# Docker - 从零开始到操作

### 从零开始

介绍Docker的基本概念和命令，并给出简单使用示例和参考信息。

Docker - 基础讲义  
<http://www.cnblogs.com/anliven/p/6281373.html>

Docker - 常用基础命令  
<http://www.cnblogs.com/anliven/p/6217297.html>

### 安装与配置

分别介绍如何在CentOS、Windows和Ubuntu系统下安装与配置Docker。

Docker - 在CentOS 7中安装Docker  
<http://www.cnblogs.com/anliven/p/6202083.html>

Docker - 在Windows7中安装Docker  
<http://www.cnblogs.com/anliven/p/6804762.html>

Docker - 在Ubuntu16.04中安装Docker CE  
<http://www.cnblogs.com/anliven/p/6804501.html>

### 国内镜像和加速器

介绍如何利用Docker国内镜像和加速器，加快容器创建与部署。

Docker - 国内镜像的配置及使用  
<http://www.cnblogs.com/anliven/p/6218741.html>

Docker - 配置DaoCloud的Docker加速器  
<http://www.cnblogs.com/anliven/p/6217418.html>

### 镜像操作

Docker - 访问仓库  
<http://www.cnblogs.com/anliven/p/6783500.html>  
简单介绍仓库与注册服务器的概念，并以具体示例的形式说明常用操作。

Docker - 手动迁移镜像  
<http://www.cnblogs.com/anliven/p/6759585.html>  
在没有Docker Registry时,如何通过docker save和docker load命令完成镜像迁移。

Docker - 生成镜像  
<http://www.cnblogs.com/anliven/p/6733353.html>  
以具体示例的形式说明利用docker commit命令生成镜像。

Docker - 定制镜像  
<http://www.cnblogs.com/anliven/p/6828565.html>  
介绍了Dockerfile的定义、常用指令及docker build命令，并以具体示例的形式说明docker build命令执行Dockerfile指令的过程。

Docker- 创建支持SSH服务的容器镜像  
<http://www.cnblogs.com/anliven/p/6833276.html>  
以具体示例的形式说明如何利用Ubuntu16.04和CentOS7作为基础镜像创建支持SSH服务的容器镜像。

### 容器操作

Docker - 终端交互  
<http://www.cnblogs.com/anliven/p/6771214.html>  
以具体示例的形式说明进入和退出容器交互终端。

Docker - 导出导入容器  
<http://www.cnblogs.com/anliven/p/6771231.html>  
以具体示例的形式说明如何导入和导出容器快照文件。

Docker - 数据管理  
<http://www.cnblogs.com/anliven/p/6784113.html>  
以具体示例的形式说明管理容器中的数据。

Docker - 虚拟网桥  
<http://www.cnblogs.com/anliven/p/6804496.html>  
简单介绍了容器的网络模式，并说明主要讲解了虚拟网桥。

Docker - 访问容器  
<http://www.cnblogs.com/anliven/p/6799429.html>  
以具体示例的形式说明通过端口映射的方式，从外部访问容器中的应用。

Docker - 容器互联  
<http://www.cnblogs.com/anliven/p/6799443.html>  
以具体示例的形式说明如何通过docker run命令的--link参数可以让容器之间通过连接（linking）系统进行交互。

Docker - 容器直连  
<http://www.cnblogs.com/anliven/p/6810264.html>  
以具体示例的形式说明如何创建两个容器间的点到点链路。

### 其它

Docker - Tips  
<http://www.cnblogs.com/anliven/p/6481516.html>  
一些使用技巧和问题处理记录。

# 走一遍

systemctl stop firewalld.service #停止firewall

systemctl disable firewalld.service #禁止firewall开机启动

firewall-cmd --state #查看默认防火墙状态（关闭后显示notrunning，开启后显示running）

# Docker安装

yum -y install docker

### CentOS7升级最新版本Docker

首先升级一下系统包 （可以跳过）

$ sudo yum update

卸载旧版本的 Docker

$ sudo yum erase docker docker-common docker-client docker-compose

添加 Docker 的 yum 源

创建一个 vim -p /etc/yum.repos.d/docker.repo 文件，内容如下：

[dockerrepo]

name=Docker Repository

baseurl=https://yum.dockerproject.org/repo/main/centos/7/

enabled=1

gpgcheck=1

gpgkey=https://yum.dockerproject.org/gpg

安装新版本 Docker

$ sudo yum install -y docker-engine

安装完后设置为系统开机自动启动服务

$ sudo systemctl enable docker.service

启动服务

$ sudo systemctl start docker

测试一下

docker info

### 关闭SELinux

* 永久方法：修改/etc/selinux/config文件中设置SELINUX=disabled ，然后重启。
* 临时方法：执行setenforce 0命令设置SELinux成为permissive模式

sudo systemctl status firewalld.service

sudo systemctl stop firewalld.service

sudo systemctl disable firewalld.service

### **启动docker服务**

[root@CentOS-7 ~]# systemctl start docker.service

设置自启动docker服务

[root@CentOS-7 ~]# systemctl enable docker.service

Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

[root@CentOS-7 ~]# systemctl is-enabled docker.service

enabled

[root@CentOS-7 ~]#

### 设置docker代理(可以不使用代理)

# vim /etc/sysconfig/docker  
添加如下内容  
HTTP\_PROXY=http://10.144.1.10:8080  
HTTPS\_PROXY=https://10.144.1.10:8080  
FTP\_PROXY=ftp://10.144.1.10:8080

### 重启docker

# systemctl daemon-reload  
# systemctl restart docker

### 安装验证

运行官方镜像hello-world文件

# **docker run hello-world**

Unable to find image 'hello-world:latest' locally

Trying to pull repository docker.io/library/hello-world ...

latest: Pulling from docker.io/library/hello-world

c04b14da8d14: Pull complete

Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9

Status: Downloaded newer image for docker.io/hello-world:latest

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it

to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:

https://hub.docker.com

For more examples and ideas, visit:

https://docs.docker.com/engine/userguide/

#

# **docker images**

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/hello-world latest c54a2cc56cbb 5 months ago 1.848 kB

