# Docker

docker岗位：Go

Swarm/Compose/Machine/mesos/kubernetes(k8s)/k8s/----CI/CD jenkinds整合...

|  |  |
| --- | --- |
| docker官网 | https://www.docker.com |
| docker官网-中国 | http://docker-cn.com/ |
| docker下载地址 | get.daocloud.io |
| docker国内镜像(加速器) | https://www.daocloud.io/ |
| docker官方模拟器 | https://www.docker.com/tryit/ |
| docker官网资料 | https://docs.docker.com |
| docker中国社区 | http://dockboard.org/ |
| docker@github | http://github.com/docker/docker/issues |

* 术语：

|  |  |
| --- | --- |
| host | 宿主机 |
| image | 镜像 |
| container | 容器 |
| registry | 仓库 |
| daemon | 守护进程 |
| client | 客户端 |

# 操作

1. docker images //查看本地镜像
2. docker run ubuntu echo hello docker //启动ubuntu，并运行echo hello docker
3. docker run -d -p 9200:9200 elasticsearch //获取并运行elasticsearch
4. docker run -d -p 8080:80 nginx //-d 后台运行不退出 -p 80:80 映射端口
5. docker run -it alpine sh // -it 参数：直接进入到Linux系统里去
6. docker ps //查看正在运行的docker container
7. docker ps -a //查看运行过的容器的残留文件
8. docker cp index.html containerId://containerAddress[usr/share/nginx/html]
9. docker stop containerId //停止此容器
10. docker kill CONTAINER\_ID //杀死此ID的程序
11. docker commit -m ‘fun’ containerID imageName //创建一个新镜像
12. docker logs CONTAINER\_ID //查看container运行日志
13. docker rmi containerID //删除不用的image
14. docker rm containerID //删除运行过的残留文件
15. curl <http://localhost:8080> //测试访问此地址

* 注意：

1. docker在container中做出的改动都是暂时的，没有被保存下来，下次启动又会回到默认值
2. 使用commit创建一个新的image，保存改动的image

## 命令总结：

|  |  |
| --- | --- |
| docker pull | 获取image |
| docker build | 创建image |
| docker images | 列出image |
| docker run | 运行container |
| docker ps | 列出正在运行的container |
| docker rm | 删除container残留 |
| docker rmi | 删除image |
| docker cp | 在host和container之间cp文件 |
| docker commit | 保存改动为新的image |
| docker search | 查找镜像 |

# 自创docker镜像

## 通过Dockerfile构建image

约定文件名为Dockerfile

Dockerfile的指令分为两种：注释(#开头)和指令

create file：$ vi Dockerfile

build image：$ docker build -t (imageTag) (.)[此路径下的所有文件]

run：$ docker run (imageTag)

* 第一个Dockerfile：

FROM alpine:latest //基础对象；alpine是专门针对docker做的image

MAINTAINER xbf //共享的时候，告诉其他人，是我写的

CMD echo ‘hello docker’ //实际的命令

* 第二个Dockerfile

FROM ubuntu

MAINTAINER xbf

##使用国内的镜像，用来加速##

RUN sed -i ‘s/archive.ubuntu.com/mirrors.ustc.edu.cn/g’ /etc/apt/sources.list

RUN apt-get update

RUN apt-get install -y nginx

COPY index.html /var/www/html

ENTRYPOINT[“/usr/sbin/nginx”,”-g”,”daemon off;”]

EXPOSE 80

## Dockerfile命令

|  |  |
| --- | --- |
| FROM | base image |
| EXPOSE | 暴露端口 |
| MAINTAINER | 维护者 |

RUN：镜像构建时执行该命令

CMD：容器运行时执行的默认命令，会被docker run指定的启动命令覆盖

ENTERYPOINT：不会被docker run命令中指定的启动命令覆盖；如果想覆盖，需要显式的在docker run中指定ENTERYPOINT指令

CMD/ENTERYPOINT：组合使用，CMD指定默认参数，ENTERYPOINT指定命令

-----------------------------------------------------------------------------------------------------------------

ADD/COPY src desc → src为相对路径，desc为绝对路径

ADD：自动tar解压功能；可以从URL上下载；

COPY：如果只是copy文件，推荐使用此命令

VOLUME：添加卷

----------------------------------------------------------------------------------------------------------------

WORKDIR：从image创建container时设置工作目录，CMD/ENTERYPOINT都会在该目录下执行；在构建中为后续的指令指定工作目录；通常使用绝对路径，如果使用相对路径，结果会向下传递

ENV：设置环境变量；可以作用于构建中，运行中依然有效

----------------------------------------------------------------------------------------------------------------

