManagement:

fork: true

timeZoneInfo: /usr/share/zoneinfosharding:

configDB: cfg/172.18.0.11:27019,172.18.0.12:27019,172.18.0.13:27019

### **10.3 启动Docker容器**

#### **10.3.1 启动3个Config-Server容器**

docker run -d --name=cfg\_1 --network=znet --ip=172.18.0.11 -v /home/devops/configsvr:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

docker run -d --name=cfg\_2 --network=znet --ip=172.18.0.12 -v /home/devops/configsvr:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

docker run -d --name=cfg\_3 --network=znet --ip=172.18.0.13 -v /home/devops/configsvr:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

进入其中一个容器配置Config-Server副本集

# 宿主机

docker exec -it cfg\_1 bash

# 容器中

mongo --port 27019

# Mongo Shell中

rs.initiate({

"\_id":"cfg",

"members":[

{

"\_id":0,

"host":"172.18.0.11:27019"

},

{

"\_id":1,

"host":"172.18.0.12:27019"

},

{

"\_id":2,

"host":"172.18.0.13:27019"

}

]

})

#### **10.3.2 启动3\*3个Shard-Server容器**

* 说明：分片服务器启动后默认是以27018作为端口

##### **10.3.2.1 启动第一个分片 - shard1**

docker run -d --name=shard1\_1 --network=znet --ip=172.18.0.14 -v /home/devops/shard1:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

docker run -d --name=shard1\_2 --network=znet --ip=172.18.0.15 -v /home/devops/shard1:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

docker run -d --name=shard1\_3 --network=znet --ip=172.18.0.16 -v /home/devops/shard1:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

进入其中一个容器配置Shard-Server副本集

# 宿主机

docker exec -it shard1\_1 bash

# 容器中

mongo --port 27018

# Mongo Shell中

rs.initiate({

"\_id":"shard1",

"members":[

{

"\_id":0,

"host":"172.18.0.14:27018"

},

{

"\_id":1,

"host":"172.18.0.15:27018"

},

{

"\_id":2,

"host":"172.18.0.16:27018"

}

]

})

##### **10.3.2.2 启动第二个分片 - shard2**

docker run -d --name=shard2\_1 --network=znet --ip=172.18.0.17 -v /home/devops/shard2:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

docker run -d --name=shard2\_2 --network=znet --ip=172.18.0.18 -v /home/devops/shard2:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

docker run -d --name=shard2\_3 --network=znet --ip=172.18.0.19 -v /home/devops/shard2:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

进入其中一个容器配置Shard-Server副本集

# 宿主机

docker exec -it shard2\_1 bash

# 容器中

mongo --port 27018

# Mongo Shell中

rs.initiate({

"\_id":"shard2",

"members":[

{

"\_id":0,

"host":"172.18.0.17:27018"

},

{

"\_id":1,

"host":"172.18.0.18:27018"

},

{

"\_id":2,

"host":"172.18.0.19:27018"

}

]

})

##### **10.3.2.3 启动第三个分片 - shard3**

docker run -d --name=shard3\_1 --network=znet --ip=172.18.0.20 -v /home/devops/shard3:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

docker run -d --name=shard3\_2 --network=znet --ip=172.18.0.21 -v /home/devops/shard3:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

docker run -d --name=shard3\_3 --network=znet --ip=172.18.0.22 -v /home/devops/shard3:/etc/mongodb mongo:4.0.0 -f /etc/mongodb/mongod.conf

进入其中一个容器配置Shard-Server副本集

# 宿主机

docker exec -it shard3\_1 bash

# 容器中

mongo --port 27018

# Mongo Shell中

rs.initiate({

"\_id":"shard3",

"members":[

{

"\_id":0,

"host":"172.18.0.20:27018"

},

{

"\_id":1,

"host":"172.18.0.21:27018"

},

{

"\_id":2,

"host":"172.18.0.22:27018"

}

]

})

##### **10.3.2.4 启动3个mongos服务器**

* 说明：这里也使用了mongo镜像，但是需要开启mongos进程，mongod进程并不需要用到。

docker run -d --name=mongos\_1 --network=znet --ip=172.18.0.23 -v /home/devops/mongos:/etc/mongodb mongo:4.0.0

docker run -d --name=mongos\_2 --network=znet --ip=172.18.0.24 -v /home/devops/mongos:/etc/mongodb mongo:4.0.0

docker run -d --name=mongos\_3 --network=znet --ip=172.18.0.25 -v /home/devops/mongos:/etc/mongodb mongo:4.0.0