### 另外一种Docker代理设置方法：

修改docker.service文件  
# vim /usr/lib/systemd/system/docker.service  
在[Service]部分添加如下内容  
Environment="HTTP\_PROXY=http://10.144.1.10:8080"  
Environment="HTTPS\_PROXY=https://10.144.1.10:8080"  
Environment="FTP\_PROXY=ftp://10.144.1.10:8080"

或者，另外创建代理文件

# mkdir /etc/systemd/system/docker.service.d

# touch /etc/systemd/system/docker.service.d/proxy.conf

# vim /etc/systemd/system/docker.service.d/proxy.conf

增加以下内容  
[Service]  
Environment="HTTP\_PROXY=http://10.144.1.10:8080"  
Environment="HTTPS\_PROXY=https://10.144.1.10:8080"  
Environment="FTP\_PROXY=ftp://10.144.1.10:8080"

重启docker  
# systemctl daemon-reload  
# systemctl restart docker

检查docker环境变量是否加载

# systemctl show docker --property Environment

Environment=GOTRACEBACK=crash HTTP\_PROXY=http://10.144.1.10:8080 HTTPS\_PROXY=https://10.144.1.10:8080 FTP\_PROXY=ftp://10.144.1.10:8080

### Log - Install Docker in CentOS-7

[root@CentOS7 ~]# route add default gw 10.0.3.2

[root@CentOS7 ~]#

[root@CentOS7 ~]# yum -y install docker

[root@CentOS7 ~]# vim /etc/selinux/config

[root@CentOS7 ~]# cat /etc/selinux/config |grep "SELINUX="

# SELINUX= can take one of these three values:

SELINUX=disabled

[root@CentOS7 ~]#

[root@CentOS7 ~]# setenforce 0

[root@CentOS7 ~]#

[root@CentOS7 ~]# systemctl start docker.service

[root@CentOS7 ~]# systemctl enable docker.service

Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

[root@CentOS7 ~]# systemctl is-enabled docker.service

enabled

[root@CentOS7 ~]#

[root@CentOS7 ~]# sudo mkdir -p /etc/docker

[root@CentOS7 ~]# sudo tee /etc/docker/daemon.json <<-'EOF'

> {

> "registry-mirrors": ["https://t5t8q6wn.mirror.aliyuncs.com"]

> }

> EOF

{

"registry-mirrors": ["https://t5t8q6wn.mirror.aliyuncs.com"]

}

[root@CentOS7 ~]# systemctl daemon-reload

[root@CentOS7 ~]# systemctl restart docker

[root@CentOS7 ~]#

[root@CentOS7 ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

[root@CentOS7 ~]#

[root@CentOS7 ~]# docker run hello-world

Unable to find image 'hello-world:latest' locally

Trying to pull repository docker.io/library/hello-world ...

latest: Pulling from docker.io/library/hello-world

78445dd45222: Pull complete

Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it

to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://cloud.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/engine/userguide/

[root@CentOS7 ~]#

[root@CentOS7 ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/hello-world latest 48b5124b2768 3 months ago 1.84 kB

[root@CentOS7 ~]#

# 配置镜像加速器

针对Docker客户端版本大于 1.10.0 的用户

您可以通过修改daemon配置文件/etc/docker/daemon.json来使用加速器

sudo mkdir -p /etc/docker

sudo tee /etc/docker/daemon.json <<-'EOF'

{

"registry-mirrors": ["https://ubsqzgnc.mirror.aliyuncs.com"]

}

EOF

sudo systemctl daemon-reload

sudo systemctl restart docker

# 常用命令

### 帮助命令:

docker version

docker info

docker --help #linux中的命令man

### 镜像命令:

docker images #列出本地的镜像

docker images -a #显示所有镜像

docker images -q #只显示镜像的id

docker images -qa #查看所有id

docker images --digests #显示镜像摘要信息

docker images --no-trunc #显示完整的镜像信息

docker search 镜像XXX名称 #从dockerhub中查询docker镜像

docker search -s 30 tomcat #查询超过30颗星的tomcat镜像

docker pull 镜像XXX名称 #拉去指定的镜像

docker rmi 镜像XXX名称 #删除指定镜像

docker rmi -f 镜像id #删除单个镜像

docker rmi -f 镜像名称1:TAG 镜像名称2:TAG #删除多个

docker rmi -f $(docker images -qa) #删除全部

### 容器命令:

有镜像才能创建容器,这是一个根本前提

docker container ls #列出正在运行的容器

docker ps #列出正在运行的容器

docker container ls -a #列出所有的容器

docker ps -a #列出所有的容器

docker container start #启动容器

docker start #启动容器

docker container stop #停止容器

docker stop #停止容器

docker container restart #重启容器

docker restart #重启容器

docker container rm #删除容器

docker rm #删除容器

docker container logs #打印容器的标准输出

docker logs #打印容器的标准输出

#### 新建并启动容器

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

[OPTIONS]有哪些?