USER：指定该镜像以什么样的用户身份运行；也可以在此命令中使用用户组(uid，gid)，如果不指定，则默认使用root用户；

USER user USER user:group USER user:gid

USER uid USER uid:group USER uid:gid

-----------------------------------------------------------------------------------------------------------------

ONBUILD：为镜像添加触发器；当一个镜像被其他镜像作为基础镜像时执行；会在构建过程中插入指令；

ONBUILD COPY index.html /usr/share/nginx/html/

## Docker镜像的构建过程

* 从基础镜像运行一个容器
* 执行一条指令，对容器进行修改
* 执行类似docker commit的操作，提交一个新的镜像层
* 再基于刚提交的镜像层运行一个新容器
* 执行Dockerfile中的下一条指令，直至所有指令执行完毕

在构建过程中，中间层的容器会被删除，但是中间层的镜像会保留下来，可以通过中间层的镜像来调试；

* docker构建缓存；再次构建时，直接使用缓存，使构建过程很高效
* 不想使用构建缓存时：docker build --no-cache 可以不使用缓存

$ docker history [image] //可以查看镜像的构建过程

## 镜像分层

Docker的image是分层存储的

Dockerfile中的每一行都产生一个新层

Docker的image是只读文件，Docker run后的container会附件一个虚拟层，在此层可以读写，但是原层都是只读的。要想保存修改，只能重新commit一个新的镜像。Docker的操作思想：写时复制

## Volume

* 提供独立于容器之外的持久化存储，比如数据库操作
* 提供容器之间的共享数据
* 容器本身不能持久化文件，关闭容器后，数据销毁
* 操作1：

$ docker run -d --name nginx -v /usr/share/nginx/html nginx

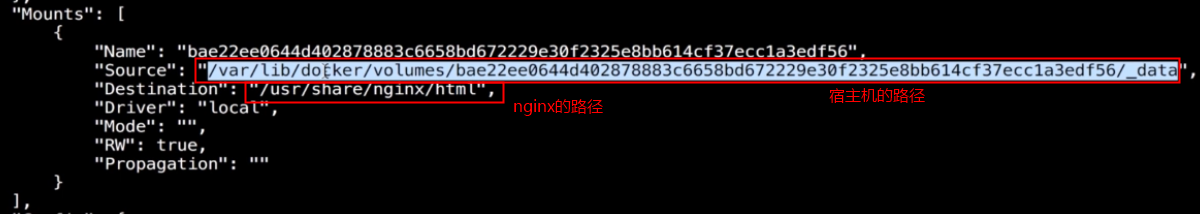
→ -v 挂载一个卷到容器中，/usr/share/nginx/html为容器nginx的资源访问路径

→ 是把宿主机的此路径挂载到nginx中路径中

→ -d 为daemon，即守护进程/后台程序

$ docker inspect nginx //检查，一些nginx的信息

$ docker exec -it nginx /bin/bash //进到容器中去看容器中的文件



* 操作2：

$ docker run -v $PWD/code:/var/www/html nginx

→$PWD是sheel的一个环境变量，指向当前目录；-v 就是volume(卷)的意思

→把当前code目录挂载到nginx容器中的var/www/html上

→可以把外部文件挂载到容器中，修改的时候直接修改外部文件即可，不用到容器中取修改

* 操作3：

创建一个只存储数据的container，可以把这个container的数据加载到其他容器中

$ docker create -v $PWD/data:/var/mydata --name data\_container ubuntu

→$PWD/data 表示当前目录下的data目录

→/var/mydata 为容器中的目录；--name data\_container 给新建的容器命名

→ubuntu 为新建的容器给定的基础对象

$ docker run -it --volumes-from data\_container ubuntu /bin/bash

→ -it 以交互的方式运行，进入到ubuntu中

→ --volumes-from data\_container 把此容器加载到ubuntu中

# mount //可以在此信息中找到挂载的文件位置

# touch file → # exit //在本地挂载的文件中同步生成修改内容，挂载成功

## 镜像仓库Registry

### 命令

$ docker search whalesay //查找仓库中的镜像

$ docker pull whalesay //把镜像拖下来

$ docker push myname/whalesay //上传自己的镜像到仓库

### 仓库

daocloud: http://hub.daocloud.io/

aliyun:https://opsx.alibaba.com/mirror

https://dev.aliyun.com/search.html

时速云

网易云：c.163.com；[http://mirrors.163.com/](http://mirrors.163.com/" \t "https://blog.csdn.net/tfdg126/article/details/_blank)1111

docker默认仓库：https://hub.docker.com

运行whalesay镜像

$ docker run docker/whalesay cowsay Dockerisfun

* 上传镜像：

$ docker tag docker/whalesay xbf/whalesay

→先命名要上传的镜像

$ docker images

$ docker push xbf/whalesay //上传镜像

上传之前需要先登录

$ docker login //登录仓库，默认为dockerhub仓库

## 多容器APP(docker-compose)

### docker-compose安装

* Mac/Windows: 自带
* Linux安装：

$ curl -L <https://github.com/docker/compose/releases/download/1.9.0/docker-compose-$(uname> -s)-$(uname -m) > /usr/local/bin/docker-compose

