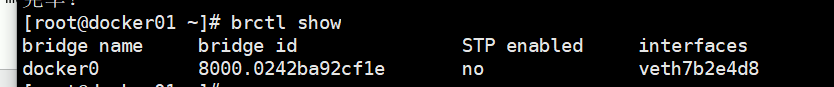
Docker 网络

Docker容器之间默认使用桥接网络



yum install bridge-utils



交换机上的网卡，就是nginx在docker0上的网卡

查看nginx的网络

docker inspect b20c602574de

因为docker0也是物理机上的一张网卡，所以物理机也可以通过docker0访问nginx容器ip地址

curl 172.17.0.2

由于docker容器之间默认使用桥接网络，所以两台物理机之间的docker容器是不能直接通信的，想要通信可以通过1、端口转发，将容器端口映射到物理机端口。

2、还可以使用overlay网络实现通信

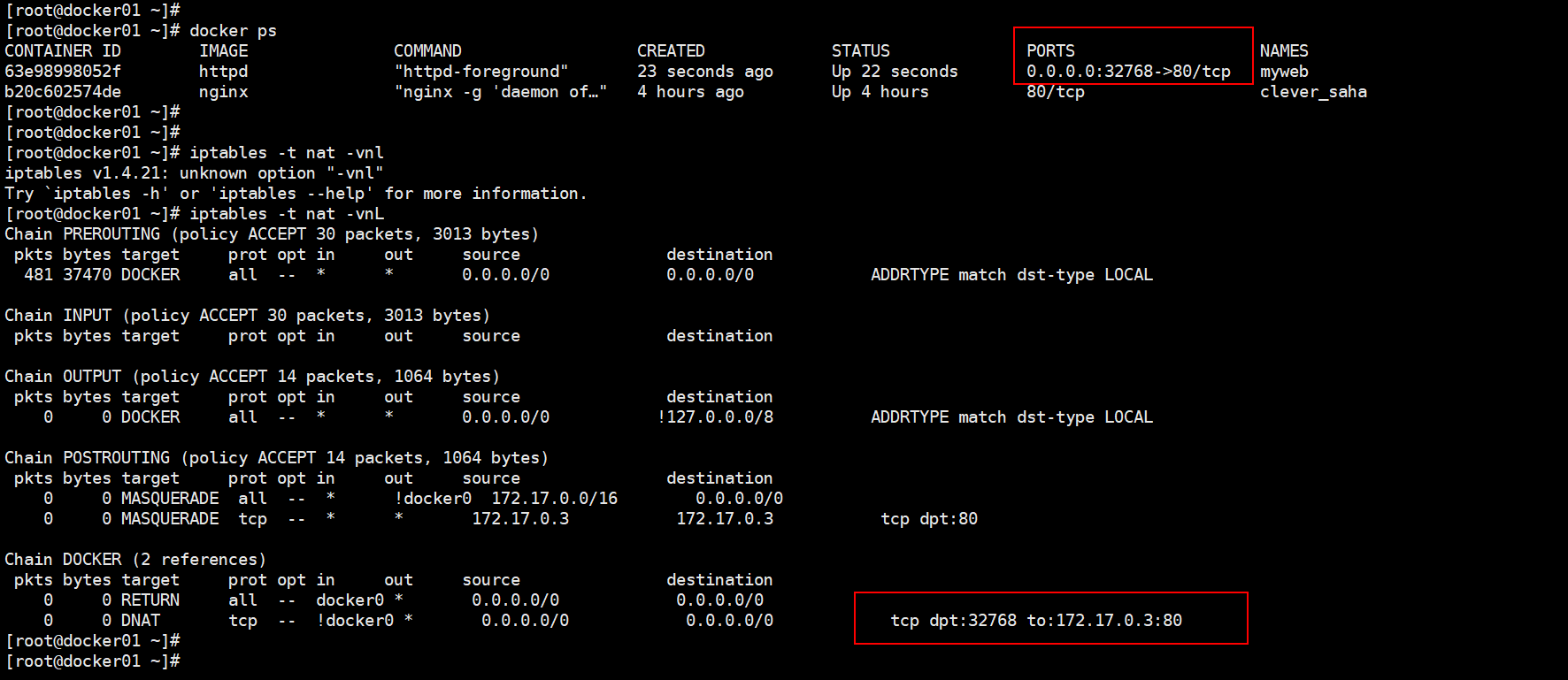
Host网络表示容器使用物理机的网络命名空间

None网络表示没有网络，用于运行一些不需要网络的任务，比如说批处理数据



通过iptables转发建立本地端口和容器的映射关系

docker run --name myweb --rm -p 80 httpd



同一物理机有多个docker network（交换机），network之间的的容器可以通过开启物理机的网络转发、修改iptables规则实现通信。当然，既然已经将容器放到不同的交换机下，那么就意味着这容器就不需要通信了

存储卷 volume

docker run --name b2 -it -v /data busybox

容器中有/data目录

docker container inspect b2

"Source": "/var/lib/docker/volumes/70836ea4485b4e302f6876d43d99783a362ac27f3e11e170772e535cfcd4bba7/\_data",

