# Docker

## 核心概念

### Docker镜像(Image)

Docker镜像类似于虚拟机镜像，可以将它理解为一个只读的模板。例如，一个镜像可以包含一个基本的操作系统环境，里面仅安装了Apache应用程序（或用户需要的其他软件）。可以把它称为一个Apache镜像。

镜像是创建Docker容器的基础。通过版本管理和增量的文件系统，Docker提供了一套十分简单的机制来创建和更新现有的镜像，用户甚至可以从网上下载一个已经做好的应用镜像，并直接使用。

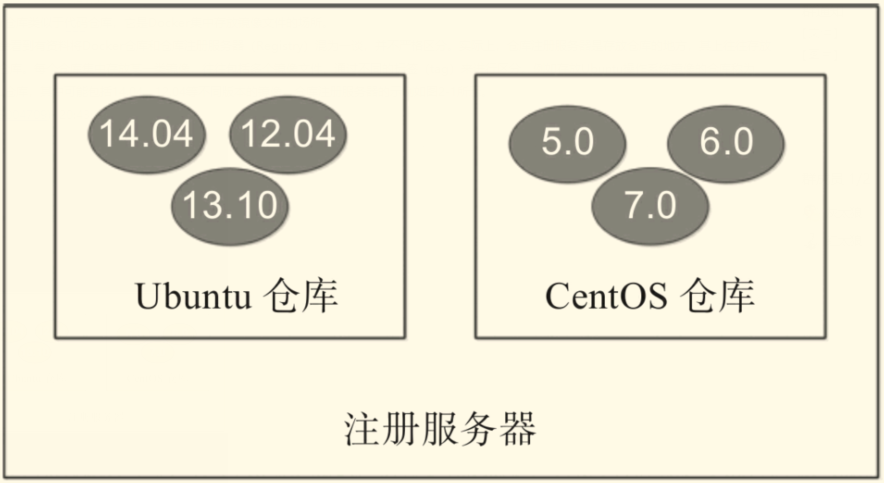
### 容器(Container)

Docker容器类似于一个轻量级的沙箱，Docker利用容器来运行和隔离应用。容器是从镜像创建的应用运行实例。可以将其启动、开始、停止、删除，而这些容器都是彼此相互隔离的、互不可见的。

可以把容器看做是一个简易版的Linux系统环境（包括root用户权限、进程空间、用户空间和网络空间等）以及运行在其中的应用程序打包而成的盒子。

### 仓库(Repository)

Docker仓库类似于代码仓库，它是Docker集中存放镜像文件的场所。  
有时候会看到有资料将Docker仓库和仓库注册服务器（Registry）混为一谈，并不严格区分。实际上，仓库注册服务器是存放仓库的地方，其上往往存放着多个仓库。每个仓库集中存放某一类镜像，往往包括多个镜像文件，通过不同的标签（tag）来进行区分。例如存放Ubuntu操作系统镜像的仓库称为Ubuntu仓库，其中可能包括14.04、12.04等不同版本的镜像。



根据所存储的镜像公开分享与否，Docker仓库可以分为公开仓库（Public）和私有仓库（Private）两种形式。目前，最大的公开仓库是官方提供的Docker Hub，其中存放了数量庞大的镜像供用户下载。国内不少云服务提供商（如时速云、阿里云等）也提供了仓库的本地源，可以提供稳定的国内访问。

当然，用户如果不希望公开分享自己的镜像文件，Docker也支持用户在本地网络内创建一个只能自己访问的私有仓库。当用户创建了自己的镜像之后就可以使用push命令将它上传到指定的公有或者私有仓库。这样用户下次在另外一台机器上使用该镜像时，只需要将其从仓库上pull下来就可以了。

## Docker安装

Docker在主流的操作系统和云平台上都可以使用，包括Linux操作系统（如Ubuntu、Debian、CentOS、Redhat等）、MacOS操作系统和Windows操作系统，以及AWS等云平台。