--name=容器新名字 #为容器指定一个名称;

-d: #后台运行容器,并返回容器id,也即启动守护式容器;

-i: #以交互模式运行容器,通常与-t同时使用;

-t: #为容器重新分配一个伪输入终端,通常与-i同时使用;

-P: #随机端口映射

-p: #指定端口映射,有以下四种格式

ip:hostPort:containerPort

ip::containerPort

hostPort:containerPort

containerPorte

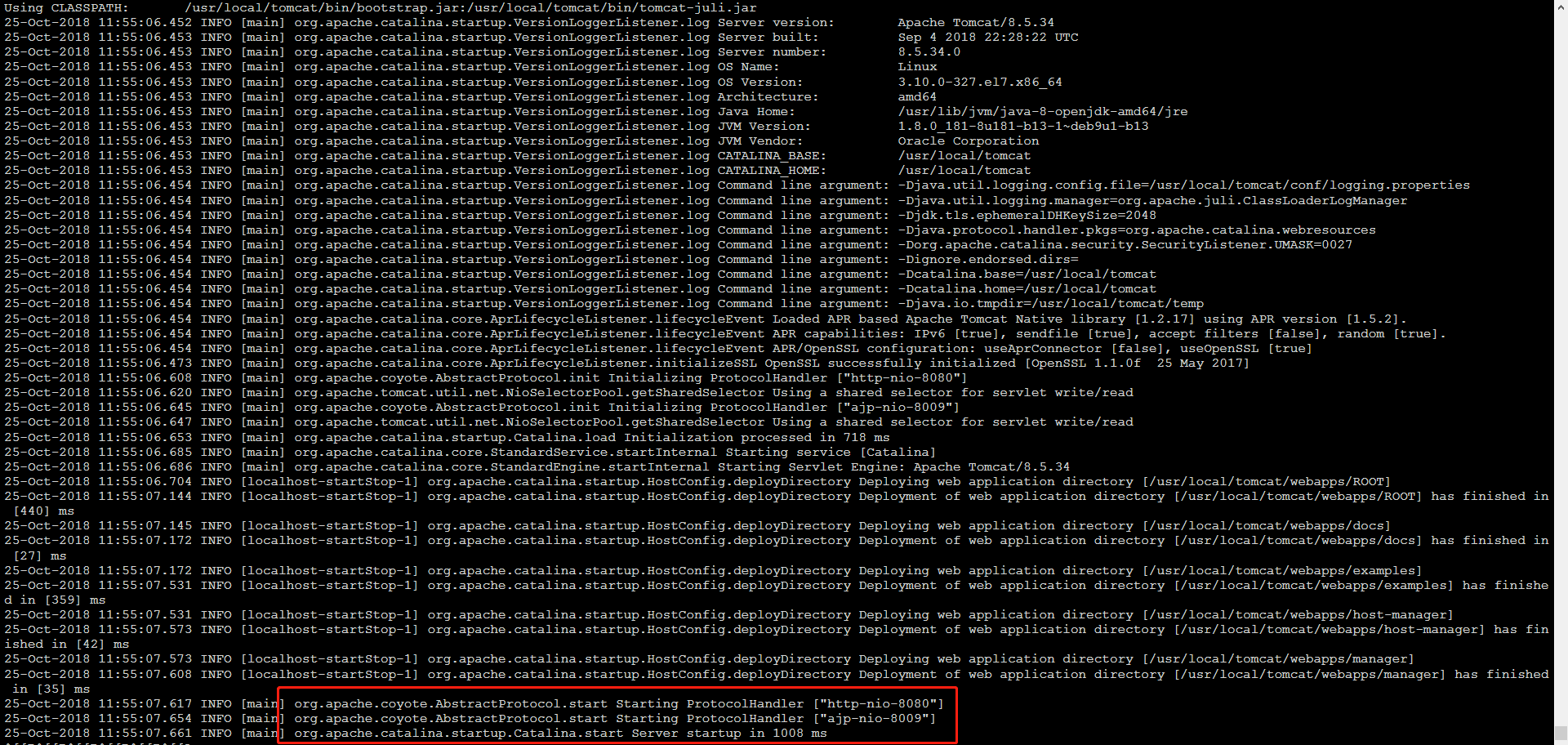
eg:

[root@localhost docker]# docker images

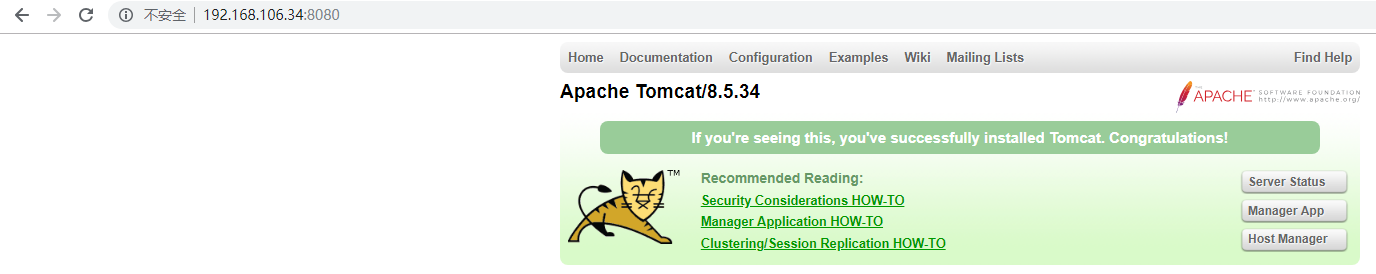
REPOSITORY TAG IMAGE ID CREATED SIZE

tomcat latest 05af71dd9251 8 days ago 463MB

[root@localhost docker]# docker run -p 8080:8080 -it 05af71dd9251



可以访问web页面



#### 列出正在运行的容器

docker ps

docker container ls

docker ps -a #列出当前所有正在运行的容器+历史上运行过的

docker ps -l #显示最近创建的容器

docker ps -n 3 #显示最近3个创建的容器

docker ps -q #静默模式,只显示容器编号

docker ps --no-trunc #不截断输出

#### 退出容器

方法一：如果要正常退出不关闭容器，请按Ctrl+P+Q进行退出容器

方法二：如果使用exit退出，那么在退出之后会关闭容器，可以使用下面的流程进行恢复

使用docker restart命令重启容器

使用docker attach命令进入容器

#### 启动容器

docker start 容器id/容器名称

#### 重启容器

docker restart 容器id/容器名称

[root@localhost ~]# docker ps -n 4

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

**8136cea88efd**  dbfc48660aeb "/bin/bash" 4 minutes ago Up 4 minutes 0.0.0.0:8888->80/tcp mynginx01

568dbdc55e69 dbfc48660aeb "nginx -g 'daemon ..." 10 minutes ago Exited (0) 8 minutes ago nginx\_web