→ -$(uname -s)-$(uname -m) 表示把 uname -s 和 uname -m 的输出写到此路径中；再把结果通过 > 写入到文件中

$ chmod a+x /usr/local/bin/docker-compose //把此文件设置为所有用户可执行文件

$ docker-compose --version //查看版本，以确认是否安装成功

### 使用

user → nginx → ghost app → mysql

yaml是配置文件，以缩进来表示层级关系

docker-compose.yaml

ghost-app:

build:ghost

depends\_on:

-db

ports:

-”2368:2368”

nginx:

build:nginx

ports:

-“80:80”

depends\_on:

-ghost-app

db:

image:”mysql:5.7.15”

* 操作：

mkdir ghost → cd ghost → mkdir ghost →mkdir nginx → mkdir data

cd ghost → touch Dockerfile → vi Dockerfile

cd .. → cd nginx → touch Dockerfile → vi Dockerfile

touch nginx.conf → vi nginx.conf

cd .. → touch docker-compose.yml → vi docker-compose.yml

* ghost/Dockerfile

FROM ghost

COPY ./config.js /var/lib/ghost/config.js

EXPOSE 2368

CMD [“npm”,”start”,”--production”]

* ghost/config.js

var path = require(‘path’),

config;

config = {

production: {

url: ‘http://mytestblog.com’,

mail: {},

database: {

client : ‘mysql’,

connection : {

host : ’db’,

user : ’ghost’,

password : ’ghost’,

database : ’ghost’,

port : ’3306’,

charset : ‘utf8’

}

debug : false

},

paths: {

contentPath: path.join(pocess.env.GHOST\_CONTENT, ‘/’)

},

server: {

host: ’0.0.0.0’,

port: ‘2368’

}

}

};

module.exports = config;

* nginx/Dockerfile

FROM nginx

COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

* nginx/nginx.conf

worker\_processes 4;

events {worker\_connections 1024;}

http {

server {

listen 80;

location / {

proxy\_pass <http://ghost-app:2368;>

}

}

}

* docker-compose.yml

version:’2’

networks:

ghost:

services:

ghost-app:

build: ghost

networks:

- ghost

depends\_on:

- db

ports:

- “2368:2368”

nginx:

build: nginx

networks:

- ghost

depends\_on:

- ghost-app

ports:

- “80:80”

db:

image: “mysql:5.7.15”

networks:

- ghost

environment:

MYSQL\_ROOT\_PASSWORD: mysqlroot

MYSQL\_USER: ghost

MYSQL\_PASSWORD: ghost

volumes:

- $PWD/data:/var/lib/mysql

ports:

- “3306:3306”

### 构建容器

$ docker-compose up -d //构建容器

//如果第一次构建错误，需要如下操作

$ docker-compose stop //停止已运行的compose-app

$ docker-compose rm //删除已停止的compose-app

$ docker-compose build //重新构建

$ docker-compose up -d //重新构建容器

项目访问路径：localhost/ghost/setup/

### docker-compose.yml常用命令

|  |  |
| --- | --- |
| build | 本地创建镜像 |
| command | 覆盖缺省命令 |
| depends\_on | 连接容器 |
| ports | 暴露端口 |
| volumes | 卷 |
| image | pull镜像 |

### docker-compose命令

|  |  |
| --- | --- |
| up | 启动服务 |
| stop | 停止服务 |
| rm | 删除服务中的各个容器 |
| logs | 观察各个容器的日志 |
| ps | 列出服务相关的容器 |

