# 命令总结

## 帮助/辅助

$ docker version

$ docker info

$ docker --help

## 镜像

$ docker images

-a -q --digests --no-trunc

$ docker search image\_name

-s : $ docker search -s 30 tomcat

--no-trunc

--automated : 只列出automated build 类型镜像

$ docker pull image\_name:tag

$ docker rmi [image\_name:tag ...]

-f

$ docker rmi -f $(docker images -qa) 全部删除镜像

$ docker history imageId //列出镜像的变更历史

$ docker tag image\_name:tag new\_name:tag //为镜像重命名（包括tag）

## 容器

$ docker run [options] image [command] [arg...] 新建/启动容器

[options]:

--name=”new\_name” → 为容器指定一个名词

-d → 后台运行

-i → 以交互模式运行

-t → 为容器重新分配一个伪输入终端

-P → 随机端口映射

-p → 指定端口映射；有四种模式

**ip:hostPort:containerPort**

ip::containerPort

**hostPort:containerPort**

containerPort

$ docker run -d centos

docker ps查看，发现容器已退出

说明：docker容器后台运行，就必须有一个前台的进程，容器运行的命令如果不是那些一直挂起的命令（比如：top、tail），就会自动退出。因为docker觉得此容器已经没事可做了

$ docker ps [options]

[options]

-a → 列出当前运行 + 历史运行过的容器

-l → 显示最近创建的容器

-n → 显示n个最近创建的容器

-q → 静默模式，只显示容器编号

--no-trunc

* 退出容器

exit：停止并退出

Ctrl+P+Q：不停止退出

* 进入正在运行的容器

$ docker attach containerID

→ 直接进入容器启动命令的终端，不会启动新进程

$ docker exec -it containerID bashShell

→ 在容器中打开新的终端，并可以启动新的进程

$ docker exec -it containerID ls -l /tmp

→ 只输出 $ ls -l /tmp 的结果，不会进入容器交互：隔空操作容器内部

$ docker exec -it containerID /bin/bash

→ 进入容器交互，打开新的终端

$ docker start containerID/containerName → 启动容器

$ docker restart containerID → 重启容器

$ docker stop containerID/containerName → 停止容器

$ docker kill containerID/containerName → 强制停止容器

$ docker rm containerID → 删除已停止的容器

一次删除所有容器：

$ docker rm -f $(docker ps -a -q)

$ docker ps -a -q | xargs docker rm

$ docker logs -f -t --tain n containerID → 查看container日志

-t 加入时间戳；

-f 跟随最新的日志打印，追加；

--tail 数字：显示最后多少条

$ docker top containerID → 查看容器中的进程

$ docker inspect containerID → 查看容器细节（内部）

$ docker cp containerID:container路径 目的主机路径

→ 把容器中的文件copy到宿主机中

$ docker commit [options] containerID 镜像名:[tag]

-m=”描述” -a=”作者”

$ docker commit -a=”nyfblack” -m=”del docs” containerID nyf/tomcat:1.2

→ 把tomcat的docs删除，重新做一个镜像

$ docker port container\_name #查看端口映射

## 数据卷

* 命令

$ docker run -it -v /宿主机绝对路径目录**:**/容器目录 镜像名

e.g. $ docker run -it -v /myDataVolume:/dataVolume centos

→ 同时在Host上生成myDataVolume目录，在容器中生成dataVolume目录

命令带权限：

$ docker run -it -v /myDataVolume:/dataVolume:ro centos

→ 容器侧目录为只读：ro -> readonly

* dockerfile 添加

dockerfile 的volume指令添加数据卷

→ 基于移植性的考虑，dockerfile中不支持host目录的设置，host的目录自动生成

构建：

$ docker build -f /mydocker/Dockerfile -t zy/centos **.**

→ 如果dockerfile的命名为Dockerfile 则-f参数指定文件可省略；docker会默认使用名为Dockerfile的文件创建镜像

e.g.Dockerfile文件如下：

#volume test

FROM centos

VOLUME [“data1”,”data2”]

CMD /bin/bash

* 问题及解决方案

如果docker挂载主机目录，docker访问出现以下错误：

cannot open directory. : Permission denied

这是权限问题；解决办法：在挂载目录后添加如下参数即可：

--privileged=true

* 数据卷容器

$ docker run -it --name dc02 --volume-from dc01 centos

→ dc01为父容器，dc02继承dc01的数据卷配置

→ 容器之间配置信息的传递，数据卷的生命周期一直持续到没有容器使用他为止

# Dockerfile构建image

Dockerfile ----build---> image ----run----> container

面向开发 交付标准 涉及部署和运维

* Dockerfile命令

|  |  |
| --- | --- |
| FROM | base image |
| EXPOSE | 暴露端口 |
| MAINTAINER | 维护者 |
| .dockerignore | 忽略；与。gitignore相似 |

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

ADD：tar解压/URL下载 + copy

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

→ ENV MY\_PATH /usr/mytest //定义

→ WORKDIR $MY\_PATH //使用

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

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/

* centos的Dockerfile

FROM scratch //scratch→基镜像，相当于类中的object类

MAINTAINER The Centos Project

ADD c68-docker.tar.xz/

LABEL name=”Centos Base Image”\

vendor=”Centos”\

license=”GPLv2”\

build-data=”2016-06-02”

CMD [“bin/bash”]

# 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

docker为每个container都分配了一个Mac地址

* ubuntu的网桥管理工具：

# apt-get install bridge-utils

# brctl show → 查看网桥设备

* 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

启动的所有容器是加到同一虚拟网桥上的，所有可以互联

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

在/etc/hosts中可以查看到cct1的ip到webtest的映射

### 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；

修改配置之后需要重启docker服务才能生效；

### 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

Linux命令：

$ sudo iptables -L -n #查看设置

$ sudo iptables -f #清空当前内容，重新加载设置

## 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。

Iptables包含：

表（table） 同一类操作

链（chain）数据处理中的不同环节（阶段）

规则（rule）每个链下的操作

ACCEPT/REJECT/DROP

filter表中包含的链：

INPUT FORWARD OUTPUT

查看filter表：

$ sudo iptables -t filter -L -n

-t：指定查看的表明，默认查看filter表

-L：执行的操作

-n：列表显示

$ 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

-s：源路径ip：被访问的ip

-d：目标ip：被禁止的ip

-p：协议

--dport：端口

-j：访问规则

# Docker容器的跨主机连接

## 使用网桥实现跨主机容器连接

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；

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

# apt-get install openvswitch-switch //安装

$ 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>

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