Docker技术入门与应用实战

个人介绍

讲师：李振良

资深运维工程师，曾混在IDC，大数据，金融行业。下到搬服务器，上到Linux平台架构设计。经重重磨练，具备各方综合能力。

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

关注微信公众号：DevOps大咖

专注于分享运维开发领域技术及经验教训，包括Linux、Shell、Python、Docker、数据库、网站架构、集群等主流技术。每日一篇高质量文章，助你快速提升专业能力！

课程目录

第一章 Docker介绍与安装

- Docker是什么
- Docker体系结构
- 内部组件
- 虚拟机与容器区别
- Docker应用场景
- Linux安装Docker

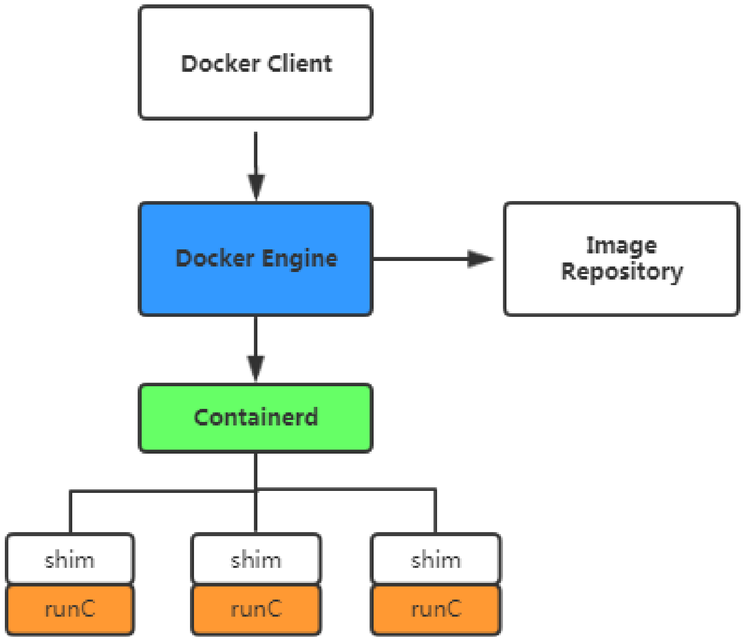
Docker是什么

Docker是一个开源的应用容器引擎，使用Go语言开发，基于Linux内核的cgroup，namespace，Union FS等技术，对应用进程进行封装隔离，并且独立于宿主机与其他进程，这种运行时封装的状态称为容器。

Docker早起版本实现是基于LXC，并进一步对其封装，包括文件系统、网络互联、镜像管理等方面，极大简化了容器管理。从0.7版本以后开始去除LXC，转为自行研发的libcontainer，从1.11版本开始，进一步演进为使用runC和containerd。