进入每个容器中，启动mongos进程

# 宿主机

docker exec -it mongos\_1 bash

# 容器中

mongos -f /etc/mongodb/mongos.conf

可以就在其中一个mongos容器中使用mongo shell连接mongos进程配置分片集群

# 连接mongos，端口号与mongos配置文件中设定一致

mongo -port 27020

# 将分片加入集群

sh.addShard("shard1/172.18.0.14:27018,172.18.0.15:27018,172.18.0.16:27018")

sh.addShard("shard2/172.18.0.17:27018,172.18.0.18:27018,172.18.0.19:27018")

sh.addShard("shard3/172.18.0.20:27018,172.18.0.21:27018,172.18.0.22:27018")

# 对数据库开启分片功能

sh.enableSharding("cms")

# 对数据库中集合开启分片，并指定片键

sh.shardCollection("cms.user",{"name":1})

### **10.4 写入数据**

use cms;

for(var i=1;i<=1000000;i++){

db.user.insert({name:i,age:i});

}

# 查看分片状态

sh.status()

## **11. redis集群**

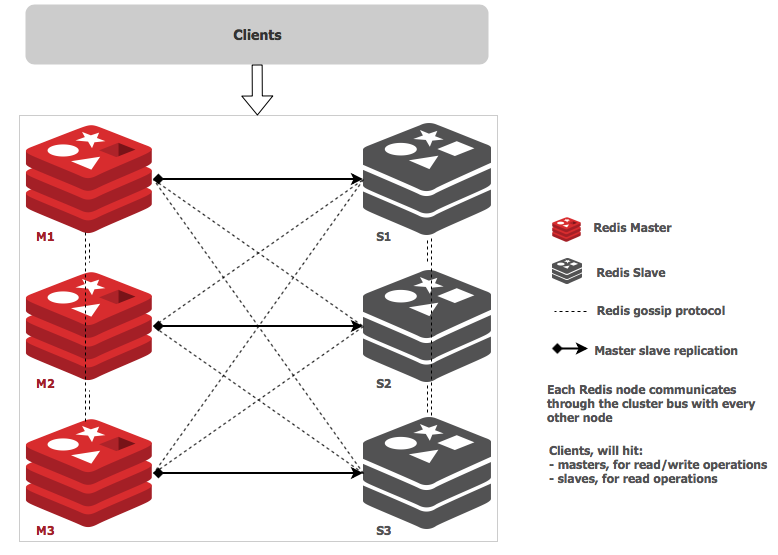
* 高速缓存利用内存保存数据，读写速度远超硬盘
* 高速缓存可以减少I/O操作，降低I/O压力
* Redis是VMware开发的开源免费的KV型NoSQL缓存产品
* Redis具有很好的性能，最多可以提供10万次/秒的读写

### **11.1 RedisCluster**

* RedisCluster是官方推荐的，没有中心节点
* 无中心节点，客户端与redis节点直连，不需要中间代理层
* 数据可以被分片存储，每个节点存储的数据是不一样的，每个节点需要提供冗余节点
* Redis管理方便，可以随时自行增加和摘除节点

### **11.2 主从同步**

* Redis集群中的数据库复制是通过主从同步来实现的
* 主节点(Master)把数据分发给从节点(Slave)
* 主从同步的好处在于高可用，Redis节点有冗余设计
* Redis集群中应该包含奇数个Master,至少应该有3个Master
* Redis集群中每个Master都应该有Slave



### **11.3 实操**

#### **11.3.1 安装Redis镜像**

* 导入本地Redis镜像文件，运行Redis容器

docker pull zhangrenyang/redis:latest

docker tag zhangrenyang/redis:latest zredis

#### **11.3.2 启动容器**

docker run -it -d --name redis1 -p 5001:6379 --net=znet --ip 172.18.0.23 zredis bash

docker run -it -d --name redis2 -p 5002:6379 --net=znet --ip 172.18.0.24 zredis bash

docker run -it -d --name redis3 -p 5003:6379 --net=znet --ip 172.18.0.25 zredis bash

docker run -it -d --name redis4 -p 5004:6379 --net=znet --ip 172.18.0.26 zredis bash

docker run -it -d --name redis5 -p 5005:6379 --net=znet --ip 172.18.0.27 zredis bash

docker run -it -d --name redis6 -p 5006:6379 --net=znet --ip 172.18.0.28 zredis bash

#### **11.3.3 配置文件**

/user/redis/redis.conf

| **参数** | **含义** |
| --- | --- |
| daemonize yes | 以后台模式运行 |
| cluster-enabled yes | 开启集群 |
| cluster-config-filter nodes.conf | 集群配置文件 |
| cluster-node-timeout 15000 | 超时时间 |
| appendonly yes | 开启AOF模式，实现日志恢复数据 |