# Docker容器网络实现

## Docker容器网络基础

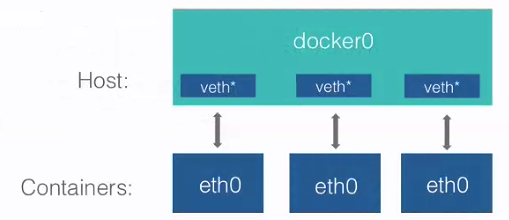
***Docker0：***相当于一个虚拟网桥

Linux虚拟网桥的特点：

可以设置IP地址；

相当于拥有一个隐藏的虚拟网卡；

运行docker image时，相当于创建了网络连接的两端，即container中的eth0和host中的veth



* 自定义docker0

修改docker0地址：

$ sudo ifconfig docker0 192.168.200.1 netmask 255.255.255.0

* ***自定义虚拟网桥***：

添加虚拟网桥：

$ sudo brctl addbr br0

$ sudo ifconfig br0 192.168.100.1 netmask 255.255.255.0

更改docker守护进程的启动配置：

/etc/default/docker中添加DOCKER\_OPS值 [-b=br0]

## Docker容器的互联

### 允许所有容器互联

--icc=true //默认

环境准备：

用于测试的docker镜像的dockerfile文件

FROM ubuntu

RUN apt-get install -y ping

RUN apt-get update

RUN apt-get install -y nginx

RUN apt-get install -y curl

EXPOSE 80

CMD /bin/bash

容器的ip地址是不可靠的连接，会随着容器的重新启动而改变；

如果在一个容器使用ip来连接了另一个容器，当容器重启的时候，该连接失效；

解决办法：

--link

$ docker run --link=[container\_name]:[alias] [image] [commond]

-----------------------------------------------------------------------------------------------------------------

$ docker run -it --name cct3 --link cct1:webtest image\_name

docker会在容器启动的时候自动把cct1的ip映射到webtest上，在cct3中要连接cct1时，直接使用webtest即可

### 拒绝容器间的互联

修改docker配置文件，添加--icc=false

$ sudo vi /etc/default/docker

添加：DOCKER\_OPS值 [--icc=false]

考虑到安全性，阻断容器之间的连接；

### 允许特定容器间的连接

Docker守护进程的启动项配置：

--icc=false

--iptables=true

启动时设置：

--link

说明：

阻断所有容器之间的访问，只使用link设置的容器访问

## Docker容器与外部网络的连接

### ip\_forward

linux中的配置项，此项设置是否允许流量转发

docker中也有ip-forward配置项：

--ip-forward=true //默认

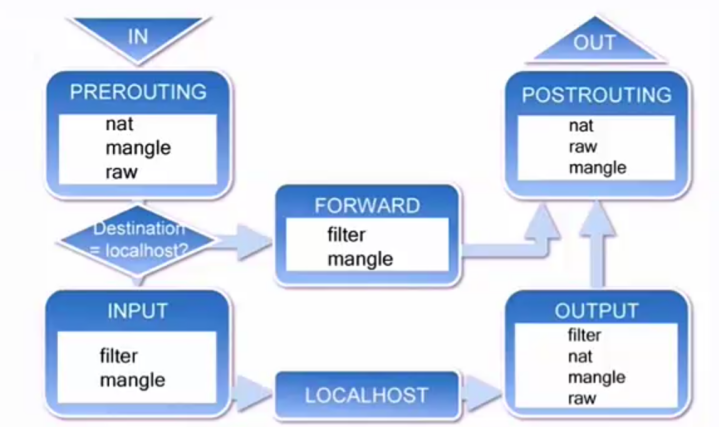
在docker启动的时候会自动设置linux中的ip\_forward为1，即允许流量转发

查看linux中的ip\_forward设置：

$ sysctl net.ipv4.conf.all.forwarding

### iptables

Iptables是Linux内核集成的包过滤防火墙系统，几乎所有的linux发行版都会包含Iptables的功能。



Iptables包含：

表（table）

链（chain）

规则（rule）

ACCEPT/REJECT/DROP

filter表中包含的链：

INPUT FORWARD OUTPUT

查看filter表：

$ sudo iptables -t filter -L -n

$ sudo iptables -L -n //默认就是filter表

限制ip访问容器

可以限制特定的端口访问特定的容器：

$ sudo iptables -I DOCKER -s 10.211.55.3 -d 172.17.0.7 -p TCP --dport 80 -j DROP

## Docker客户端与守护进程

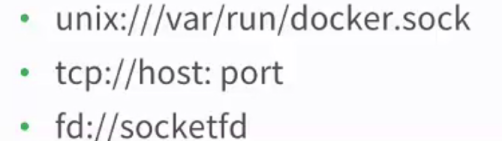
**### 此内容为2015年的，要验证一下**

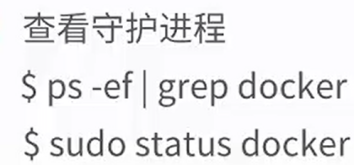
### RemoteAPI

Docker官方的Remote Api Reference

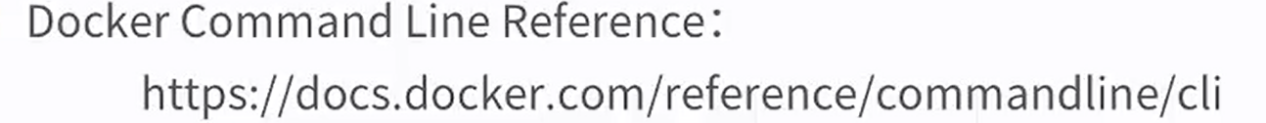
https://docs.docker.com/reference/api/docker\_remote\_api/