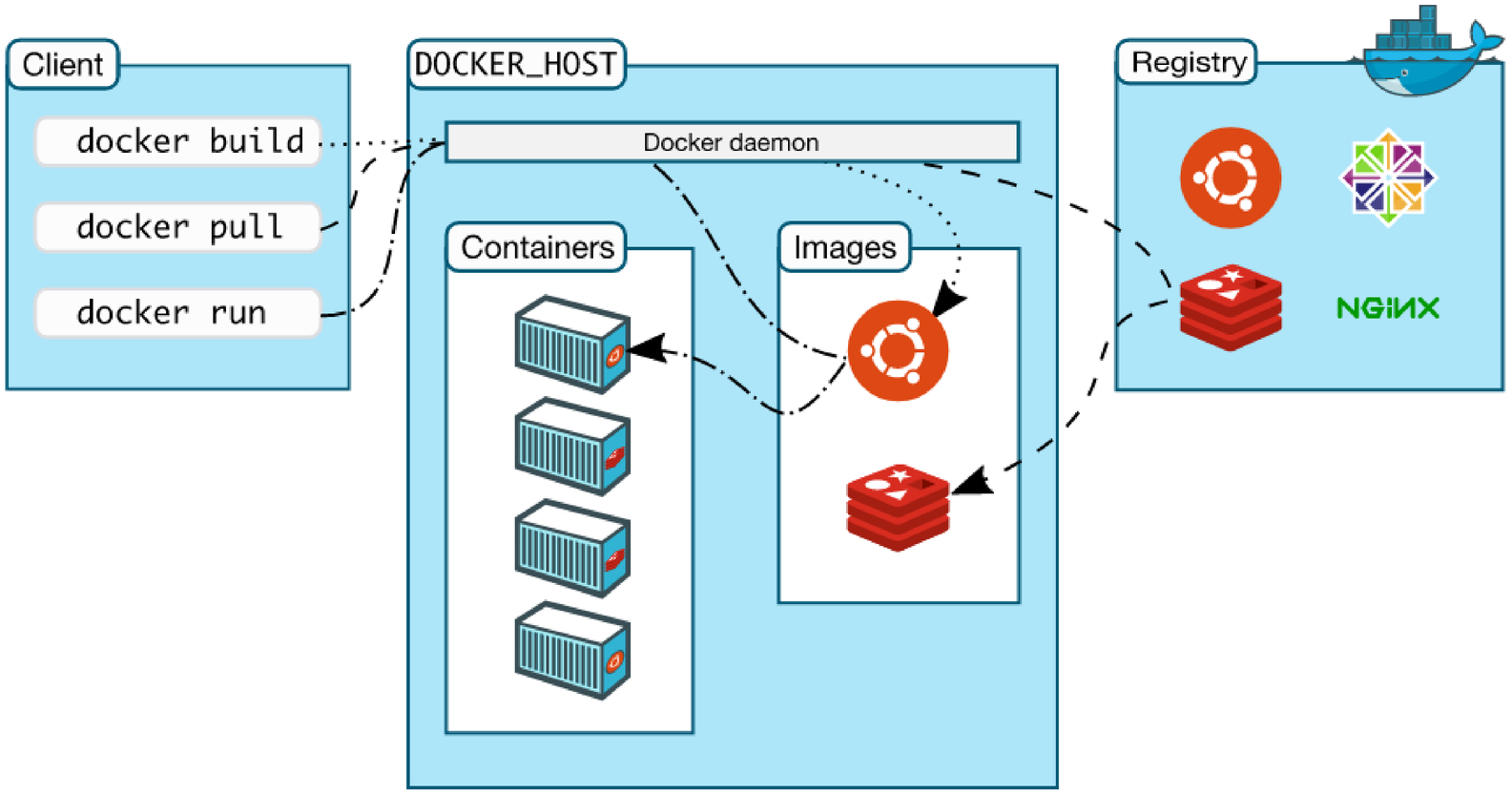
Docker理念是将应用及依赖包打包到一个可移植的容器中，可发布到任意Linux发行版Docker引擎上。使用沙箱机制运行程序，程序之间相互隔离。

Docker体系结构



Containerd：是一个简单的守护进程，使用runC管理容器。向Docker Engine提供接口。
Shim：只负责管理一个容器。
runC：是一个轻量级的工具，只用来运行容器。