952332210d35 hello-world "/hello" 10 minutes ago Exited (0) 10 minutes ago vibrant\_mccarthy

c1b2fde716bc dbfc48660aeb "nginx -g 'daemon ..." 11 minutes ago Exited (0) 10 minutes ago focused\_babbage

[root@localhost ~]#

[root@localhost ~]#

[root@localhost ~]#

[root@localhost ~]# docker start mynginx01

mynginx01

[root@localhost ~]# docker restart mynginx01

mynginx01

[root@localhost ~]# docker restart 8136cea88efd

8136cea88efd

[root@localhost ~]# docker start 8136cea88efd

8136cea88efd

[root@localhost ~]#

#### 停止容器

docker stop 容器id/容器名称

#### 强制停止容器

docker kill 容器id/容器名称

#### 删除已停止的容器

docker rm 容器id/容器名称

#### 一次性删除多个已停止的容器

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

docker ps -a -q | xargs docker rm

#### 启动守护式容器

docker run -d 容器id/容器名称

docker run -d centos

然后docker ps -a 进行查看,会发现容器已经退出

#### 查看容器日

docker logs -f -t --tail 容器id

-t是加入时间戳

-f是跟随最新的日志打印

--tail数字显示最后多少条

#### 查看容器内运行的进程

docker top 容器id

#### 查看容器内细节

docker inspect 容器id

#### 进入正在运行的容器并以命令行交互

docker exec -it 容器id bashShell

重新进入

docker attach 容器id

上述两个的区别:

attach:直接进入容器启动命令的终端,不会启动新的进程

exec:是在容器中打开新的终端,并且可以启动新的进程

#### 从容器内拷贝文件到主机上

docker cp 容器id:容器内路径 目的主机路径

将docker容器里的文件cp到虚拟机中

# Docker镜像

### Docker镜像是什么?

轻量级的，可执行的独立软件包。用来打包软件运行环境和基于运行环境开发的软件，包含所需的所有内容。

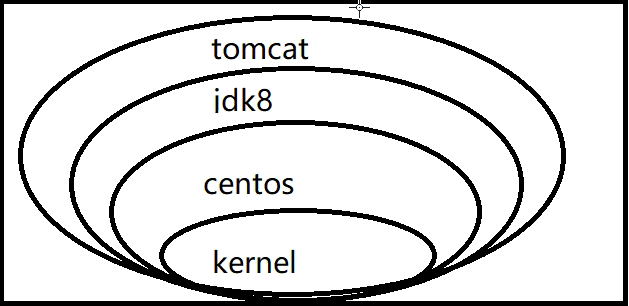
#### UnionFS联合文件系统

一层一层的叠加。

#### docker镜像加载原理

Docker镜像的最底层是bootfs。每一个容器就是精简版的linux系统，相当于linux的内核。

#### 分层的镜像：这就是为什么tomcat镜像这么大的原因



#### docker为什么要采用这种分层结构

最大的好处就是共享资源。

### Docker镜像的特点

Docker镜像都是只读的

当容器启动时，一个新的可写层被加载到镜像的顶部。这一层通常被称为“容器层”，“容器层”之下的都叫“镜像层”。

### Docker镜像commit

docker commit 提交容器副本使之成为一个容器镜像

docker commit -m=”描述信息” -a=“作者” 容器id 要创建的目标镜像名：TAG标签

# Docker容器数据卷

### 数据卷是什么

Data volumes是一个或者多个容器特别指定的目录,它区别于联合文件系统的(Union File System,或称UnionFS,通过底层的操作的文件系统,具有轻量级和快速的特性,Docker 容器也是使用联合文件系统去创建数据块的.)

Docker的理念：

1）、将运行环境和运行的代码打包成容器，运行可以伴随着容器，但是我们对容器中数据的要求可能是持久化的。

2）、容器之间希望有可能共享数据

docker中的数据如果不使用commit生成新的镜像，关闭容器或者删除容器时，容器中的数据自然就会被删除

为了能保存容器中的数据我们使用卷，有点类似Redis中的rdb和aof文件。

### 数据卷的作用（Data volumes）

作用：容器的持久化、容器间继承和共享数据

* 当容器被创建的同时,数据卷已经被初始化了.当镜像数据在制定的挂载点上时,存在的数据会复制到新的卷上.(但是,如果挂载的是主机路径不适用)
* 容器之间的数据卷可以重用和共享.
* 可以直接更改数据卷
* 镜像更新不会更改数据卷.
* 容器被删除，数据卷不受影响.

### 容器内添加数据卷

向容器中添加数据卷的两种方式：

#### 1、直接命令添加

1）命令：

docker run -it -v /宿主机绝对路径目录:/容器内目录 镜像名

#例如

docker run -it -v /myDataVolume:/containerDataVolume centos

#会在宿主机上生成myDataVolume目录

#并在docker容器centos中生成containerDataVolume目录，

#用于宿主机和容器的数据交互（就相当于将移动硬盘插入到笔记本中）

2）、查看数据卷是否挂在成功

#查看容器内部细节：以json串的格式显示容器内部细节。

docker inspect 容器id

3）、容器和宿主机之间数据共享

在容器中或者在宿主机中修改数据能够实现数据的共享！

4）、容器停止退出后，主机修改数据是否能同步？

会

5）、命令（带权限）

docker run -it -v /宿主机绝对路径目录:/容器内目录 镜像名

#例如

docker run -it -v /myDataVolume:/containerDataVolume：ro centos

#ro:相当于给容器设定权限，readOnly。只可以读不可以写！只容许宿主机操作，容器只能读！

#可以使用：docker inspect 容器id 查看

#### 2、DockerFile添加

DockerFile：镜像模板的描述文件！

使用DokcerFile添加容器数据卷

1）、根目录下创建myDocker文件夹并进入

2）、可在DockerFile中使用VOLUME指令来给镜像添加一个或多个数据卷

VOLUME["/dataVolumeContainer1","/dataVolumeContainer2","/dataVolumeContainer3"]

