**Docker基础**

针对已经有一定的Linux基础知识的用户。

**Docker是什么**

Docker是一个改进的容器技术。具体的“改进”体现在，Docker为容器引入了镜像，使得容器可以从预先定义好的模版（images）创建出来，并且这个模版还是分层的。

**Docker经常被提起的特点：**

* 轻量，体现在内存占用小，高密度
* 快速，毫秒启动
* 隔离，沙盒技术更像虚拟机

**Docker技术的基础：**

* namespace，容器隔离的基础，保证A容器看不到B容器. 6个名空间：User,Mnt,Network,UTS,IPC,Pid
* cgroups，容器资源统计和隔离。主要用到的cgroups子系统：cpu,blkio,device,freezer,memory
* unionfs，典型：aufs/overlayfs，分层镜像实现的基础

**Docker组件：**

* **docker Client**客户端————>向docker服务器进程发起请求，如:创建、停止、销毁容器等操作
* **docker Server**服务器进程—–>处理所有docker的请求，管理所有容器
* **docker Registry**镜像仓库——>镜像存放的中央仓库，可看作是存放二进制的scm

**Docker安装**

Docker的安装非常简单，支持目前所有主流操作系统，从Mac到Windows到各种Linux发行版 具体参考： [docker安装](https://docs.docker.com/installation/)

**Docker常见命令**

**容器相关操作**

* docker create # 创建一个容器但是不启动它
* docker run # 创建并启动一个容器
* docker stop # 停止容器运行，发送信号SIGTERM
* docker start # 启动一个停止状态的容器
* docker restart # 重启一个容器
* docker rm # 删除一个容器
* docker kill # 发送信号给容器，默认SIGKILL
* docker attach # 连接(进入)到一个正在运行的容器
* docker wait # 阻塞到一个容器，直到容器停止运行

**获取容器相关信息**

* docker ps # 显示状态为运行（Up）的容器
* docker ps -a # 显示所有容器,包括运行中（Up）的和退出的(Exited)
* docker inspect # 深入容器内部获取容器所有信息
* docker logs # 查看容器的日志(stdout/stderr)
* docker events # 得到docker服务器的实时的事件
* docker port # 显示容器的端口映射
* docker top # 显示容器的进程信息
* docker diff # 显示容器文件系统的前后变化

**导出容器**

* docker cp # 从容器里向外拷贝文件或目录
* docker export # 将容器整个文件系统导出为一个tar包，不带layers、tag等信息

**执行**

* docker exec # 在容器里执行一个命令，可以执行bash进入交互式

**镜像操作**

* docker images # 显示本地所有的镜像列表
* docker import # 从一个tar包创建一个镜像，往往和export结合使用
* docker build # 使用Dockerfile创建镜像（推荐）
* docker commit # 从容器创建镜像
* docker rmi # 删除一个镜像
* docker load # 从一个tar包创建一个镜像，和save配合使用
* docker save # 将一个镜像保存为一个tar包，带layers和tag信息
* docker history # 显示生成一个镜像的历史命令
* docker tag # 为镜像起一个别名

**镜像仓库(registry)操作**

* docker login # 登录到一个registry
* docker search # 从registry仓库搜索镜像
* docker pull # 从仓库下载镜像到本地
* docker push # 将一个镜像push到registry仓库中

**获取Container IP地址（Container状态必须是Up）**

docker inspect id | grep IPAddress | cut -d '"' -f 4

**获取端口映射**

docker inspect -f '{{range $p, $conf := .NetworkSettings.Ports}} {{$p}} -> {{(index $conf 0).HostPort}} {{end}}' id

**获取环境变量**

docker exec container\_id env

**杀掉所有正在运行的容器**

docker kill $(docker ps -q)

**删除老的(一周前创建)容器**

docker ps -a | grep 'weeks ago' | awk '{print $1}' | xargs docker rm

**删除已经停止的容器**

docker rm `docker ps -a -q`

**删除所有镜像，小心**

docker rmi $(docker images -q)

**Dockerfile**

Dockerfile是docker构建镜像的基础，也是docker区别于其他容器的重要特征，正是有了Dockerfile，docker的自动化和可移植性才成为可能。 不论是开发还是运维，学会编写Dockerfile几乎是必备的，这有助于你理解整个容器的运行。

**FROM , 从一个基础镜像构建新的镜像**

FROM ubuntu

**MAINTAINER , 维护者信息**

MAINTAINER William <wlj@nicescale.com>

**ENV , 设置环境变量**

ENV TEST 1

**RUN , 非交互式运行shell命令**

RUN apt-get -y update

RUN apt-get -y install nginx

**ADD , 将外部文件拷贝到镜像里,src可以为url**

ADD http://nicescale.com/ /data/nicescale.tgz

**WORKDIR /path/to/workdir, 设置工作目录**

WORKDIR /var/www

**USER , 设置用户ID**

USER nginx

**VULUME <#dir>, 设置volume**

VOLUME [‘/data’]

**EXPOSE , 暴露哪些端口**

EXPOSE 80 443

**ENTRYPOINT [‘executable’, ‘param1’,’param2’]执行命令**

ENTRYPOINT ["/usr/sbin/nginx"]

**CMD [“param1”,”param2”]**

CMD ["start"]

docker创建、启动container时执行的命令，如果设置了ENTRYPOINT，则CMD将作为参数

**Dockerfile最佳实践**

* 尽量将一些常用不变的指令放到前面
* CMD和ENTRYPOINT尽量使用json数组方式

**通过Dockerfile构建image**

docker build csphere/nginx:1.7 .