Docker体系结构



内部组件

- Namespaces

命名空间，Linux内核提供的一种对进程资源隔离的机制，例如进程、网络、挂载点等资源。

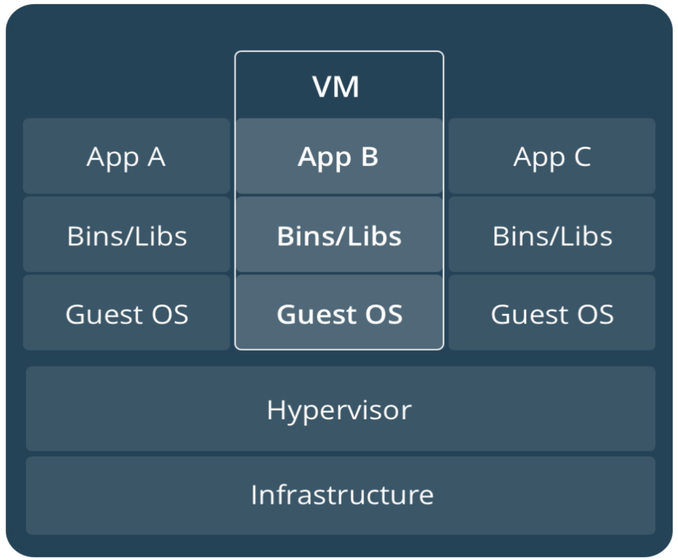
- CGroups

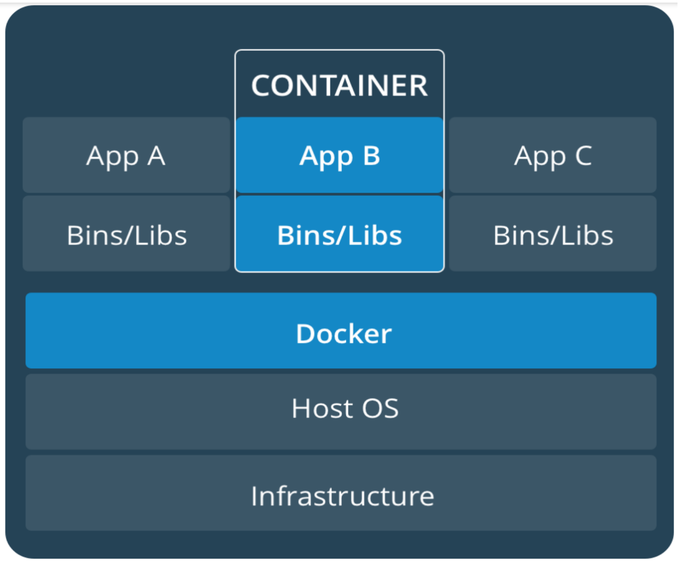
控制组，Linux内核提供的一种限制进程资源的机制；例如CPU、内存等资源。

- 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。

虚拟机与容器区别





虚拟机与容器区别

以KVM举例，与Docker对比

- **启动时间**

Docker秒级启动，KVM分钟级启动。

- **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。

应用场景

场景一：节省项目环境部署时间

1. 单项目打包
2. 整套项目打包
3. 新开源技术试用

场景二：环境一致性

场景三：持续集成

场景四：微服务

场景五：弹性伸缩

Linux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
# 卸载
yum remove docker-ce
rm -rf /var/lib/docker
```

官方安装文档：
https://docs.docker.com/engine/installation/linux/docker-ce/centos/#docker-ee-customers

Ubuntu14.06/16.04

```
# 安装证书
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
software-properties-common
# 添加Docker源的KEY
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
# 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
sudo apt-get install docker-ce
# 卸载
sudo apt-get purge docker-ce
sudo rm -rf /var/lib/docker
```

第二章 镜像管理

- 镜像是什么
- 镜像从哪里来
- 镜像与容器联系
- 存储驱动
- 镜像管理命令

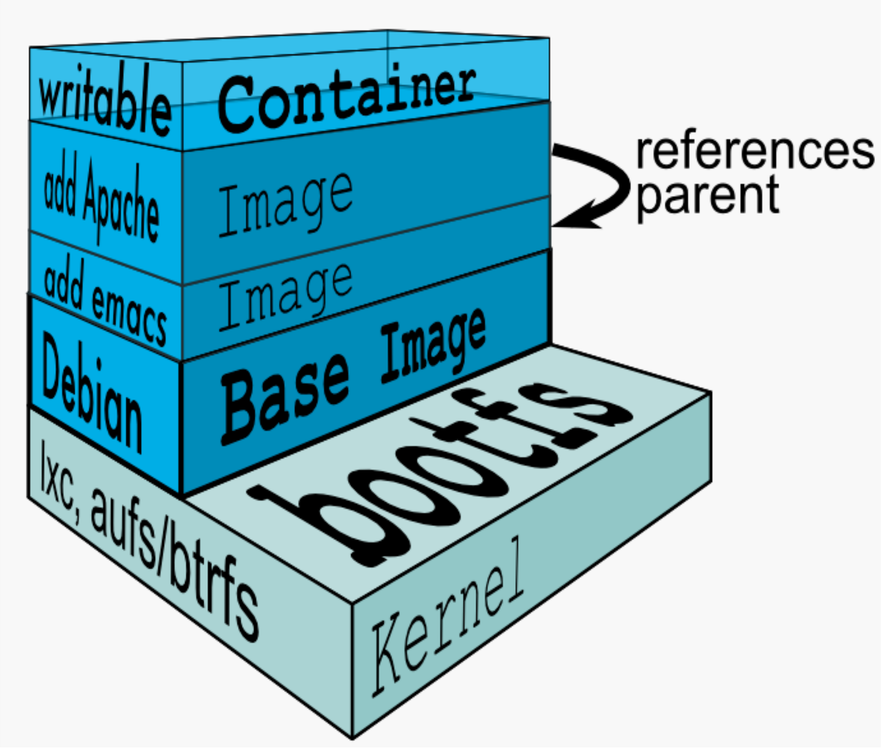
镜像

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，建议配置国内镜像仓库：