说明：处于移植和分享的考虑，使用 -v 主机目录：容器目录 这种方法不能直接在dockerFile中实现。

由于宿主机目录是依赖特定宿主机的，并不能保证所有的宿主机都存在所建的特定目录！

3）、File构建

vim dockerFile

#内容

# volume test

FROM centos

VOLUME["/dataVolumeContainer1","/dataVolumeContainer2"]

CMD echo "finished,----------success1"

CMD /bin/bash

4）、build后生成镜像

**docker build -f /mydocker/dockerfile -t zzyy/centos .**

#参数说明：

#-f：指明dockerFile的位置，如果就在dockerFile当前位置，可以不用写

#-t：给生成的镜像命名

# 最后是一个点，不能省略

5）docker run 新生成的镜像

dockerFile指明了新生成的容器中生成了两个数据卷：dataVolumeContainer1和dataVolumeContainer2

6）、通过上述步骤，容器内的卷目录地址已经知道，对应的主机目录在哪呢？

#如果不指明，会有一个默认的宿主机卷路径

docker inspect 容器id

注意：如果报错：Cannot open directroy

解决办法：在挂载目录下多加一个 -privileged=true参数即可

### 数据卷容器

#### 1、数据卷容器是什么？

命令的容器挂载数据卷，其他容器通过挂载这个（父容器）实现数据共享，挂载数据卷的容器，称之为数据卷容器。

#### 2、总体介绍

1）、以上一步新建的镜像zzyy/centos为模板并运行容器dc01/dc02/dc03

2）、他们已经具有容器卷：/dataVolumeContainer1,dataVolumeContainer2

#### 3、容器间传递共享（–volumes-form）

1）、先启动一个父容器(在dataVolumeContainer2中新增内容)

docker run -it --name dc01 zzyy/centos

2）、dc02、dc03继承自dc01

#dc02继承dc01

#并在2容器中添加内容

docker run -it --name dc02 --volumes-from dc01 zzyy/centos

#dc03继承dc01

#在3容器中添加内容

docker run -it --name dc03 --volumes-from dc01 zzyy/centos

3）、回到dc01、可以看到02/03都实现了数据的共享

4）、删除dc01，dc02修改后dc03是否可访问

5）、新建dc04继承dc03后在删除dc03，查看dc04容器中的数据

总结：容器之间的配置信息的传递，数据卷的声明周期已知延续到没有容器使用它为止。

### mysql创建数据卷

docker run --name mysql01 -v /mydocker/mysql:/mysql -it -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=root -d mysql

[root@localhost /]# docker exec -it mysql01 bash

root@04a062a078bb:/# mysql -uroot -proot

在本机/mydocker/mysql

和容器/mysql

下创建共享数据卷

此时,在本机和容器下创建文件或者在文件上写内容,两边都可以查看到并使共享的

[root@localhost mydocker]# docker run --name mysqltest -it -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=root -d mysql

037d75c0db9c3a7cf75dadca01e9466850fabd7cd198ef5025cb2d5fdf19e024

docker exec -it mysqltest bash

用inspect 可以查看docker上的容器的挂载信息

docker inspect mysqltest

|  |
| --- |
| "Mounts": [  {  "Type": "volume",  "Name": "12b1ee96d4f2015d9ed2531feafde66590d99fb988c9e218c0ca9a5f81b6d084",  "Source": "/var/lib/docker/volumes/12b1ee96d4f2015d9ed2531feafde66590d99fb988c9e218c0ca9a5f81b6d084/\_data",  "Destination": "/var/lib/mysql",  "Driver": "local",  "Mode": "",  "RW": true,  "Propagation": ""  }  ], |

source是本地持久化的地址

Destination是docker容器存放的地址

# 使用sqlyog连接Docker的容器mysql

### 一、在docker中启动mysql,定义端口号3306

[root@localhost ~]# docker run -p 3306:3306 --name mysql01 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql

### 二、执行命令，输入密码

[root@localhost ~]# docker exec -it mysql01 bash

root@863b69d78d4f:/# mysql -uroot -p

Enter password:

Welcome to the MySQL monitor.  Commands end with ; or \g.

Your MySQL connection id is 8

Server version: 8.0.12 MySQL Community Server - GPL

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

### 三、授权

mysql> GRANT ALL ON \*.\* TO 'root'@'%';

### 四、刷新权限

mysql> flush privileges;

### 五、更改加密规则

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'password' PASSWORD EXPIRE NEVER;

### 六、更新root用户密码（123456是我的密码，要替换成自己的）

mysql> ALTER USER 'root'@'%' IDENTIFIED WITH mysql\_native\_password BY 'root';

### 七、刷新权限

mysql> flush privileges;

OK，可以了，再次使用 SQLyog(Navicat) 连接数据库。

# DockerFile解析

### 1、DockerFile是什么

1）、DockerFile：用来构建Docker镜像的构建文件，是由一系列命令和参数构成的脚本。

2）、构建三步骤：

按指定语法编写dockerFile文件（上述的：dockerFile）

docker build：构建一个新的镜像

docker run

3）、文件是什么样？

以熟悉的Centos为例

FROM scratch ADD centos-7.4.1708-docker.tar.xz /   LABEL name="CentOS Base Image" \     vendor="CentOS" \     license="GPLv2" \     build-date="20170911"   CMD ["/bin/bash"]

