Docker入门与实战

运维开发技术群：323779636

Docker技术交流群：516039855

技术博客：http://lizhenliang.blog.51cto.com

课程直播：http://opsdev.ke.qq.com

主讲人：李振良

目录

第一章 Docker介绍与安装

1、什么是Docker

2、Docker架构与内部组件

3、Docker有什么优点

4、虚拟机与容器区别

5、应用场景

6、Docker安装

第一章 Docker介绍与安装

什么是Docker

Docker是一个开源的应用容器引擎，基于LXC（Linux Container）内核虚拟化技术实现，提供一系列更强的功能，比如镜像、Dockerfile等；
Docker理念是将应用及依赖包打包到一个可移植的容器中，可发布到任意Linux发行版Docker引擎上。使用沙箱机制运行程序，程序之间相互隔离；
Docker使用Go语言开发。

第一章 Docker介绍与安装

Docker架构与内部组件

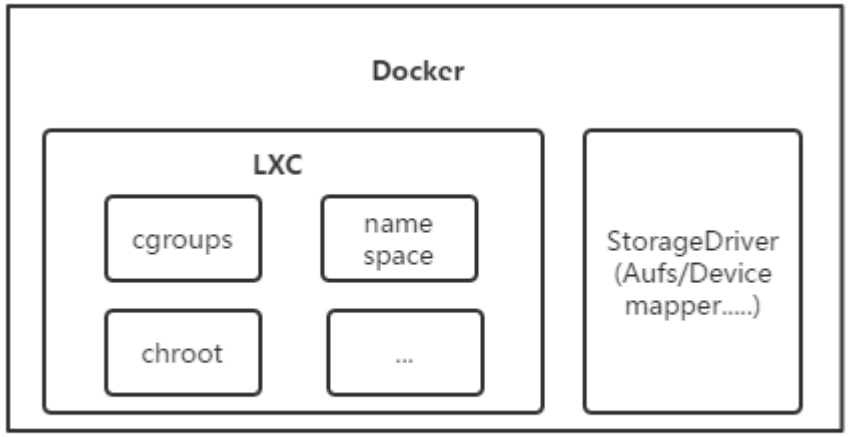
Docker采用C/S架构，Dcoker daemon作为服务端接受来自客户端请求，并处理这些请求，比如创建、运行容器等。客户端为用户提供一系列指令与Docker daemon交互。

Docker架构图

第一章 Docker介绍与安装

Docker架构与内部组件

LXC：Linux容器技术，共享内核，容器共享宿主机资源，使用namespace和cgroups对资源限制与隔离。
Cgroups（control groups）：Linux内核提供的一种限制单进程或者多进程资源的机制；比如CPU、内存等资源的使用限制。
NameSpace：命名空间，也称名字空间，Linux内核提供的一种限制单进程或者多进程资源隔离机制；一个进程可以属于多个命名空间。Linux内核提供了六种NameSpace：UTS、IPC、PID、Network、Mount和User。
AUFS（advanced multi layered unification filesystem）：高级多层统一文件系统，是UFS的一种，每个branch可以指定readonly（ro只读）、readwrite（读写）和whiteout-able（wo隐藏）权限；一般情况下，aufs只有最上层的branch才有读写权限，其他branch均为只读权限。
UFS（UnionFS）：联合文件系统，支持将不同位置的目录挂载到同一虚拟文件系统，形成一种分层的模型；成员目录称为虚拟文件系统的一个分支（branch）。



Docker内部结构

第一章 Docker介绍与安装

Docker架构与内部组件

Linux distribution	Supported storage drivers
Docker CE on Ubuntu	`aufs` , `devicemapper` , `overlay2` (Ubuntu 14.04.4 or later, 16.04 or later), `overlay` , `zfs`
Docker CE on Debian	`aufs` , `devicemapper` , `overlay2` (Debian Stretch), `overlay`
Docker CE on CentOS	`devicemapper`
Docker CE on Fedora	`devicemapper` , `overlay2` (Fedora 26 or later, experimental), `overlay` (experimental)

https://docs.docker.com/engine/userguide/storagedriver/selectadriver/#docker-ee-and-cs-engine