#### **11.3.4 配置集群**

cp /usr/redis/src/redis-trib.rb /usr/redis/cluster

cd /usr/redis/cluster

apt-**get** install ruby

apt-**get** install rubygems

gem install redis

./redis-trib.rb create --replicas 1 172.18.0.23:6379 172.18.0.24:6379 172.18.0.25:6379 172.18.0.26:6379 172.18.0.27:6379 172.18.0.28:6379

--replicas 1 参数表示为每个主节点创建一个从节点

## **12.数据库备份**

### **12.1 mysql**

wget -i -c http://dev.mysql.com/get/mysql57-community-release-el7-10.noarch.rpm

yum -y install mysql57-community-release-el7-10.noarch.rpm

yum -y install mysql-community-server

systemctl start mysqld.service

systemctl status mysqld.service

mysql -uroot -p

#!/bin/bash

DATE=$(date +%F\_%H-%M-%S)

HOST=192.168.0.1

DB=test

USER=root

PASS=123456

MAIL="83687401@qq.com"

BACKUP\_DIR=/data/db\_backup

SQL\_FILE=${DB}\_FULL\_${DATE}.sql

cd $BACKUP\_DIR

mysqldump -h$HOST -u$USER -p$PASS > $SQL\_FILE

echo "$DATE 备份成功" | mail -s "备份成功通知" $MAIL

### **12.2 mongodb备份**

dump=/usr/local/mongodb/bin/mongodump

out\_dir=/media/sf\_mongobak/dump\_bak

tar\_dir=/media/sf\_mongobak/tar\_bak

mkdir -p $out\_dir/$sysdate

$dump -h 127.0.0.1 -d masterdata -o $out\_dir/$sysdate

exit

## **13.监控主机状态**

### **13.1 定义一个颜色输出字符串函数**

#!/bin/bash

#description: test

function echo\_color(){

if [ $1 == "green" ]; then

echo -e "\033[32;40m$2\033[0m"

elif [ $1 == "red" ]; then

echo -e "\033[31;40m$2\033[0m"

fi

}

function echo\_color2(){

case $1 in

green)

echo -e "\033[32;40m$2\033[0m"

;;

red)

echo -e "\033[31;40m$2\033[0m"

;;

\*)

echo "echo\_color2 {green|red} string"

esac

}

echo -e "\033[32;40mshell\033[0m"

echo -e "\033[33;40mshell\033[0m"

echo\_color green hello

echo\_color red world

### **13.2. 批量创建用户**

#!/bin/bash**function** **echo\_color**(){

**if** [ $1 == "green" ]; **then**

echo -e "\033[32;40m$2\033[0m"

**elif** [ $1 == "red" ]; **then**

echo -e "\033[31;40m$2\033[0m"

**fi**

}

**for** USER **in** user{1..5}; **do**

**if** ! id $USER &>/dev/null; **then**

PASS=$(echo $RANDOM | md5sum | cut -c 1-8)

useradd $USER

echo $PASS | passwd --stdin $USER &> /dev/null

echo -e "$USER\t$PASS" >>user\_file

echo "$USER user create successfully."

**else**

echo\_color red "$USER already exists.";

**fidone**

### **13.3. 检查主机存活状态**

#### **13.3.1 将错误IP放到数组中里面判断是否ping失败三次**

#!/bin/bash

IP\_LIST="192.168.0.1 192.168.0.2"**for** IP **in** $IP\_LIST; **do**

**if** ping -c 1 $IP &>/dev/null; then

echo "$IP is ok."

**else**

echo "$IP is wrong!"

fi

done

#!/bin/bash

IP\_LIST="192.168.0.1 192.168.0.222"

for IP in $IP\_LIST; do

NUM=1

while [ $NUM -le 3 ]; do

if ping -c 1 $IP &>/dev/null; then

echo "$IP is ok."

break

else

echo $NUM

FAIL\_COUNT[$NUM]=$IP

fi

let NUM++

done

echo ${#FAIL\_COUNT[\*]}

if [ ${#FAIL\_COUNT[\*]} -eq 3 ]; then

echo "$IP is unreachable."

fi

unset FAIL\_COUNT[\*]

done

#!/bin/bash

IP\_LIST="192.168.0.1 192.168.0.222"**for** IP **in** $IP\_LIST; **do**

FAIL\_COUNT=1

**for** ((i=1;i<=3;i++)) **do**

**if** ping -c 1 $IP &>/dev/null; then

echo "$IP is ok."