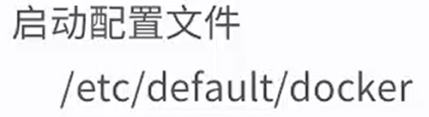
连接方式：







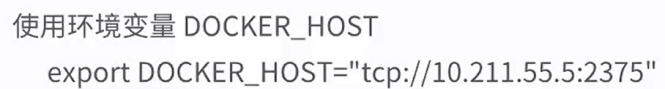




### cli远程连接







//连接完成后，断开连接

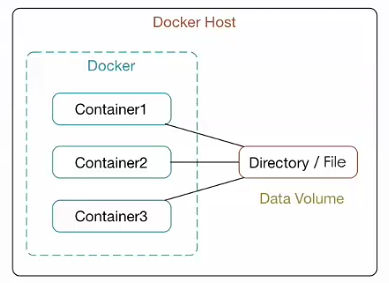
docker的-h选项可以指定多次

# Docker数据

## Docker容器的数据卷

数据卷：是经过特殊设计的目录，可以绕过联合文件系统(UFS)，为一个或多个容器提供访问。

数据卷设计的目的：数据的永久化，它完全独立与容器的生存周期，因此，Docker不会在容器删除时删除其挂载的数据卷，也不会存在类似的垃圾收集机制，对容器引用的数据卷进行处理。



数据卷(Data Volume)的特点：

* 数据卷在容器启动时初始化，如果容器使用的镜像在挂载点包含了数据，这些数据会copy到新初始化的数据卷中。
* 数据卷可以在容器之间共享和重用
* 可以对数据卷里的内容直接进行修改
* 数据卷的变化不会影响镜像的更新
* 卷会一直存在，即使挂载数据卷的容器已经被删除

为容器添加数据卷：

$ sudo docker run -v ~/container\_data:/data -it ubuntu /bin/bash

为数据卷添加访问权限

$ sudo docker run -v ~/datavolume:/data:ro -it ubuntu /bin/bash

使用Dockerfile构建包含数据卷的镜像

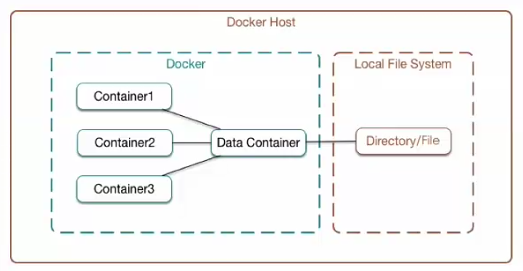
Dockerfile指令：

VOLUME[“/data”]

这样初始化的数据卷目录只能是新创建的，不能是已经存在的目录；会自动在container和host中创建相互对应的数据卷目录，这样创建的数据卷不能共享数据

## Docker的数据卷容器

命名的容器挂载数据卷，其他容器通过挂载这个容器实现数据共享，挂载数据卷的容器，就叫做数据卷容器



挂载数据卷容器的方法：

$ docker run --volumes-from [container\_name]