第一章 Docker介绍与安装

Docker有什么优点

- **持续集成**

在项目快速迭代情况下，轻量级容器对项目快速构建、环境打包、发布等流程就能提高工作效率。

- **版本控制**

每个镜像就是一个版本，在一个项目多个版本时可以很方便管理。

- **可移植性**

容器可以移动到任意一台Docker主机上，而不需要过多关注底层系统。

- **标准化**

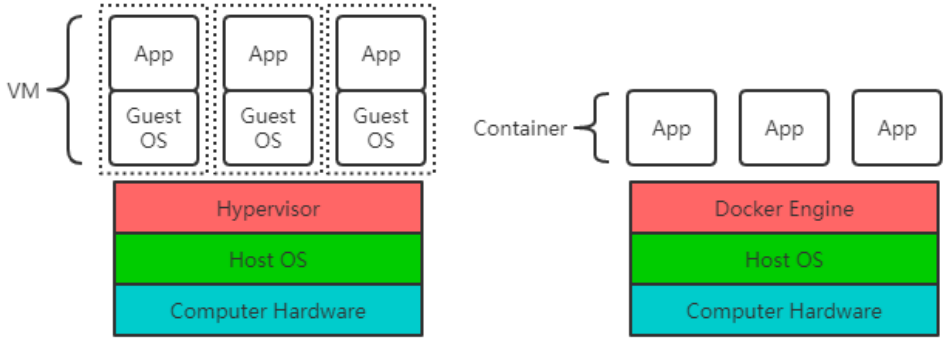
应用程序环境及依赖、操作系统等问题，增加了生产环境故障率，容器保证了所有配置、依赖始终不变。

- **隔离性与安全**

容器之间的进程是相互隔离的，一个容器出现问题不会影响其他容器。

第一章 Docker介绍与安装

虚拟机与容器区别



VM VS Container

第一章 Docker介绍与安装

虚拟机与容器区别

以KVM举例，与Docker对比

- **启动时间**

Docker秒级，KVM分钟级。

- **轻量级**

容器镜像大小通常以M为单位，虚拟机以G为单位。
容器资源占用小，要比虚拟机部署更快速。

- **性能**

容器共享宿主机内核，系统级虚拟化，占用资源少，没有Hypervisor层开销，容器性能基本接近物理机；
虚拟机需要Hypervisor层支持，虚拟化一些设备，具有完整的GuestOS，虚拟化开销大，因而降低性能，没有容器性能好。

- **安全性**

由于共享宿主机内核，只是进程级隔离，因此隔离性和稳定性不如虚拟机，容器具有一定权限访问宿主机内核，存在一定安全隐患。

- **使用要求**

KVM基于硬件的完全虚拟化，需要硬件CPU虚拟化技术支持；
容器共享宿主机内核，可运行在主流的Linux发行版，不用考虑CPU是否支持虚拟化技术。

第一章 Docker介绍与安装

应用场景

- **应用打包与部署自动化**

构建标准化的运行环境；
现在大多方案是在物理机和虚拟机上部署运行环境，面临问题是环境杂乱、完整性迁移难度高等问题，容器即开即用。

- **自动化测试和持续集成/部署**

自动化构建镜像和良好的REST API，能够很好的集成到持续集成/部署环境来。

- **部署与弹性扩展**

由于容器是应用级的，资源占用小，弹性扩展部署速度要更快。

- **微服务**

Docker这种容器华隔离技术，正式应对了微服务理念，将业务模块放到容器中运行，容器的可复用性大大增加了业务模块扩展性。

第一章 Docker介绍与安装

安装Docker

CentOS7

```
# 安装依赖包
yum install -y yum-utils device-mapper-persistent-data lvm2
# 添加Docker软件包源
yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
# 更新yum包索引
yum makecache fast
# 安装Docker CE
yum install docker-ce
# 启动
systemctl start docker
# 测试
docker run hello-world
docker version
# 卸载
yum remove docker-ce
rm -rf /var/lib/docker
```

第一章 Docker介绍与安装

安装Docker

Ubuntu14.06/16.04

```
# 安装证书
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
software-properties-common
# 添加Docker软件包源
$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
# 更新apt包索引
$ sudo apt-get update
# 安装Docker CE
$ sudo apt-get install docker-ce
# 测试
sudo docker run hello-world
sudo docker version
# 卸载Docker CE
$ sudo apt-get purge docker-ce
$ sudo rm -rf /var/lib/docker
```