"Destination": "/data",

宿主机的Source目录和容器的Destination目录建立了映射关系

修改Source、Destination会相互影响

指定本机/data/volumes/b2目录映射到容器/data busybox

docker run --name b2 -it --rm -v /data/volumes/b2:/data busybox

docker run --name infracon -it -v /data/infracon/volume/:/data/web/html busybox

nginx（busybox）使用container:infracon的网络，存储卷跟infracon相同

docker run --name nginx --network container:infracon --volumes-from infracon -it busybox

Dockerfile

Vim Dockerfile

FROM nginx:1.14-alpine

LABEL maintainer="MageEdu"

ENV NGX\_DOC\_ROOT='/data/web/html/'

ADD index.html ${NGX\_DOC\_ROOT}

ADD entrypoint.sh /bin/

CMD ["/usr/sbin/nginx","-g","daemon off;"]

ENTRYPOINT ["/bin/entrypoint.sh"]

Vim entrypoint.sh

#!/bin/sh

cat > /etc/nginx/conf.d/www.conf << EOF

server {

server\_name ${HOSTNAME};

listen ${IP:-0.0.0.0}:${PORT:-80};

root ${NGX\_DOC\_ROOT:-/usr/share/nginx/html};

}

EOF

exec "$@"

exec表示在同一个shell中执行，$@表示所有参数

exec "$@" 执行完entrypoint.sh这个shell后，在同一个shell（shell进程id是1）中执行CMD中的命令（则nginx运行的进程号也是1）

chmod +x entrypoint.sh

vim index.html

<p>hello dockerfile</p>

构建

docker build -t test01 .

运行

docker run --name test01 --rm -P test01

传入PORT环境变量修改端口

docker run --name test01 --rm -P -e "PORT=8080" test01

端口改成了nginx的另一个端口改成了8080

Build过程参数设置

ARG author="默认作者"

LABEL maintainer=${author}

docker build --build-arg author="自定义作者" -t test02 .

docker image inspect test02

docker run –e 是在运行时传变量，这些变量定义在shell语句中

安装harbor

安装好docker

虚拟机配置cpu > =2，Mem>=4G，Disk>=40G

cpu > =2，Mem>=4.5G，Disk>=45G

修改virtualbox磁盘大小

//查看虚拟磁盘空间信息，确认确实是需要扩容的虚拟硬盘。

VBoxManage showhdinfo "E:\VirtualBox VMs\harbor01\harbor01.vdi"

//扩展虚拟硬盘空间，最后的数字替换成上面计算出的硬盘空间大小,1024\*1024\*1024\*50。

VBoxManage modifymedium "E:\VirtualBox VMs\harbor01\harbor01.vdi" --resizebyte 53687091200

<https://www.cnblogs.com/zhangtianyuan/p/6908659.html>

安装docker-compose

curl -L https://github.com/docker/compose/releases/download/1.19.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

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

安装harbor

下载 https://storage.googleapis.com/harbor-releases/release-1.4.0/harbor-offline-installer-v1.4.0.tgz

tar xf harbor-offline-installer-v1.4.0.tgz -C /usr/local/

cd /usr/local/harbor

vim harbor.cfg

hostname = 192.168.4.179

max\_job\_workers = 2

./install.sh

登陆

<http://192.168.4.179/harbor/sign-in>

admin Harbor12345

#新建项目

项目 — 新建项目 — 名称、公开

#配置docker insecure-registries

vim /etc/docker/daemon.json

{

"registry-mirrors": ["https://ojmdil2j.mirror.aliyuncs.com"],

"insecure-registries": ["192.168.4.179"]

}

systemctl daemon-reload

systemctl restart docker

docker login 192.168.4.179

admin

Harbor12345

#推送镜像

docker tag busybox:latest 192.168.4.179/p2/busybox:v1.0

docker tag busybox:latest 192.168.4.179/p2/busybox:v2.0

docker push 192.168.4.179/p2/busybox:v1.0

docker push 192.168.4.179/p2/busybox:v2.0

#在docker-compose.yml文件目录下，停止、启动harbor

cd /usr/local/harbor

docker-compose start/stop

docker资源限制，改下虚拟机的cpu个数为2，cpu使用率200%表示使用了2个cpu，每个cpu使用率为100%

#docker压测工具

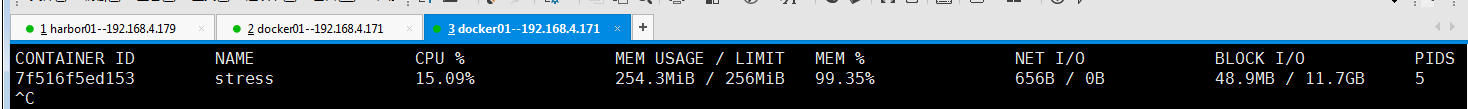
docker pull lorel/docker-stress-ng:latest

# 限制容器使用256m内存，但是压测容器需要用到2\*256m内存

docker run --name stress -it --rm -m 256m lorel/docker-stress-ng:latest stress --vm 2

#查看资源使用情况

docker stats stress



#限制使用一个cpq，但是起了8个进程

docker run --name stress -it --rm --cpus 1 lorel/docker-stress-ng:latest stress --cpu 8

docker top stress

docker stats stress

#限制使用一个cpq的50%，但是起了8个进程

docker run --name stress -it --rm --cpus 0.5 lorel/docker-stress-ng:latest stress --cpu 8