删除了数据卷容器，挂载了此容器的container依然可以访问挂载的目录；数据卷容器只是配置信息的传递。

$ docker rm -v dvt8 //-v注明删除的时候卷也删除

但是这样还可以使用挂载的数据卷；docker规定，正在使用的数据卷会一直存在

## Docker数据卷的备份和还原

数据备份方法：

$ docker run --volumes-from [container\_name] -v $(pwd):/backup ubuntu

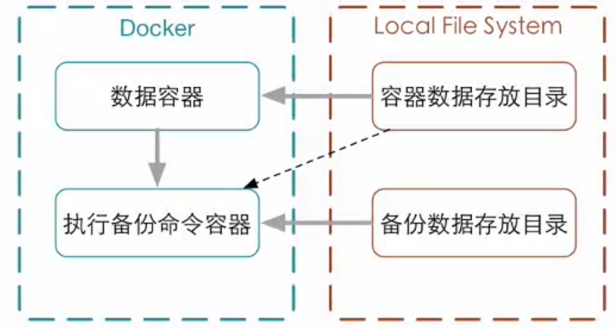
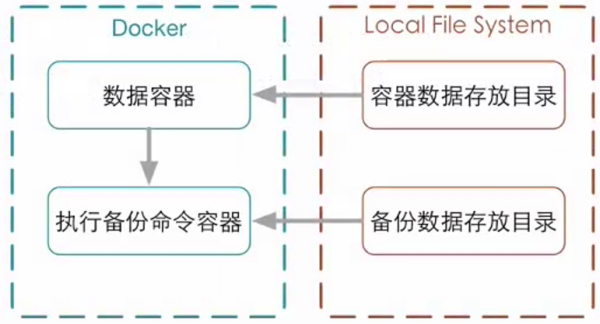
tar cvf /backup/backup.tar [container data volume]

$ docker run --volumes-from dvt5 -v ~/backup:/backup --name dvt10 ubuntu tar cvf /backup/dvt.tar /datavolume1

数据还原方法：

$ docker run --volumes-from [container name] -v $(pwd):/backup ubuntu

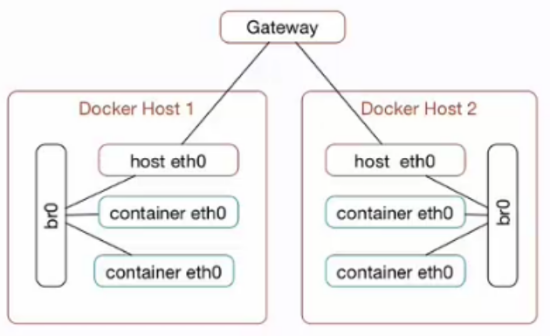
tar xvf /backup/backup.tar [container data volume]



# Docker容器的跨主机连接

## 使用网桥实现跨主机容器连接

原理：网络拓扑



环境准备：



网络设置：

修改/etc/network/interfaces文件

auto br0

iface br0 inet static

address 10.211.55.3

netmask 255.255.255.0

gateway 10.211.55.1

bridge\_ports eth0

Docker设置

修改/etc/default/docker文件

-b指定使用自定义网桥

-b=br0

--fixed-cidr限制ip地址分配范围

IP地址划分：

Host1:10.211.55.64/26

地址范围：10.211.55.65~10.211.55.126

Host2:10.211.55.128/26

地址范围：10.211.55.129~10.211.55.190

总结：

优点：

配置简单，不依赖第三方软件

缺点：

与主机在同网段，需要小心划分IP地址

需要有网段控制权，在生产环境中不易实现

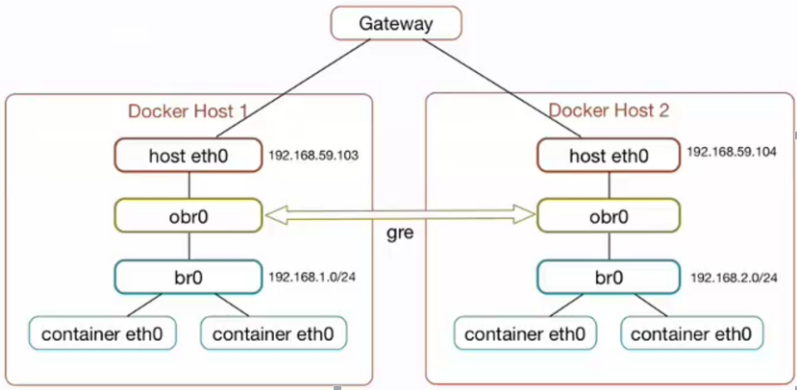
不容易管理

兼容性不佳

## 使用Open vSwitch实现跨主机容器连接

Open vSwitch是一个高质量的、多层虚拟交换机，使用开源Apache2.0许可协议，由Nicira Networks开发，主要实现代码为可移植的c代码。它的目的是让大规模网络自动化可以通过编程扩展，同时仍然支持标准的管理接口和协议（例如：NetFlow,sFlow,SPAN,RSPAN,CLI,LACP,802.lag）

原理图解：



GRE（通用路由协议封装）隧道:

隧道技术（Tunneling）是一种通过使用互联网络的基础设施在网络之间传递数据的方式。使用隧道传递的数据（或负载）可以是不同协议的数据帧或包。隧道协议将其他协议的数据帧或包重新封装然后通过隧道发送。新的帧头提供路由信息，以便通过互联网传递被封装的负载数据。