第二章 镜像管理

1、什么是镜像

2、镜像从哪里来

3、镜像工作原理

4、镜像文件存储结构

5、镜像管理命令

第二章 镜像管理

什么是镜像？
简单说，Docker镜像是一个不包含Linux内核而又精简的Linux操作系统。

镜像从哪里来？
Docker Hub是由Docker公司负责维护的公共注册中心，包含大量的容器镜像，Docker工具默认从这个公共镜像库下载镜像。
https://hub.docker.com/explore
默认是国外的源，下载会慢，可以国内的源提供下载速度：
curl -sSL https://get.daocloud.io/daotools/set_mirror.sh | sh -s http://04be47cf.m.daocloud.io

第二章 镜像管理

镜像工作原理？

当我们启动一个新的容器时，Docker会加载只读镜像，并在其之上添加一个读写层，并将镜像中的目录复制一份到/var/lib/docker/aufs/mnt/容器ID为目录下，我们可以使用chroot进入此目录。如果运行中的容器修改一个已经存在的文件，那么会将该文件从下面的只读层复制到读写层，只读层的这个文件就会覆盖，但还存在，这就实现了文件系统隔离，当删除容器后，读写层的数据将会删除，只读镜像不变。

镜像文件存储结构？

docker相关文件存放在：/var/lib/docker目录下

/var/lib/docker/aufs/diff　　# 每层与其父层之间的文件差异

/var/lib/docker/aufs/layers/　　# 每层一个文件，记录其父层一直到根层之间的ID，大部分文件的最后一行都已，表示继承来自同一层

/var/lib/docker/aufs/mnt　　# 联合挂载点，从只读层复制到最上层可读写层的文件系统数据

在建立镜像时，每次写操作，都被视作一种增量操作，即在原有的数据层上添加一个新层；所以一个镜像会有若干个层组成。每次commit提交就会对产生一个ID，就相当于在上一层有加了一层，可以通过这个ID对镜像回滚

第二章 镜像管理

镜像管理命令

search
pull
push
images
commit
build
rmi
export
import
save
load

第三章 容器管理

1、创建容器常用选项

2、命令管理容器

3、容器数据持久化

4、搭建LNMP网站平台

创建容器常用选项

创建容器命令格式：

Usage: docker create [OPTIONS] IMAGE [COMMAND] [ARG...]

Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

OPTIONS（常用选项）

-i, --interactive
-t, --tty
-d, --detach
--add-host list
-a, --attach list
--cap-add list
--cap-drop list
--cidfile string
--device list
--dns list
-e, --env list
--env-file list
--expose list
-h, --hostname string
--ip string
--link list

--log-driver
（none、json-file、syslog、fluentd、splunk等）
--log-opt
--mount mount
--network string
--oom-kill-disable
--pid string
-p, --publish list
-P, --publish-all=true|false
--restart
--ulimit ulimit
-v, --volume list
--volumes-from list
-w, --workdir string

--cpu-period int
--cpu-quota int
-c, --cpu-shares int
-c, --cpu-shares int
--cpuset-cpus string
--device-read-bps list
--device-write-bps list
--device-read-iops list
--device-write-iops list
-m, --memory bytes
--memory-reservation bytes
--memory-swap bytes
--memory-swappiness int
--storage-opt list只支持devicemapper存储驱动

第三章 容器管理

命令管理容器

◆ 容器基本操作

ps
attach
rm
start
stop
kill
pause/unpause
rename

◆ 容器更多操作

inspect
exec
top
port
cp
diff
logs
stats
update
events

第三章 容器管理

数据持久化

1. 数据卷

将宿主机目录挂载到容器目录。

数据卷特点：

- 在容器启动初始化时，如果容器使用的宿主机挂载点有数据，这些数据就会拷贝到容器中。
- 数据卷可以在容器直接共享和重用。
- 可以直接对数据卷里的内容进行修改。
- 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删除。

示例：

```
docker run -itd --name web01 -v /container_data/web:/data Ubuntu
```

注：/container_data/web为宿主机目录，/data是容器中目录，目录不存在会自动创建。

2. 容器数据卷

将一个运行的容器作为数据卷，让其他容器通过挂载这个容器实现数据共享。

示例：

```
docker run -itd -v /data --name dvdata ubuntu
docker run -itd --name web01 --volumes-from dvdata ubuntu
```