### Ubuntu环境下安装Docker

#### 系统要求

Docker目前只能运行在64位平台上，并且要求内核版本不低于3.10，实际上内核越新越好，过低的内核版本容易造成功能不稳定。

用户可以通过如下命令检查自己的内核版本详细信息：

$ uname -a

Linux Host 3.16.0-43-generic #58~14.04.1-Ubuntu SMP Mon Jun 22 10:21:20 UTC

2015 x86\_64 x86\_64 x86\_64 GNU/Linux或者：

$ cat /proc/version

Linux version 3.16.0-43-generic (buildd@brownie) (gcc version 4.8.2 (Ubuntu

4.8.2-19ubuntu1) ) #58~14.04.1-Ubuntu SMP Mon Jun 22 10:21:20 UTC 2015

Docker目前支持的最低Ubuntu版本为12.04 LTS，但实际上从稳定性上考虑，推荐至少使用14.04 LTS版本。

如果使用12.04 LTS版本，首先要更新系统内核和安装可能需要的软件包，包括：

linux-image-generic-lts-trusty（必备）

linux-headers-generic-lts-trusty（必备）

xserver-xorg-lts-trusty（带图形界面时必备）

libgl1-mesa-glx-lts-trusty（带图形界面时必备）

另外，为了让Docker使用aufs存储，推荐安装linux-image-extra软件包。

$ sudo apt-get install -y linux-image-extra-$(uname -r)

注意

Ubuntu发行版中，LTS（Long-Term-Support）意味着更稳定的功能和更长期（目前为5年）的升级支持，生产环境中尽量使用LTS版本。

#### 添加镜像源

首先需要安装apt-transport-https包支持HTTPS协议的源：

$ sudo apt-get install -y apt-transport-https

添加源的gpg密钥：

$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys

58118E89F3A912897C070ADBF76221572C52609D

获取当前操作系统的代号：

$ lsb\_release -c

Codename: trusty

一般情况下，12.04（LTS）代号为precise，14.04（LTS）代号为trusty，15.04代号为vivid，15.10代号为wily。这里获取的代号为trusty。

接下来就可以添加Docker的官方apt软件源了。通过下面的命令创建/etc/apt/sources.list.d/docker.list文件，并写入源的地址内容。非trusty版本的系统注意修改为自己对应的代号：

$ sudo cat <<EOF > /etc/apt/sources.list.d/docker.list

deb https://apt.dockerproject.org/repo ubuntu-trusty main

EOF

添加成功后，更新apt软件包缓存：

$ sudo apt-get update

#### 开始安装Docker

在成功添加源之后，就可以安装最新版本的Docker了，软件包名称为docker-engine：

$ sudo apt-get install -y docker-engine

如果系统中存在较旧版本的Docker（lxc-docker），会提示是否先删除，选择“是”即可。

除了基于手动添加软件源的方式，也可以使用官方提供的脚本来自动化安装Docker：

$ sudo curl -sSL https://get.docker.com/ | sh

安装成功后，启动docker服务：$ sudo service docker start

### CentOS环境下安装Docker

系统的要求与Ubuntu情况下类似：64位操作系统，内核版本至少为3.10。

Docker目前支持CentOS 6.5及以后的版本，推荐使用CentOS 7系统。

首先，也是要添加yum软件源：

$ sudo tee /etc/yum.repos.d/docker.repo <<-'EOF'

[dockerrepo]

name=Docker Repository

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

enabled=1

gpgcheck=1

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

EOF

之后更新yum软件源缓存，并安装docker-engine：

$ sudo yum update

$ sudo yum install -y docker-engine

对于CentOS 7系统，CentOS-Extras源中已内置Docker，如果已经配置了CentOS-Extras源，可以直接通过上面的yum命令进行安装。

### 通过脚本安装

用户还可以使用官方提供的shell脚本来在Linux系统（目前支持Ubuntu、Debian、Oracleserver、Fedora、Centos、OpenSuse、Gentoo等常见发行版）上安装Docker的最新正式版本，该脚本会自动检测系统信息并进行相应配置：

$ curl -fsSL https://get.docker.com/ | sh

或者：

$ wget -qO- https://get.docker.com/ | sh

如果想尝鲜使用最新功能，可以使用下面的脚本来安装预发布版本。但要注意，预发布版本往往意味着功能还不够稳定，不要在生产环境中使用：

$ curl -fsSL https://test.docker.com/ | sh

另外，也可以从github.com/docker/docker/releases找到所有的发行版本信息和二进制包，自行下载使用。

### windows下安装Docker

目前Docker可以通过虚拟机方式来支持Windows 7.1和8，只要平台CPU支持硬件虚拟化特性即可。如果无法确定自己计算机的CPU是否支持该特性也无需担心，实际上，目前市面上主流的CPU都早已支持了硬件虚拟化特性。