环境准备：



操作步骤：

* 建立ovs网桥
* 添加gre连接
* 配置docker容器虚拟网桥
* 为虚拟网桥添加ovs接口
* 添加不同docker容器网段路由

建立ovs网桥，添加gre连接

$ sudo ovs-vsctl show //查看ovs的版本

$ sudo ovs-vsctl add-br obr0

$ sudo ovs-vsctl add-port obr0 gre0

$ sudo ovs-vsctl set interface gre0 type=gre options:remote\_ip=192.168.59.104

$ sudo ovs-vsctl show

设置虚拟网桥并连接到gre上

$ sudo brctl addbr br0

$ sudo ifconfig br0 192.168.1.1 netmask 255.255.255.0

$ sudo brctl addif br0 obr0

$ suod brctl show

$ route //查看路由表

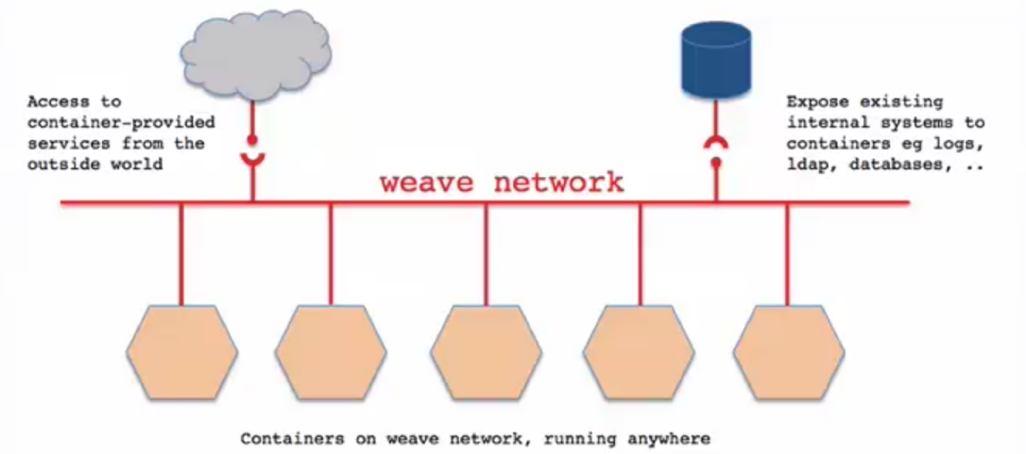
$ sudo ip route add 192.168.2.0/24 via 192.168.59.104 dev eth0 //添加路由表

## 使用weave实现跨主机容器连接

建立一个虚拟的网络，用于将运行在不同主机的Docker容器连接起来

<http://weave.works>

<https://github.com/weaveworks/weave#readme>



环境准备：



操作步骤：

* 安装Weave
* 启动Weave $weave launch
* 连接不同主机
* 通过weave启动容器

DockerHost1：

下载、安装weave：

$ sudo wget -o /usr/bin/weave https://raw.githubusercontent.com/zettio/weave/master/weave

$ sudo chmod a+x /usr/bin/weave

启动weave：

$ weave launch

通过weave启动容器：

$ weave run 192.168.1.10/24 -it --name wc1 ubuntu /bin/bash

$ docker attach wc1

$ ping 另一台主机的container\_addr

DockerHost2:

下载安装同host1；

启动weave：

$ weave launch 192.168.59.103

启动容器：

$ c2=$(weave run 192.168.1.2/24 -it ubuntu /bin/bash)

$ echo $c2 //查看c2的值

$ docker attch $c2

# Docker实战

http://www.putty.ws/PuTTY-LinuxVPS

Git: <http://git.oschina.net>

search：docker-training

## 项目1

* 构建contos镜像作为基础镜像

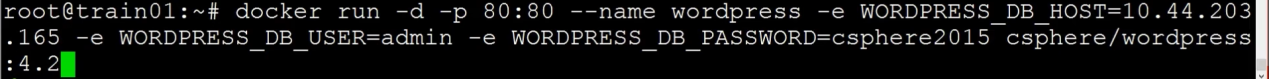
# docker build -t centos:7.1 .