搭建LNMP网站平台

```
# 创建mysql数据库容器
docker run -itd --name lnmp_mysql -p 3308:3306 -e MYSQL_ROOT_PASSWORD=123456 mysql --character-set-server=utf8
# 创建wp数据库
docker exec lnmp_mysql sh -c 'exec mysql -uroot -p"$MYSQL_ROOT_PASSWORD" -e"create database wp"'
# 创建PHP环境容器
docker run -itd --name lnmp_web --link lnmp_mysql:db -p 88:80 -v /container_data/web:/var/www/html richarvey/nginx-php
# 以wordpress博客为例测试
wget https://cn.wordpress.org/wordpress-4.7.4-zh_CN.tar.gz
tar zxf wordpress-4.7.4-zh_CN.tar.gz
mv wordpress/* /container_data/web/
# 浏览器测试访问
http://IP:88
```

第四章 网络管理

1、网络模式

2、容器网络访问原理

3、桥接宿主机网络与配置固定IP

4、容器SSH连接

第四章 网络管理

网络模式

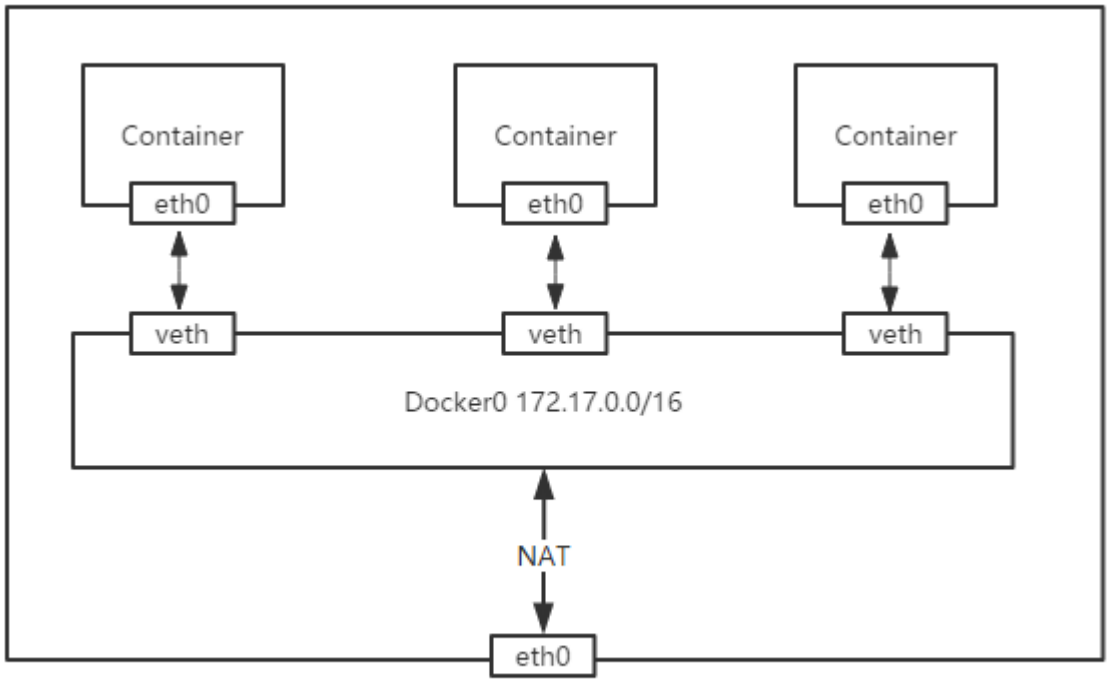
Docker支持五种网络模式

◆ bridge
默认网络，Docker启动后创建一个docker0网桥，默认创建的容器也是添加到这个网桥中；IP地址段是172.17.0.1/16
◆ host
容器不会获得一个独立的network namespace，而是与宿主机共用一个。
◆ none
获取独立的network namespace，但不为容器进行任何网络配置。
◆ container
与指定的容器使用同一个network namespace，网卡配置也都是相同的。
◆ 自定义
自定义网桥，默认与bridge网络一样。

第四章 网络管理

网络模式

先创建一个docker0的网桥，使用veth pair创建一对虚拟网卡，一端放到新创建的容器中，并重命名eth0，另一端放到宿主机上，以veth+随机7个字符串命名，并将这个网络设备加入到docker0网桥中，网桥自动为容器分配一个IP，并设置docker0的IP为容器默认网关。所以容器默认网络都加入了这个网桥，因此都可以彼此通信。同时在iptables添加SNAT转换网络段IP，以便容器访问外网。