```
# vi /etc/docker/daemon.json
{
  "registry-mirrors": [ "https://registry.docker-cn.com"]
}
```

镜像与容器联系



镜像不是一个单一的文件，而是有多层构成。我们可以通过docker history <ID/NAME> 查看镜像中各层内容及大小，每层对应着Dockerfile中的一条指令。Docker镜像默认存储在/var/lib/docker/<storage-driver>中。

容器其实是在镜像的最上面加了一层读写层，在运行容器里做的任何文件改动，都会写到这个读写层。如果容器删除了，最上面的读写层也就删除了，改动也就丢失了。

Docker使用存储驱动管理镜像每层内容及可读写层的容器层。

存储驱动

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

Storage driver	Supported backing filesystems
`overlay`, `overlay2`	`ext4`, `xfs`
`aufs`	`ext4`, `xfs`
`devicemapper`	`direct-lvm`
`btrfs`	`btrfs`
`zfs`	`zfs`

镜像管理指令

指令	描述
ls	列出镜像
build	构建镜像来自Dockerfile
history	查看镜像历史
inspect	显示一个或多个镜像详细信息
pull	从镜像仓库拉取镜像
push	推送一个镜像到镜像仓库
rm	移除一个或多个镜像
prune	移除未使用的镜像。没有被标记或被任何容器引用的。
tag	创建一个引用源镜像标记目标镜像
export	导出容器文件系统到tar归档文件
import	导入容器文件系统tar归档文件创建镜像
save	保存一个或多个镜像到一个tar归档文件
load	加载镜像来自tar归档或标准输入

第三章 容器管理

- 创建容器常用选项
- 管理容器常用命令

创建容器常用选项

指令	描述	资源限制指令	描述
-i, --interactive	交互式	-m，--memory	容器可以使用的最大内存量
-t, --tty	分配一个伪终端	--memory-swap	允许交换到磁盘的内存量
-d, --detach	运行容器到后台	--memory-swappiness=<0-100>	容器使用SWAP分区交换的百分比（0-100，默认为-1）
-a, --attach list	附加到运行的容器	--memory-reservation	内存软限制，Docker检测主机容器争用或内存不足时所激活的软限制，使用此选项，值必须设置低于—memory，以使其优先
--dns list	设置DNS服务器	--oom-kill-disable	当宿主机内存不足时，内核会杀死容器中的进程。建议设置了-memory选项再禁用OOM。如果没有设置，主机可能会耗尽内存
-e, --env list	设置环境变量	--cpus	限制容器可以使用多少可用的CPU资源
--env-file list	从文件读取环境变量	--cpuset-cpus	限制容器可以使用特定的CPU
-p, --publish list	发布容器端口到主机	--cpu-shares	此值设置为大于或小于默认1024值，以增加或减少容器的权重，并使其可以访问主机CPU周期的更大或更小比例
-P, --publish-all	发布容器所有EXPOSE的端口到宿主机随机端口		
-h, --hostname string	设置容器主机名		
--ip string	指定容器IP，只能用于自定义网络		
--link list	添加连接到另一个容器		
--network	连接容器到一个网络		
--mount mount	挂载宿主机分区到容器		
-v, --volume list	挂载宿主机目录到容器		
--restart string	容器退出时重启策略，默认no [always|on-failure]		
--add-host list	添加其他主机到容器中/etc/hosts		

管理容器常用命令

指令	描述
ls	列出容器
inspect	显示一个或多个容器详细信息
attach	附加本地标准输入，输出和错误到一个运行的容器
exec	在运行容器中执行命令
commit	创建一个新镜像来自一个容器
cp	拷贝文件/文件夹到一个容器
logs	获取一个容器日志
port	列出或指定容器端口映射
stats	显示容器资源使用统计
top	显示一个容器运行的进程
update	更新一个或多个容器配置
stop/start	停止/启动一个或多个容器
rm	删除一个或多个容器

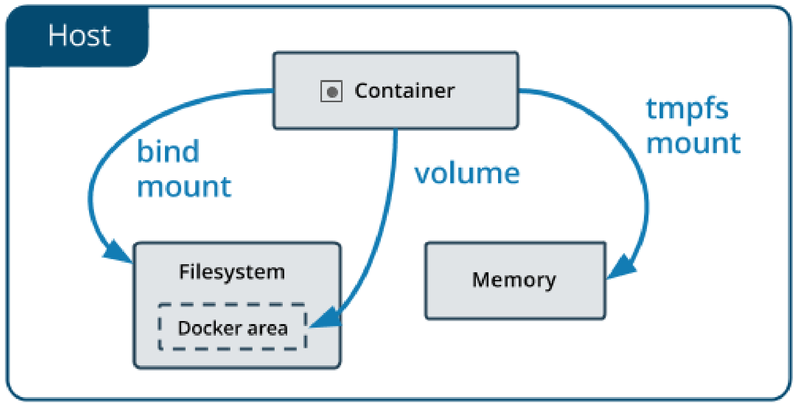
将Docker主机数据挂载到容器

Docker提供三种不同的方式将数据从宿主机挂载到容器中：volumes，bind mounts和tmpfs。

volumes：Docker管理宿主机文件系统的一部分（/var/lib/docker/volumes）。

bind mounts：可以存储在宿主机系统的任意位置。

tmpfs：挂载存储在宿主机系统的内存中，而不会写入宿主机的文件系统。



Volume

管理卷：