* 构建php-fpm镜像作为中间件镜像

# docker build -t php-fpm:5.4 .

* 构建mysql镜像

# docker build -t mysql:5.5 .



# 附录：安装docker

**win10家庭版安装docker总结**

1.下载docker toolbox镜像，并安装(一直next)

[http://mirrors.aliyun.com/docker-toolbox/windows/docker-toolbox/](http://mirrors.aliyun.com/docker-toolbox/windows/docker-toolbox/" \t "https://www.cnblogs.com/chongyao/p/_blank)

2.双击Docker Quickstart Terminal图标运行

3.如果以前安装了git需要手动选择git/bin/bash.exe

在右击→属性→目标中设置如下

"C:\Program Files\Git\bin\bash.exe" --login -i "C:\Program Files\Docker Toolbox\start.sh"

4.下载boot2docker.iso插件

https://github.com/boot2docker/boot2docker/releases/download/v18.05.0-ce/boot2docker.iso

如果运行报错，需要手动下载boot2docker.iso放到默认路径中，再运行

注：

cup虚拟化要打开

任务管理器→性能→cup中查看

**ubuntu中安装docker**

$ sudo apt-get update

$ sudo apt install docker.io

$ sudo usermod -aG docker your-user

# 附录：docker tools外部不能访问内部端口

博文：<https://blog.csdn.net/xbinworld/article/details/78945879>

如果直接在Linux主机上安装docker的，就像一般的用法

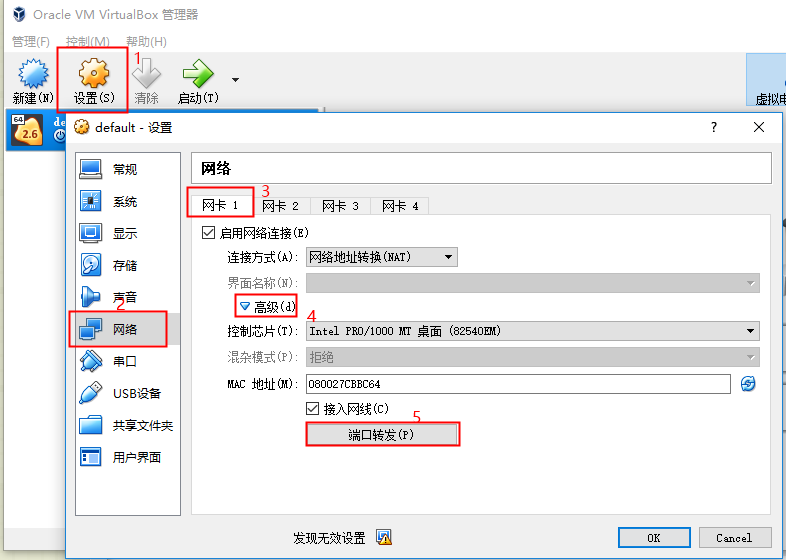
在Win10上运行了虚拟机，再在虚拟机上运行了dokcer，这样就找不到服务了

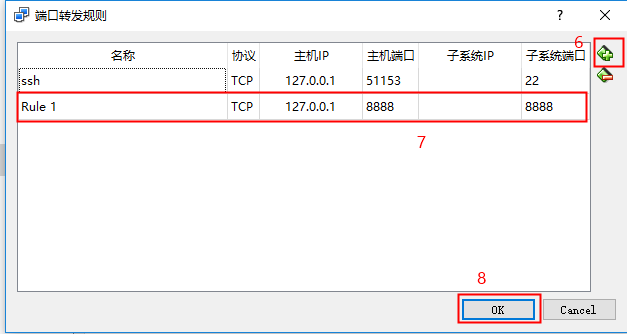
解决方案：

1.stop虚拟机的default

docker-machine stop default

2.去VitualBox中设置网络





3.开启虚拟机

docker-machine start default //开启虚拟机

docker-machine ls //查看虚拟机状态

# 附录：docker应用

博文：https://my.oschina.net/huangyong/blog/372491?fromerr=kHrZPM01

***一个java应用***

1. 下载Jpress项目(项目已过期)

https://github.com/jpressprojects/jpress

1. 下载tomcat

docker run -it --rm -p 8888:8080 tomcat:8.0

1. 编写Dockerfile

FROM hub.c.163.com/library/tomcat

MAINTAINER liuguoguo

COPY jpress.war /usr/local/tomcat/webapps //复制到tomcat的CATALINA\_HOME目录下

4.创建镜像

docker build -t jpress:latest .

5.运行程序

$ docker run -d -p 8888:8080 jpress

$ netstat -na | grep 8888 //查看端口是否监听

6.安装mysql

docker pull mysql

docker run -d -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=111111 -e MYSQL\_DATABASE=jpress mysql

7.重启web容器

docker restart containerID

***部署一个静态网站***

$ docker run -p 80 --name web -it ubuntu /bin/bash

# apt-get update

# apt-get install -y nginx

# apt-get install -y vim

# mkdir -p /var/www/html

# vim /var/www/html/index.html

# vim /etc/nginx/sites-enabled/default → 修改root的路径