Docker主机默认网络

第四章 网络管理

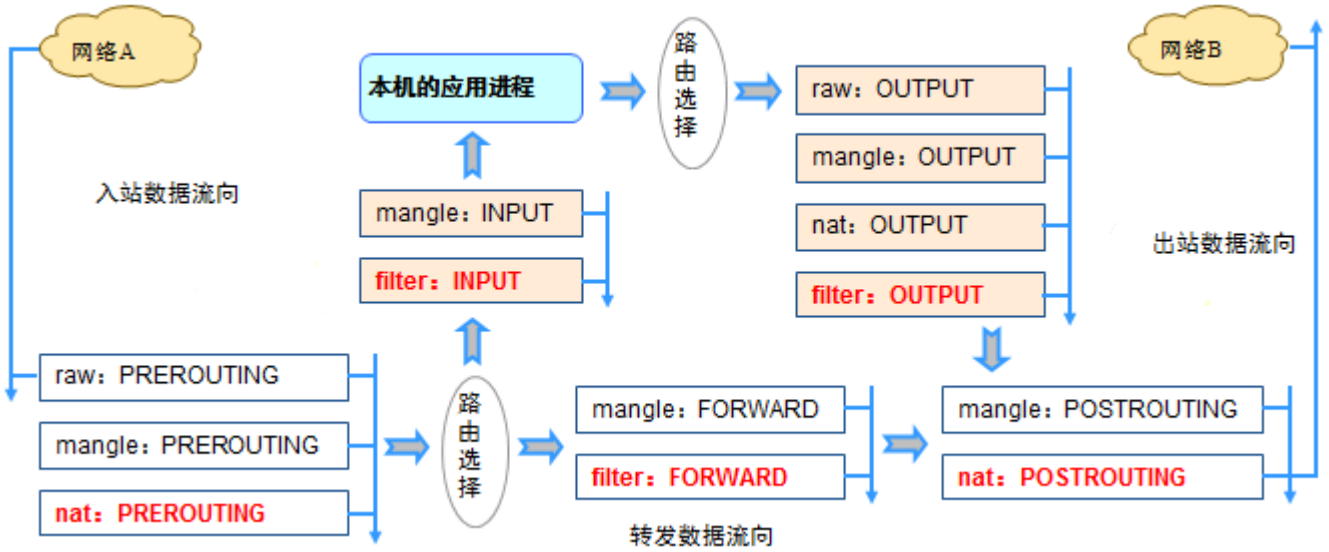
容器网络访问原理

◆ Linux IP信息包过滤原理

Docker主要通过netfilter/iptables实现网络通信。

iptables由netfilter和iptables组成，netfilter组件是Linux内核集成的信息包过滤系统，它维护一个信息包过滤表，这个表用于控制信息包过滤处理的规则集。而iptables只是一个在用户空间的工具，用于增删改查这个过滤表的规则。

表	链
filter（过滤）	INPUT、OUTPUT、FORWARD
nat（地址转换）	PREROUTING、POSTROUTING、OUTPUT
mangle（拆包、修改、封装）	INPUT、OUTPUT、PREROUTING、POSTROUTING、OUTPUT
raw（数据包状态跟踪）	PREROUTING、OUTPUT



容器网络访问原理

◆ 容器访问外部

```
# iptables -t nat -nL
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  tcp  --  172.17.0.2          172.18.0.2           tcp dpt:80
```

◆ 外部访问容器

```
# iptables -t nat -nL
Chain DOCKER (2 references)
target     prot opt source               destination
DNAT       tcp  --  0.0.0.0/0           0.0.0.0/0            tcp dpt:88 to:172.18.0.2:80
```

容器网络访问原理

◆ 桥接宿主机网络

临时生效：

```
# 网桥名称
br_name=br0
# 添加网桥
brctl addbr $br_name
# 给网桥设置IP
ip addr add 192.168.1.120/24 dev $br_name
# 删除已存在的eth0网卡配置
ip addr del 192.168.1.120/24 dev eth0
# 激活网桥
ip link set $br_name up
# 添加eth0到网桥
brctl addif $br_name eth0
```