```
# docker volume create nginx-vol
# docker volume ls
# docker volume inspect nginx-vol
```

用卷创建一个容器：

```
# docker run -d -it --name=nginx-test --mount src=nginx-vol,dst=/usr/share/nginx/html nginx
# docker run -d -it --name=nginx-test -v nginx-vol:/usr/share/nginx/html nginx
```

清理：

```
# docker container stop nginx-test
# docker container rm nginx-test
# docker volume rm nginx-vol
```

注意：

1. 如果没有指定卷，自动创建。
2. 建议使用—mount，更通用。

官方文档：https://docs.docker.com/engine/admin/volumes/volumes/#start-a-container-with-a-volume

Bind Mounts

用卷创建一个容器：

```
# docker run -d -it --name=nginx-test --mount type=bind,src=/app/wwwroot,dst=/usr/share/nginx/html nginx
# docker run -d -it --name=nginx-test -v /app/wwwroot:/usr/share/nginx/html nginx
```

验证绑定：

```
# docker inspect nginx-test
```

清理：

```
# docker container stop nginx-test
# docker container rm nginx-test
```

注意：

1. 如果源文件/目录没有存在，不会自动创建，会抛出一个错误。
2. 如果挂载目标在容器中非空目录，则该目录现有内容将被隐藏。

官方文档：https://docs.docker.com/engine/admin/volumes/bind-mounts/#start-a-container-with-a-bind-mount

搭建LNMP网站平台

1、自定义网络

```
docker network create lnmp
```

2、创建Mysql数据库容器

```
docker run -itd \
--name lnmp_mysql \
--net lnmp \
-p 3306:3306 \
--mount src=mysql-vol,dst=/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=123456 \
mysql --character-set-server=utf8
```

3、创建所需数据库

```
docker exec lnmp_mysql sh \
-c 'exec mysql -uroot -p"$MYSQL_ROOT_PASSWORD" -e"create database wp"'
```

4、创建PHP环境容器

```
docker run -itd \
--name lnmp_web \
--net lnmp \
-p 88:80 \
--mount type=bind,src=/app/wwwroot,dst=/var/www/html richarvey/nginx-php-fpm
```

5、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 -C /app/wwwroot
# 浏览器测试访问
http://IP:88/wordpress
```

第五章 网络管理

- 网络模式
- 容器网络访问原理
- 桥接宿主机网络与配置固定IP地址

网络模式

Docker支持5种网络模式

- bridge

默认网络，Docker启动后默认创建一个docker0网桥，默认创建的容器也是添加到这个网桥中。

- 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。

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。

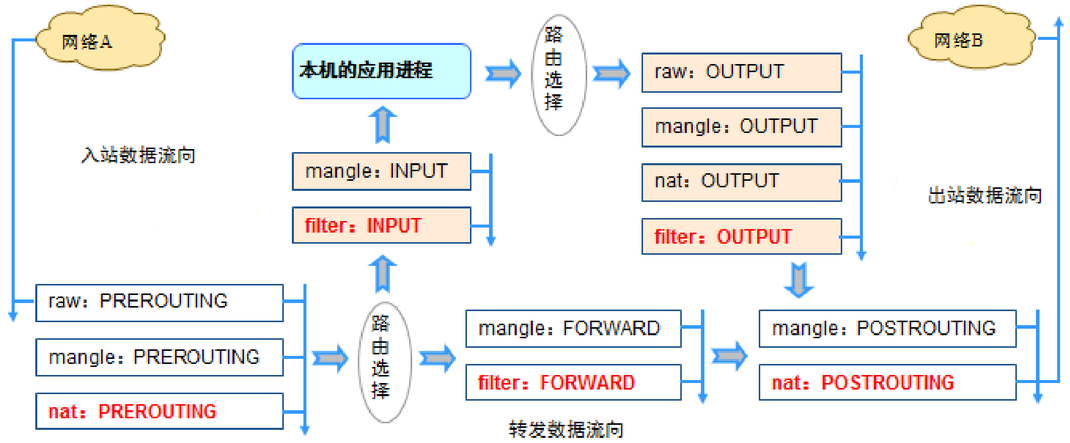


表	链
filter（过滤）	INPUT、OUTPUT、FORWARD
nat（地址转换）	PREROUTING、POSTROUTING、OUTPUT
mangle（拆包、修改、封装）	INPUT、OUTPUT、PREROUTING、POSTROUTING、OUTPUT
raw（数据包状态跟踪）	PREROUTING、OUTPUT