**镜像仓库Registry**

镜像从Dockerfile build生成后，需要将镜像推送(push)到镜像仓库。企业内部都需要构建一个私有docker registry，这个registry可以看作二进制的scm，CI/CD也需要围绕registry进行。

**部署registry**

mkdir /registry

docker run -p 80:5000 -e STORAGE\_PATH=/registry -v /registry:/registry registry:2.0

**推送镜像保存到仓库**

假设192.168.1.2是registry仓库的地址：

docker tag csphere/nginx:1.7 192.168.1.2/csphere/nginx:1.7

docker push 192.168.1.2/csphere/nginx:1.7

**几个简单小例子**

**容器操作**

1.创建并拉取busybox

# docker run -it --name con01 busybox:latest

/ # ip addr #容器里执行

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

Segmentation fault (core dumped)

/ # ping www.csphere.cn

PING www.csphere.cn (117.121.26.243): 56 data bytes

64 bytes from 117.121.26.243: seq=0 ttl=48 time=3.139 ms

64 bytes from 117.121.26.243: seq=1 ttl=48 time=3.027 ms

^C

--- www.csphere.cn ping statistics ---

2 packets transmitted, 2 packets received, 0% packet loss

round-trip min/avg/max = 3.027/3.083/3.139 ms

exit #退出容器

2.创建测试容器

docker run -d --name con03 csphere/test:0.1

efc9bda4a2ff2f479b18e0fc4698e42c47c9583a24c93f5ce6b28a828a172709

3.登陆到con03中

# docker exec -it con03 /bin/bash

[root@efc9bda4a2ff /]# exit

4.停止con03

# docker stop con03

con03

5.开启con03

# docker start con03

con03

6.删除con03

# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

efc9bda4a2ff csphere/test:0.1 "/usr/local/bin/run 4 minutes ago Up 17 seconds con03

99aa6ee25adc busybox:latest "/bin/sh" 14 minutes ago Exited (0) 12 minutes ago con02

831c93de9b9f busybox:latest "/bin/sh" 2 hours ago Up 27 minutes con01

# docker rm con02 #容器停止的状态

# docker rm -f con03 #容器开启的状态

**镜像操作**

1.从docker hub官方镜像仓库拉取镜像

# docker pull busybox:latest

atest: Pulling from busybox

cf2616975b4a: Pull complete

6ce2e90b0bc7: Pull complete

8c2e06607696: Already exists

busybox:latest: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to provide security.

Digest: sha256:38a203e1986cf79639cfb9b2e1d6e773de84002feea2d4eb006b52004ee8502d

Status: Downloaded newer image for busybox:latest

2.从本地上传镜像到镜像仓库

docker push 192.168.1.2/csphere/nginx:1.7

3.查找镜像仓库的某个镜像

# docker search centos/nginx

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

johnnyzheng/centos-nginx-php-wordpress 1 [OK]

sergeyzh/centos6-nginx 1 [OK]

hzhang/centos-nginx 1 [OK]

4.查看本地镜像列表

# docker images

TAG IMAGE ID CREATED VIRTUAL SIZE

docker.io/csphere/csphere 0.10.3 604c03bf0c9e 3 days ago 62.72 MB

docker.io/csphere/csphere latest 604c03bf0c9e 3 days ago 62.72 MB

csphere/csphere 0.10.3 604c03bf0c9e 3 days ago 62.72 MB

registry 2.0 2971b6ce766c 7 days ago 548.1 MB

busybox latest 8c2e06607696 3 weeks ago 2.43 MB

5.删除镜像

docker rmi busybox:latest #没有容器使用此镜像创建，如果有容器在使用此镜像会报错：Error response from daemon: Conflict, cannot delete 8c2e06607696 because the running container 831c93de9b9f is using it, stop it and use -f to force

FATA[0000] Error: failed to remove one or more images

docker rmi -f busybox:latest #容器使用此镜像创建，此容器状态为Exited

6.查看构建镜像所用过的命令

# docker history busybox:latest

IMAGE CREATED CREATED BY SIZE

8c2e06607696 3 weeks ago /bin/sh -c #(nop) CMD ["/bin/sh"] 0 B

6ce2e90b0bc7 3 weeks ago /bin/sh -c #(nop) ADD file:8cf517d90fe79547c4 2.43 MB

cf2616975b4a 3 weeks ago /bin/sh -c #(nop) MAINTAINER Jérôme Petazzo 0 B