还需要在Docker启动时桥接这个网桥：

```
vi /etc/default/docker
DOCKER_OPTS="-b=br0"
service docker restart
```

永久生效：

```
# vi /etc/network/interfaces
auto eth0
iface eth0 inet static

auto br0
iface br0 inet static
    address 192.168.1.120
    netmask 255.255.255.0
    gateway 192.168.1.1
    dns-nameservers 192.168.1.1
    bridge_ports eth0
```

容器网络访问原理

◆ 配置固定IP

```
C_ID=$(docker run -itd --net=none ubuntu)
C_PID=$(docker inspect -f '{{.State.Pid}}' $C_ID)
# 创建network namespace目录并将容器的network namespace软连接到此目录，以便ip netns命令读取
mkdir -p /var/run/netns
ln -s /proc/$C_PID/ns/net /var/run/netns/$C_PID
# 添加虚拟网卡veth+容器PID，类型是veth pair，名称是vp+容器PID
ip link add veth$C_PID type veth peer name vp$C_PID
# 添加虚拟网卡到br0网桥
brctl addif br0 veth$C_PID
# 激活虚拟网卡
ip link set veth$C_PID up
# 设置容器网络信息
IP='192.168.1.123/24'
GW='192.168.1.1'
# 给进程配置一个network namespace
ip link set vp$C_PID netns $C_PID
# 在容器进程里面设置网卡信息
ip netns exec $C_PID ip link set dev vp$C_PID name eth0
ip netns exec $C_PID ip link set eth0 up
ip netns exec $C_PID ip addr add $IP dev eth0
ip netns exec $C_PID ip route add default via 192.168.1.1
```

容器网络访问原理

◆ pipework

如果你觉得使用上面命令比较复杂，也有别人封装好的脚本：

```
git clone https://github.com/jpetazzo/pipework.git
cp pipework/pipework /usr/local/bin/
docker run -itd --net=none --name test01 ubuntu
pipework br0 test01 192.168.1.88/24@192.168.1.1
```

第四章 网络管理

容器SSH连接

```
# docker run -itd --name test01 centos:6
# docker attach test01
/# yum install openssh-server
/# passwd root
# docker commit test01 centos6_ssh
# docker run -itd --name test03 -p 2222:22 centos6_ssh
```

第五章 Dockerfile

1、Dockerfile指令

2、Build镜像命令

3、构建PHP网站环境镜像

4、构建JAVA网站环境镜像

5、构建支持SSH服务的镜像

第五章 Dockerfile

Dockerfile常用指令

指令	描述	指令	描述
FROM	构建的新镜像是基于哪个镜像 例如：FROM centos:6	COPY	拷贝文件或目录到镜像，用法同上 例如：COPY ./start.sh /start.sh
MAINTAINER	镜像维护者姓名或邮箱地址 例如：MAINTAINER lizhenliang	ENTRYPOINT	运行容器时执行的Shell命令 例如： ENTRYPOINT [“/bin/bash”, “-c”, “/start.sh”] ENTRYPOINT /bin/bash -c ‘/start.sh’
RUN	构建镜像时运行的Shell命令 例如： RUN [“yum”, “install”, “httpd”] RUN yum install httpd	VOLUME	指定容器挂载点到宿主机自动生成的目录或其他容器 例如： VOLUME ["/var/lib/mysql"]
CMD	运行容器时执行的Shell命令 例如： CMD [“-c”, “/start.sh”] CMD ["/usr/sbin/sshd", "-D"] CMD /usr/sbin/sshd -D	USER	为RUN、CMD和ENTRYPOINT执行命令指定运行用户 USER <user>[:<group>] or USER <UID>[:<GID>] 例如：USER lizhenliang
EXPOSE	声明容器运行的服务端口 例如：EXPOSE 80 443	WORKDIR	为RUN、CMD、ENTRYPOINT、COPY和ADD设置工作目录 例如：WORKDIR /data
ENV	设置容器内环境变量 例如：ENV MYSQL_ROOT_PASSWORD 123456	HEALTHCHECK	健康检查 HEALTHCHECK --interval=5m --timeout=3s \ CMD curl -f http://localhost/ || exit 1
ADD	拷贝文件或目录到镜像，如果是URL或压缩包会自动下载或自动解压 ADD <src>… <dest> ADD [“<src>”,… “<dest>”] ADD https://xxx.com/html.tar.gz /var/www/html ADD html.tar.gz /var/www/html	ARG	在构建镜像时指定一些参数 例如： FROM centos:6 ARG user # ARG user=root USER $user # docker build --build-arg user=lizhenliang Dockerfile .