容器网络访问原理

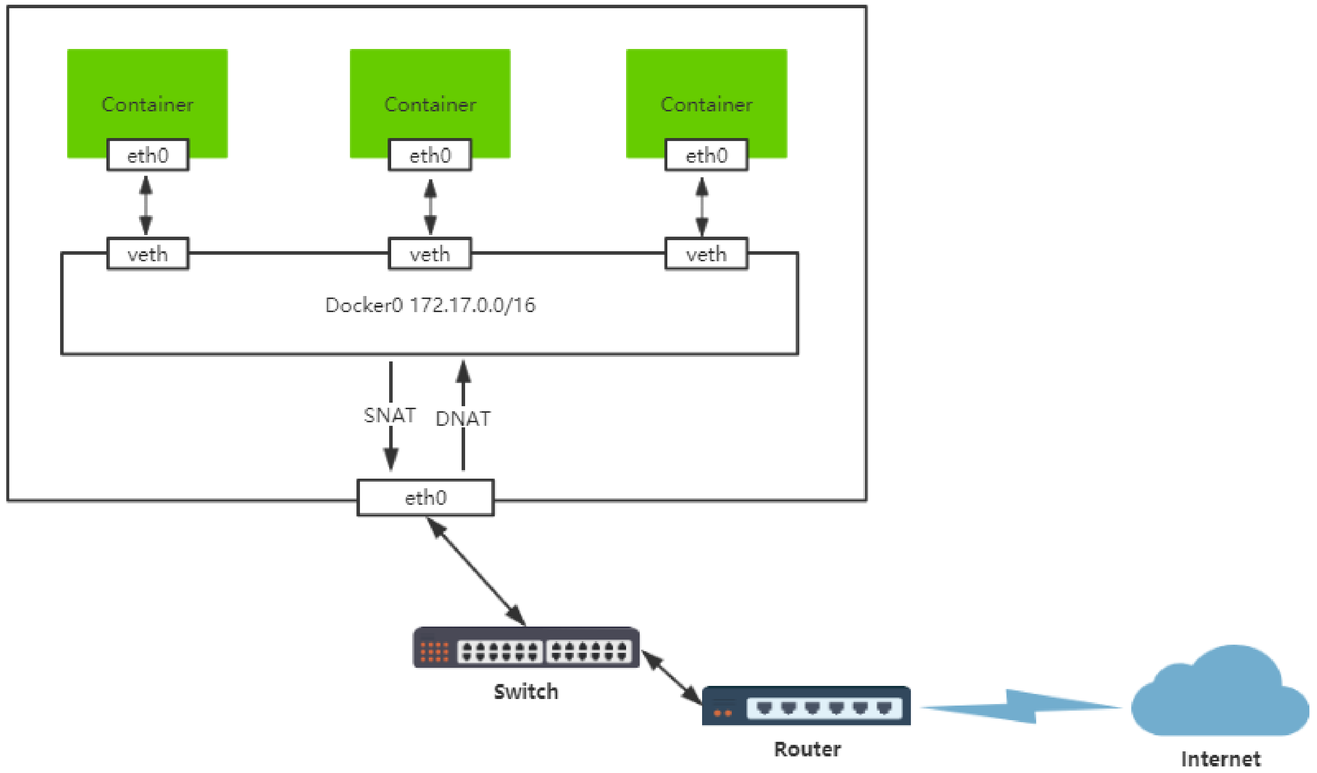
◆ 容器访问外部

```
# iptables -t nat -nL
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  172.17.0.0/16        0.0.0.0/0
```

◆ 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



桥接宿主机网络与配置固定IP地址

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 192.168.0.211/24 dev $br_name
# 删除已存在的eth0网卡配置
ip addr del 192.168.0.211/24 dev eth0
# 激活网桥
ip link set $br_name up
# 添加eth0到网桥
brctl addif $br_name eth0
# 添加路由
ip route add default via 192.168.0.1 dev br0
```

还需要在Docker启动时桥接这个网桥：

```
# vi /usr/lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd -b=br0
# systemctl restart docker
```

永久生效：

```
# vi /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
TYPE=Ethernet
ONBOOT=yes
BRIDGE=br0

# vi /etc/sysconfig/network-scripts/ifcfg-br0
DEVICE=br0
TYPE=Bridge
ONBOOT=yes
BOOTPROTO=static
IPADDR=192.168.0.211
NETMASK=255.255.255.0
GATEWAY=192.168.0.1
DNS1=114.114.114.114
```

桥接宿主机网络与配置固定IP地址

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$C_PID up
# 设置容器网络信息
IP='192.168.0.123/24'
GW='192.168.0.1'
# 给进程配置一个network namespace
ip link set vp$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
```

pipework工具配置容器固定IP

```
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 192.168.0.123/24@192.168.0.1
```

第六章 Dockerfile

- Dockerfile指令
- Build镜像命令
- 构建PHP网站环境镜像
- 构建JAVA网站环境镜像

Dockerfile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 --retries=3 \ 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 .