**break**

**else**

echo $NUM

**let** FAIL\_COUNT++

fi

done

**if** [ $FAIL\_COUNT -eq 3 ]; then

echo "$IP is unreachable."

fi

done

#### **13.4 获得CPU利用率**

* 借助vmstat工具来分析CPU统计信息

#!/bin/bash

cpu(){

local user system idle cwait

user=$(vmstat | awk 'NR==3{print $13}')

system=$(vmstat | awk 'NR==3{print $14}')

idea=$(vmstat | awk 'NR==3{print $15}')

cwait=$(vmstat | awk 'NR==3{print $16}')

echo "user cpu: $user%"

echo "system cpu: $system%"

echo "idle cpu: $idea%"

echo "wait: $cwait%"

}

cpu

memory(){

local total used free

used=$(free -m | awk 'NR==3{print $3}')

free=$(free -m | awak 'NR==3{print $4}')

total=$(($used+$free))

echo "内存总计: $(total)M"

echo "内存使用: $(used)M"

echo "内存剩余: $(free)M"

}

memory

disk(){

local mount total used used\_percent free

part=$(df -h|awk 'BEGIN{OFS="="}/^\/dev/{print $6,$2,$3,$4,$5}')

echo $p

**for** p **in** $part; **do**

mount=$(echo $p | cut -d"=" -f1)

total=$(echo $p | cut -d"=" -f2)

used=$(echo $p | cut -d"=" -f3)

free=$(echo $p | cut -d"=" -f4)

used\_percent=$(echo $p | cut -d"=" -f5|cut -d"%" -f1)

**if** [ $used\_percent -ge 5 ]; then

echo "挂载点: $mount"

echo "总大小: $total"

echo "使用大小: $used"

echo "空闲大小: $free"

echo "使用百分比: $used\_percent%"

fi

done

}

**while** true; **do** bash system.sh; sleep 1s;done

#### **13.5 监控网络流量**

traffic(){

local old\_in old\_out new\_in new\_out

old\_in=$(ifconfig eth0 | awk '/RX/&&/bytes/{print $2}' |cut -d":" -f2)

old\_out=$(ifconfig eth0 | awk '/TX/&&/bytes/{print $2}' |cut -d":" -f2)

sleep 1s

new\_in=$(ifconfig eth0 | awk '/RX/&&/bytes/{print $2}' |cut -d":" -f2)

new\_out=$(ifconfig eth0 | awk '/TX/&&/bytes/{print $2}' |cut -d":" -f2)

**in**=$(($new\_in-$old\_in))

out=$(($new\_out-$old\_out))

echo "${in}B/s ${out}B/s"

}

#### **13.6 监控网站状态**

curl -o /dev/null -s -w "%{http\_code}" http://www.baidu.com

**function** **check\_url**(){

HTTP\_CODE=$(curl -o /dev/null -s -w "%{http\_code}" $1)

**if** [ $HTTP\_CODE -ne 200 ]; then

echo "$1不可达"

**else**

echo "$1状态正常"

fi

}

URL\_LIST="http://www.baidu.com http://www.baidu2222.com"

**for** URL **in** $URL\_LIST; **do**

check\_url $URL

done

#### **13.7 监控nginx状态**

#!/bin/bash

Web=`ps -ef |grep nginx|grep -v grep|wc -l`

**if** [ $Web -eq 2 ];then

echo "your nginx is running"

exit 0

**else**

service nginx start

exit 1

fi

#### **13.8 监控mysql状态**

#!/bin/sh

PortNum=`netstat -lnt|grep 3306|wc -l`**if** [ $PortNum -eq 1 ]

then

echo "mysqld is running."**else**

echo "mysqld is stoped."

fi

## **12.参考**

### **12. 网址**

* [linux](https://www.linux.org/)
* [docker](https://www.docker.com/)
* [nginx](http://nginx.org/)
* [mysql](https://www.mysql.com/)
* [mongodb](https://www.mongodb.com/)
* [xshell](https://xshell.en.softonic.com/)
* [xftp](https://www.netsarang.com/zh/xftp/)
* [robomongo](https://robomongo.org/)
* [navicat](https://www.navicat.com.cn/products/)
* [redis](https://redis.io/)

### **12.配置node.js**

#### **12.1 安装nvm**

* [nvm](https://github.com/nvm-sh/nvm)

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash

source /root/.bashrc

nvm install stable

npm i cnpm -g

cnpm i nrm -g

cnpm i pm2 -g