### 2、DockerFile构建过程解析

1）、dockerFile内容基础知识

a、每条保留字指令都必须为大写字母且后面要跟随至少一个参数

b、指令按照从上到下顺序执行

c、# 表示注释

d、每条指令都会创建一个新的镜像层，并对镜像进行提交

2）Docker执行DockerFile的大致流程

a、docker从基础镜像运行一个容器（镜像的祖宗：scratch）

b、执行一条指定就对容器做出修改

c、执行类似docker commit的操作提交一个新的镜像层

d、docker再基于刚提交的镜像运行一个新容器

e、执行dockerFile中的下一条指令直到所有指令都执行完成

3）、总结

从应用软件的角度看，DockerFile、Docker镜像，Docker容器分别代表软件的三个不同阶段。

DockerFile是软件的原材料

Docker镜像是软件的交付品

Docker容器可以认定为软件的运行状态

DockerFile面向开发，镜像称为交付标准，容器设计部署与运维，三者缺一不可，合力充当docker体系的基石。

### 3、DockerFile体系结构（保留字指令）

FROM：基础镜像，当前镜像是基于哪个镜像的

MAINTAINER：镜像维护者的姓名和邮箱地址

RUN：构建容器时需要运行的命令（相当于运行java中的sout）

EXPOSE：当前容器对外暴露的端口

WORKDIR：指定创建容器后，终端默认登录进来的目录，一个落脚点

ENV：用来在构建镜像过程中设置环境变量

ADD：将宿主机目录下的文件拷贝进镜像且ADD命令会自动处理URL和解压tar压缩包

COPY：类似ADD，拷贝文件和目录到镜像中。

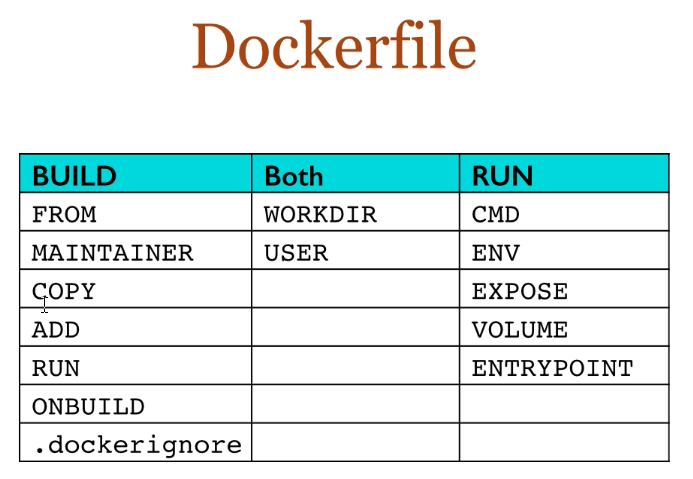
将从构建上下文目录中<原路径>的文件/目录复制到新的一层的镜像内的<目标路径>位置COPY src dest/COPY["src",""]

VOLUME : 容器数据卷，用于数据保存和持久化工作

CMD： 指定一个容器启动时要运行的命;Dockerfile中可以有多个CMD指令，但只有最后一个生效，CMD会被docker run之后的参数替换

ENTRYPOINT： 指定一个容器启动时要运行的命;ENTRYPOINT的目的和CMD一样，都是指定容器启动程序及参数

ONBUILD：当构建一个被继承的DockerFile时运行命令，父镜像在被子镜像继承后父镜像的onbulid被触发



### 4、案例

1）、Base镜像（Scratch）：docker hub中99%的镜像都是通过在base镜像中安装和配置需要的软件构建出来的

2）、自定义镜像

#### 案例一：

自定义mycentos具有如下特征：

登录后的默认路径

vim编辑器

查看网络配置支持ifconfig

步骤：

1、在mydocker文件夹下创建dockerfile文件

cd /

mkdir -p mydocker

vim -p /mydocker/dockerfile

2、写入如下dockerFile

FROM centos

MAINTAINER haiyang<441001065@qq.com>

ENV MYPATH /usr/local

WORKDIR $MYPATH

RUN yum -y install vim

RUN yum -y install net-tools

EXPOSE 80

CMD echo $MYPATH

CMD echo "success-------------ok

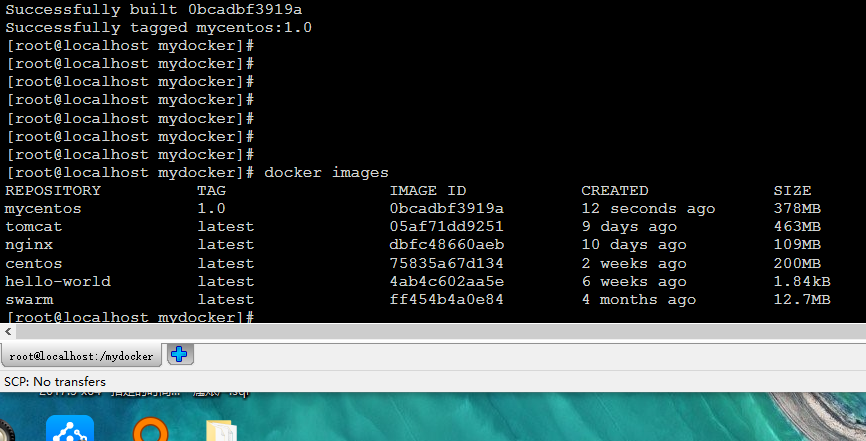
CMD /bin/bash

3、构建新的镜像

docker bulid -f -t 新镜像名：TAG .

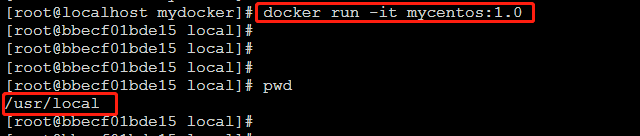
# 例

docker build -f /mydocker/dockerfile -t mycentos:1.0 .