Build镜像命令

```
Usage:  docker image 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 .
docker build -t shykes/myapp .
docker build -t shykes/myapp -f /path/Dockerfile /path
```

构建PHP网站环境镜像

```
FROM centos:7
MAINTAINER www.aliangedu.com
RUN yum install -y gcc gcc-c++ make openssl-devel pcre-devel
ADD nginx-1.12.1.tar.gz /tmp

RUN cd /tmp/nginx-1.12.1 && \
    ./configure --prefix=/usr/local/nginx && \
    make -j 2 && \
    make install

RUN rm -rf /tmp/nginx-1.12.1* && yum clean all

COPY nginx.conf /usr/local/nginx/conf

WORKDIR /usr/local/nginx
EXPOSE 80
CMD ["./sbin/nginx", "-g", "daemon off;"]
```

```
FROM centos:7
MAINTAINER www.aliangedu.com
RUN yum install -y gcc gcc-c++ make gd-devel libxml2-devel libcurl-devel libjpeg-
devel libpng-devel openssl-devel
ADD php-5.6.31.tar.gz /tmp/

RUN cd /tmp/php-5.6.31 && \
    ./configure --prefix=/usr/local/php \
    --with-config-file-path=/usr/local/php/etc \
    --with-mysql --with-mysqli \
    --with-openssl --with-zlib --with-curl --with-gd \
    --with-jpeg-dir --with-png-dir --with-iconv \
    --enable-fpm --enable-zip --enable-mbstring && \
    make -j 4 && \
    make install && \
    cp /usr/local/php/etc/php-fpm.conf.default /usr/local/php/etc/php-fpm.conf && \
    sed -i "s/127.0.0.1/0.0.0.0/" /usr/local/php/etc/php-fpm.conf && \
    sed -i "21a \daemonize = no" /usr/local/php/etc/php-fpm.conf
COPY php.ini /usr/local/php/etc

RUN rm -rf /tmp/php-5.6.31* && yum clean all

WORKDIR /usr/local/php
EXPOSE 9000
CMD ["./sbin/php-fpm", "-c", "/usr/local/php/etc/php-fpm.conf"]
```

构建PHP网站环境镜像

1、自定义网络

```
docker network create lnmp
```

2、创建PHP容器

```
docker run -itd \
--name lnmp_php \
--net lnmp \
--mount type=bind,src=/app/wwwroot/,dst=/usr/local/nginx/html \
php:v1
```

3、创建Nginx容器

```
docker run -itd \
--name lnmp_nginx \
--net lnmp \
--mount type=bind,src=/app/wwwroot/,dst=/usr/local/nginx/html \
nginx:v1
```

4、创建MySQL容器

```
docker run -itd \
--name lnmp_mysql \
--net lnmp \
-p 3306:3306 \
--mount src=mysql-vol,dst=/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=123456 \
mysql --character-set-server=utf8
```

构建JAVA网站环境镜像

```
FROM centos:7
MAINTAINER www.aliangedu.com

ADD jdk-8u45-linux-x64.tar.gz /usr/local
ENV JAVA_HOME /usr/local/jdk1.8.0_45

ADD apache-tomcat-8.0.46.tar.gz /usr/local
COPY server.xml /usr/local/apache-tomcat-8.0.46/conf

RUN rm -f /usr/local/*.tar.gz

WORKDIR /usr/local/apache-tomcat-8.0.46
EXPOSE 8080
ENTRYPOINT ["./bin/catalina.sh", "run"]
```

创建容器：

```
docker run -itd \
--name=tomcat \
-p 8080:8080 \
--mount type=bind,src=/app/webapps/,dst=/usr/local/apache-tomcat-8.0.46/webapps \
tomcat:v1
```

第七章 镜像仓库

- 搭建私有镜像仓库
- 私有镜像仓库管理
- Docker Hub公共镜像仓库使用

搭建私有镜像仓库

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://192.168.0.212:5000/v2/_catalog
{"repositories":[]}
```

私有镜像仓库管理

1、配置私有仓库可信任

```
# vi /etc/docker/daemon.json
{"insecure-registries":["192.168.0.212:5000"]}
# systemctl restart docker
```

2、打标签