第五章 Dockerfile

Dockerfile常用指令

RUN、CMD和ENTRYPOINT指令区别

1. RUN在building时运行，可以写多条
2. CMD和ENTRYPOINT在运行container时运行，只能写一条，如果写多条，最后一条生效。
3. CMD在run时可以被COMMAND覆盖，ENTRYPOINT不会被COMMAND覆盖，但可以指定—entrypoint覆盖。

第五章 Dockerfile

Build镜像命令

使用Dockerfile文件构建镜像

```
Usage:  docker build [OPTIONS] PATH | URL | -
Options:
-t, --tag list     # 镜像名称
-f, --file string  # 指定Dockerfile文件位置
```

示例：

```
docker build .   # 默认找当前目录以Dockerfile为命名的文件
docker build -t shykes/myapp .
docker build -t shykes/myapp -f /path/Dockerfile /path
docker build -t shykes/myapp - < Dockerfile
docker build -t shykes/myapp - < context.tar.gz
docker build -t shykes/myapp http://www.example.com/Dockerfile
docker build -f shykes/myapp http://www.example.com/contex.tar.gz
```

第五章 Dockerfile

构建PHP网站环境镜像

```
FROM centos:6
MAINTAINER lizhenliang

RUN yum install -y httpd php php-gd php-mysql mysql mysql-server

ENV MYSQL_ROOT_PASSWORD 123456

RUN echo "<?php phpinfo()?>" > /var/www/html/index.php

ADD start.sh /start.sh
RUN chmod +x /start.sh

ADD https://cn.wordpress.org/wordpress-4.7.4-zh_CN.tar.gz /var/www/html
COPY wp-config.php /var/www/html/wordpress

VOLUME ["/var/lib/mysql"]

CMD /start.sh

EXPOSE 80 3306
```

```
# cat start.sh
service httpd start
service mysqld start
mysqladmin -uroot password $MYSQL_ROOT_PASSWORD
tail -f
```

第五章 Dockerfile

构建JAVA网站环境镜像

```
FROM centos:6
MAINTAINER lizhenliang

ADD jdk-8u45-linux-x64.tar.gz /usr/local

ENV JAVA_HOME /usr/local/jdk1.8.0_45

ADD http://mirrors.tuna.tsinghua.edu.cn/apache/tomcat/tomcat-8/v8.0.45/bin/apache-tomcat-8.0.45.tar.gz /usr/local

WORKDIR /usr/local/apache-tomcat-8.0.45
ENTRYPOINT ["bin/catalina.sh", "run"]

EXPOSE 8080
```

第五章 Dockerfile

构建支持SSH服务的镜像

```
FROM centos:6
MAINTAINER lizhenliang

ENV ROOT_PASSWORD 123456

RUN yum install -y openssh-server
RUN echo $ROOT_PASSWORD |passwd --stdin root

RUN ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key
RUN ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key

CMD ["/usr/sbin/sshd", "-D"]

EXPOSE 22
```

第六章 私有与公有镜像仓库

1、搭建私有镜像仓库

2、私有镜像仓库管理

3、Docker Hub公共镜像仓库使用

第六章 私有与公共镜像仓库

搭建私有仓库

Docker Hub作为Docker默认官方公共镜像；如果想自己搭建私有镜像仓库，官方也提供registry镜像，使得搭建私有仓库非常简单。

◆ **下载registry镜像并启动**

```
# docker pull registry
# docker run -d -v /opt/registry:/var/lib/registry -p 5000:5000 --restart=always --name registry registry
```

◆ **测试，查看镜像仓库中所有镜像**

```
# curl http://192.168.1.120:5000/v2/_catalog
{"repositories":[]}
```

第六章 私有与公共镜像仓库　私有仓库管理

◆ 配置私有仓库可信任

```
# vi /etc/docker/daemon.json
{"insecure-registries":["192.168.1.120:5000"]}
# service docker restart
```