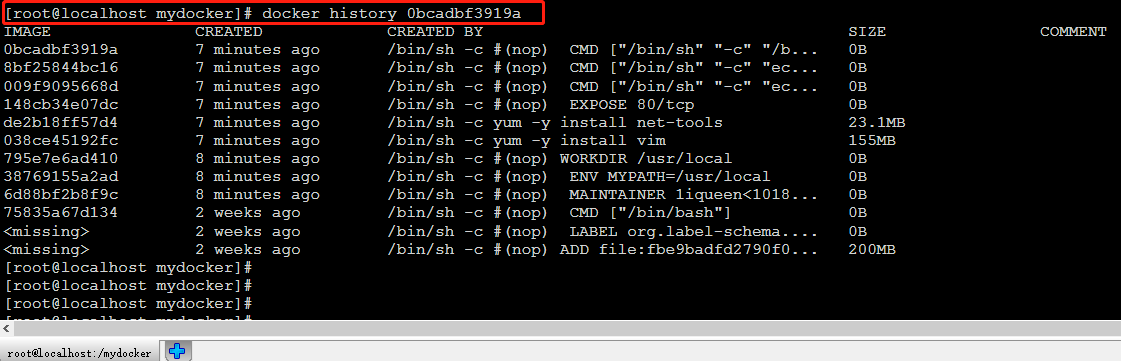
4、运行

docker run -it mycentos:1.0



5、列出镜像的变更历史

docker history 镜像id



在容器中自带vim和ifconfig的功能

#### 案例二：CMD与ENTRYPOINT镜像案例

它们都是指定一个容器启动时要运行的命令

**CMD：Dockerfile中可以有多个CMD命令，但只有最后一个生效，CMD会被docker run后面的参数覆盖**

CMD tomcat的演示：

#运行tomcat

docker run -it -p 8888:8080 tomcat

#加一条命令：就相当于加了CMD ls -l的指令

docker run -it -p 8888:8080 tomcat ls -l

#结果会不会启动tomcat也不会打日志，只会显示tomcat的目录结构,因为tomcat的WORKDIR在tomcat目录下

ENTRYPOINT：docker run之后的参数会被当做参数传递给ENTRYPOINT，之后形成新的命令组

**演示：制作CMD版查询ip的容器**

#curl命令的解释

#dockerFile3

FROM centos

RUN yum install -y crul

CMD ["crul","-s","http://ip.cn"]

#在此dockerFile基础之上，追加http头信息，加上-i会报错

docker run 镜像 -i

#dockerFile4

FROM centos

RUN yum install -y crul

ENTRYPOINT ["crul","-s","http://ip.cn"]

**#在此dockerFile基础之上，追加http头信息，加上-i不会报错**

docker run 镜像 -i

#因为CMD会替换上面的功能，只是最后一个CMD生效

#就相当于ENTRYPOINT ["crul","-s","-i","http://ip.cn"]

#### 案例三：ONBUILD

#在dockerFile中添加这个触发器，继承他的dockerFile在build时，会触发这个触发器

ONBUILD RUN echo "father images onbuild----------888"

#### 案例四：ADD和COPY的区别

1、第一步

mkdir -p /zzyyuse/myDockerFile/tomcat9

2、在上述目录上创建c.txt

3、将jdk和tomcat安装的压缩包copy进上一层目录

4、在/zzyyuse/myDockerFile/tomcat9目录下新建DockerFile文件

FROM centos

MAINTAINER haiyang<441001065@qq.com>

#把宿主机上的c.txt文件copy到容器的/usr/local/下,并重名了

COPY c.txt /usr/local/cincontainer.txt

#把java和tomcat添加到容器指定的位置，并解压

ADD apache-tomcat-7.0.70.tar.gz /usr/local/

ADD jdk-7u79-linux-x64.gz /usr/local/

#安装vim编辑器

RUN yum -y install vim

#设置工作访问的WORKDIR路径，登录落脚点

ENV MYPATH /usr/local

WORKDIR $MYPATH

#配置java和tomcat环境变量

ENV JAVA\_HOME /usr/local/jdk1.7.0\_79

ENV CLASSPATH $JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

ENV CATALINA\_HOME /usr/local/apache-tomcat-7.0.70

ENV CATALINA\_BASE /usr/local/apache-tomcat-7.0.70

ENV PATH $PATH:$JAVA\_HOME/bin:$CATALINA\_HOME/lib:$CATALINA\_HOME/bin

#容器运行的监听端口

EXPOSE 8080

#启动时运行tomcat

#ENTRYPOINT ["usr/local/apache-tomcat-7.0.70/bin/startup.sh"]

#CMD ["usr/local/apache-tomcat-7.0.70/bin/catalina.sh","run"]

CMD usr/local/apache-tomcat-7.0.70/bin/startup.sh && tail -f /usr/local/apache-tomcat-7.0.70/bin/logs/catalina.out

5、构建

docker build -f /zzyyuse/myDockerFile/tomcat9/DockerFile -t zzyytomcat9

6、run

#后台运行

#端口映射

#起名称

#建了两个卷

docker run -d -p 8888:8080 --name myt9 -v /zzyyuse/myDockerFile/tomcat9/test:/usr/local/apache-tomcat-7.0.70/webapps/test -v /zzyyuse/myDockerFile/tomcat9/tomcat9logs:/usr/local/apache-tomcat-7.0.70/logs --privilege=true zzyytomcat9

7、验证

1）、端口验证

2）、容器类的环境变量的验证

8、结合前述的容器卷将测试的web服务test发布

#在test下新建

cd /zzyyuse/myDockerFile/tomcat9/test

mkdir WEB-INFO

#在web-info下创建web.xml并添加内容

vim web.xml

#在test目录下创建a.jsp

vim a.jsp

# Docker应用程序的安装

### docker Tomcat安装

[root@localhost ~]# docker pull tomcat

[root@localhost ~]# docker images

REPOSITORY         TAG                 IMAGE ID           CREATED             SIZE

tomcat             latest             61205f6444f9        27 hours ago       467MB