```
# docker tag centos:6 192.168.0.212:5000/centos:6
```

3、上传

```
# docker push 192.168.0.212:5000/centos:6
```

4、下载

```
# docker pull 192.168.0.212:5000/centos:6
```

5、列出镜像标签

```
# curl http://192.168.0.212:5000/v2/centos/tags/list
```

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
```

第八章 图形页面管理

Portainer

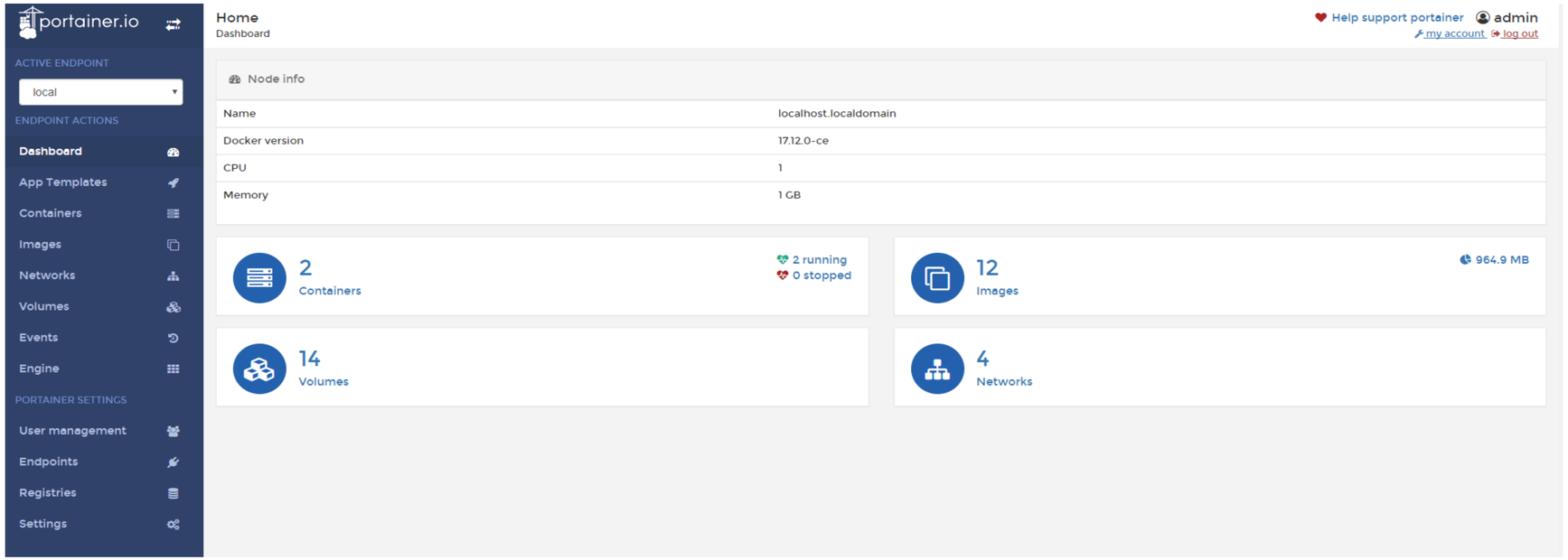
Portainer是一个开源、轻量级Docker管理用户界面，基于Docker API，可管理Docker主机或Swarm集群，支持最新版Docker和Swarm模式。

1、创建卷

```
# docker volume create portainer_data
```

2、创建Portainer容器

```
# docker run -d \
-p 9000:9000 \
-v /var/run/docker.sock:/var/run/docker.sock \
-v portainer_data:/data \
portainer/portainer
```



cAdvisor+InfluxDB+Grafana

Influxdb

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

cAdvisor

```
docker run -d \
--name=cadvisor \
--net monitor \
-p 8081:8080 \
--mount type=bind,src=/,dst=/rootfs,ro \
--mount type=bind,src=/var/run,dst=/var/run \
--mount type=bind,src=/sys,dst=/sys,ro \
--mount type=bind,src=/var/lib/docker/,dst=/var/lib/docker,ro \
google/cadvisor \
-storage_driver=influxdb \
-storage_driver_db=cadvisor \
-storage_driver_host=influxdb:8086
```

Grafana

```
docker run -d \
--name grafana \
--net monitor \
-p 3000:3000 \
grafana/grafana
```

谢谢

关注微信公众号：DevOps大咖

专注于分享运维开发领域技术及经验教训，包括Linux、Shell、Python、Docker、数据库、网站架构、集群等主流技术。每日一篇高质量文章，助你快速提升专业能力！