◆ 打标签

```
# docker tag centos:6 192.168.1.120:5000/centos:6
```

◆ 上传

```
# docker push 192.168.1.120:5000/centos:6
```

◆ 下载

```
# docker pull 192.168.1.120:5000/centos:6
```

◆ 列出镜像标签

```
# curl http://192.168.1.120:5000/v2/centos/tags/list
```

第六章 私有与公共镜像仓库 Docker Hub公共镜像仓库使用

1、注册账号

https://hub.docker.com

2、登录Docker Hub

```
# docker login
```

或

```
# docker login --username=lizhenliang --password=123456
```

3、镜像打标签

```
# docker tag wordpress:v1 lizhenliang/wordpress:v1
```

4、上传

```
# docker push lizhenliang/wordpress:v1
```

搜索测试：

```
# docker search lizhenliang
```

5、下载

```
# docker pull lizhenliang/wordpress:v1
```

第七章 图形界面管理

1、DockerUI

2、Shipyard

第七章 图形界面管理

DockerUI

DockerUI是一个基于Docker API提供图形化页面简单的容器管理系统，支持容器管理、镜像管理。

```
docker run \
-d \
-p 9000:9000 \
-v /var/run/docker.sock:/docker.sock \
--name dockerui abh1nav/dockerui:latest \
-e="/docker.sock"
```

也可以通过Rest API管理：

```
docker run \
-d \
-p 9000:9000 \
--name dockerui \
-e "http://<dockerd host ip>:2375"
abh1nav/dockerui:latest
http://<dockerd host ip>:9000
```

第七章 图形界面管理

Shipyard

Shipyard也是基于Docker API实现的容器图形管理系统，支持container、images、engine、cluster等功能，可满足我们基本的容器部署需求。

Shipyard分为手动部署和自动部署。

镜像名称	运行服务	描述
rethinkdb	shipyard数据库	一个NoSQL数据库，用于存储shipyard系统的数据，比如账号、节点、容器等信息。
microbox/etcd	服务注册、发现系统	K/V存储系统，用于Swarm节点实现服务注册、发现。也支持consul、zookeeper。
shipyard/docker-proxy	docker API代理	连接本地/var/run/docker.sock代理，用于让Swarm Agent连接API管理。
swarm	swarm集群	官方管理Docker集群工具，使得多个engine为一个整体管理，对外提供Swarm manager API，用户就像操作单台Engine一样。
shipyard/shipyard	shipyard前端	容器Web管理系统，内部连接Swarm Manager管理容器和RethinkDB存储数据。

官方部署文档：https://www.shipyard-project.com/docs/deploy/

第八章 构建容器监控系统

cAdvisor+InfluxDB+Grafana

cAdvisor：Google开源的工具，用于监控Docker主机和容器系统资源，通过图形页面实时显示数据，但不存储；它通过宿主机/proc、/sys、/var/lib/docker等目录下文件获取宿主机和容器运行信息。
InfluxDB：是一个分布式的时间序列数据库，用来存储cAdvisor收集的系统资源数据。
Grafana：可视化展示平台，可做仪表盘，并图表页面操作很方面，数据源支持zabbix、Graphite、InfluxDB、OpenTSDB、Elasticsearch等

它们之间关系：
cAdvisor容器数据采集->InfluxDB容器数据存储->Grafana可视化展示

第八章 构建容器监控系统

cAdvisor+InfluxDB+Grafana

部署：

influxdb

```
docker run \
-d \
-p 8083:8083 \
-p 8086:8086 \
--name influxdb tutum/influxdb
```

cadvisor

```
docker run -d \
--volume=/:/rootfs:ro \
--volume=/var/run:/var/run:rw \
--volume=/sys:/sys:ro \
--volume=/var/lib/docker/:/var/lib/docker:ro \
--link influxdb:influxdb \
-p 8081:8080 \
--name=cadvisor \
google/cadvisor:latest \
-storage_driver=influxdb \
-storage_driver_db=cadvisor \
-storage_driver_host=influxdb:8086
```

grafana

```
docker run -d \
-p 3000:3000 \
-e INFLUXDB_HOST=influxdb \
-e INFLUXDB_PORT=8086 \
-e INFLUXDB_NAME=cadvisor \
-e INFLUXDB_USER=cadvisor \
-e INFLUXDB_PASS=cadvisor \
--link influxdb:influxsrv \
--name grafana \
grafana/grafana
```

谢谢