#运行

[root@localhost ~]# docker run -d -p 8080:8080 --name mytomcat 61205f6444f9

#访问http://192.168.16.200:8080/

# 查看所有的 容器，然后根据 CONTAINER ID 启动或者停止

[root@localhost ~]# docker ps -a

#、 停止运行中的容器（第二次运行启动，已经映射过了）

docker stop 容器的id

#、启动容器

docker start 容器id

### docker Mysql安装

#下载

[root@localhost ~]# docker pull mysql:5.5

#查看下载的镜像

[root@localhost ~]# docker images

REPOSITORY         TAG                 IMAGE ID           CREATED             SIZE

mysql               5.5             a8a59477268d        5 weeks ago         445MB

#安装mysql ,设置编码 和 root的登陆密码

[root@localhost ~]# docker run --name mysql -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.5 --character-set-server=utf8mb4 --collation-server=utf8mb4\_unicode\_ci

​

​

### docker redis安装

docker redis安装

​

# docker images

​

# docker pull registry.docker-cn.com/library/redis                         //通过镜像下载

#或者 方法二：

[root@localhost ~]# docker pull redis

#启动

[root@localhost ~]# docker images

REPOSITORY         TAG                 IMAGE ID           CREATED             SIZE

tomcat             latest             61205f6444f9        27 hours ago       467MB

redis               latest             bfcb1f6df2db        5 weeks ago         107MB

[root@localhost ~]# docker run -d -p 6379:6379 --name myredis bfcb1f6df2db

#   -d 后台启动，-p暴露端口

​

### docker zookeeper安装

​

# docker images

# docker pull registry.docker-cn.com/library/zookeeper                         //通过镜像下载

#或者 方法二：

[root@localhost ~]# docker pull zookeeper

#启动

[root@localhost ~]# docker images

REPOSITORY                                 TAG                 IMAGE ID           CREATED             SIZE

registry.docker-cn.com/library/zookeeper   latest             2a8fecd00fba        4 days ago         146MB

[root@localhost ~]# docker run --name zookeeper -p 2181:2181 --restart always -d 2a8fecd00fba

#This image includes EXPOSE 2181 2888 3888 (the zookeeper client port, follower port, election port respectively), 选举等功能功能需要开启 2888 3888 端口

### docker rabbitmq安装

#第一步搜索

[root@localhost ~]# docker search rabbitmq

#第二步拉取（选择带有management的版本，有web界面管理功能）

[root@localhost ~]# docker pull rabbitmq:3.7.3-management

#查看镜像

[root@localhost ~]# docker images

REPOSITORY         TAG                 IMAGE ID           CREATED             SIZE

rabbitmq            3.7.3-management   2f415b0e9a6e        3 months ago       151MB

#运行

[root@localhost ~]# docker run -d -p 5672:5672 -p 15672:15672 --name myrabbitmq 2f415b0e9a6e

#访问 http://192.168.16.200:15672/#/   guest:guest 登陆

### docker elasticsearch安装（安装2.4.6版本的）

docker search elasticsearch   //搜索

docker pull registry.docker-cn.com/library/elastiscearch   //下载

或者

[root@localhost ~]# docker pull elasticsearch:2.4.6 下载

#docker images

docker run -e ES\_JAVA\_OPTS="-Xms256m -Xmx256m" -d -p 9200:9200 -p 9300:9300  --name ES01 镜像Id

#默认占用2个G内存，限制内存大小，初始大小Xms256m ，最大Xmx256m

​

​

#打开浏览器访问：http://ip:9200

#返回JSON就是安装成功

#官方文档   https://www.elastic.co/cn/products/elasticsearch

#https://www.elastic.co/guide/cn/elasticsearch/guide/current/index.html

# 基础入门-->面向文档-->

# 具体介绍文档 https://www.elastic.co/guide/cn/elasticsearch/guide/current/\_retrieving\_a\_document.html

​

​

#指令

# get   操作获取   GET /megacorp/employee/1

# put   存储

# head   检查文档是否存在

# delete 删除

#GET /megacorp/employee/\_search 搜索所有员工

#文档：https://www.elastic.co/guide/cn/elasticsearch/guide/current/\_search\_lite.html

​

#GET /megacorp/employee/\_search?q=last\_name:Smith

#搜索last\_name=Smith的员工

#返回的json找那个score：质量分数

​

​

#查询表达式：

https://www.elastic.co/guide/cn/elasticsearch/guide/current/\_search\_with\_query\_dsl.html

#查询last\_name=smith ；age>30的员工信息

GET /megacorp/employee/\_search

{

   "query" : {

       "bool": {

           "must": {

               "match" : {

                   "last\_name" : "smith"

              }

          },

           "filter": {

               "range" : {

                   "age" : { "gt" : 30 }

              }

          }

      }

  }

}

#全文搜索，搜索rock or climbing ,返回\_score 对应质量得分

GET /megacorp/employee/\_search

{

   "query" : {

       "match" : {

           "about" : "rock climbing"

      }

  }

}

### docker nginx安装

[root@localhost ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

tomcat latest 05af71dd9251 8 days ago 463MB

nginx latest dbfc48660aeb 9 days ago 109MB

hello-world latest 4ab4c602aa5e 6 weeks ago 1.84kB

[root@localhost ~]#

[root@localhost ~]#

[root@localhost ~]#

[root@localhost ~]# docker run -it --name mynginx01 -p 8888:80 dbfc48660aeb /bin/bash

root@8136cea88efd:/# /etc/init.d/nginx start

root@8136cea88efd:/# 192.168.106.61 - - [25/Oct/2018:12:31:19 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.

root@8136cea88efd:/# 2018/10/25 12:31:19 [error] 14#14: \*1 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 192.168.106.61, server: localhost, request: "GET

/favicon.ico HTTP/1.1", host: "192.168.106.34:8888", referrer: "http://192.168.106.34:8888/"

192.168.106.61 - - [25/Oct/2018:12:31:19 +0000] "GET /favicon.ico HTTP/1.1" 404 555 "http://192.168.106.34:8888/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C

hrome/69.0.3497.100 Safari/537.36" "-"

root@8136cea88efd:/#

启动nginx并访问web页面

http://192.168.106.34:8888/