对于Windows 10用户，Docker官方提供了原生虚拟化应用Docker for Windows。详情见：https://docs.docker.com/windows/step\_one/。目前国内Windows 7还是主导地位的版本，所以下面主要讲解如何在Windows 7环境下安装Docker环境。

由于Docker引擎使用了Linux内核特性，所以如果要在Windows 10之外的Windows上运行，需要额外使用一个虚拟机来提供Linux支持。这里推荐使用Boot2Docker工具，它会首先安装一个经过加工与配置的轻量级虚拟机，然后在其中运行Docker。主要步骤如下：

首先，从https://docs.docker.com/installation/windows/下载最新官方Docker for Windows Installer。双击打开Installer。这个过程将安装VirtualBox、MSYS-git、boot2docker Linux ISO镜像，以及Boot2Docker管理工具。

最后，打开桌面的Boot2Docker Start程序，或者Program Files\Boot2Docker for Windows。此初始化脚本在第一次运行时需要输入一个SSH Key Passphrase（用于SSH密钥生成的口令）。读者可以自行设定，也可以直接按回车键跳过此设定。

此时Boot2Docker Start程序将连接至虚拟机中的Shell会话，Docker已经运行起来了！

### 验证Docker安装

打开终端控制器或其他系统命令行，执行docker version命令。

$ docker version

Client:

Version: 1.12.0

API version: 1.24

Go version: go1.6.3

Git commit: 8eab29e

Built: Thu Jul 28 21:15:28 2016

OS/Arch: darwin/amd64

Server:

Version: 1.12.0

API version: 1.24

Go version: go1.6.3

Git commit: 8eab29e

Built: Thu Jul 28 21:15:28 2016

OS/Arch: linux/amd64

如果我们看到Client和Server均有输出，则说明Docker已经正常启动。如果我们看到报错：“Cannot connect to the Docker daemon.Is the docker daemon running on this host？”，则说明Docker没有启动或启动失败。

### 启动一个Nginx容器

docker run -d -p 80:80 --name webserver nginx

### 查看当前运行的容器

docker ps

## 配置Docker服务

为了避免每次使用docker命令都要用特权身份，可以将当前用户加入安装中自动创建的docker用户组：

$ sudo usermod -aG docker USER\_NAME

用户更新组信息后，退出并重新登录后即可生效。

另外，Docker服务支持多种启动参数。以Ubuntu 14.04系统为例，Docker服务的默认配置文件为/etc/default/docker，可以通过修改其中的DOCKER\_OPTS来修改服务启动的参数，例如，下一行代码让Docker服务可以通过本地2375端口接收来自外部的请求：

DOCKER\_OPTS="$DOCKER\_OPTS -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock"

修改之后，通过service命令来重启Docker服务：

$ sudo service docker restart

一般情况下，Docker服务的管理脚本为/etc/init.d/docker，通过查看其中的内容，发现主要是将Docker进程的id写入/var/run/docker.pid文件，以及通过ulimit调整系统的资源限制。

如果是通过较新的upstart工具来管理服务，则管理服务配置文件在/etc/init/docker.conf。

另外，对于CentOS、Redhat等系统，服务可能是通过systemd来管理，与此略有不同，可以查阅systemd相关手册。

例如，需要通过systemctl命令来管理Docker服务：

$ sudo systemctl start docker.service

此外，如果服务工作不正常，可以通过查看Docker服务的日志信息来确定问题，例如在Ubuntu系统上日志文件可能为/var/log/upstart/docker.log：

$ sudo tail /var/log/upstart/docker.log

每次重启Docker服务后，可以通过查看Docker版本信息，确保服务已经正常运行：

$ docker version

Client:

Version: 1.12.0

API version: 1.24

Go version: go1.6.3

Git commit: 8eab29e

Built: Thu Jul 28 21:15:28 2016

OS/Arch: darwin/amd64

Server:

Version: 1.12.0

API version: 1.24

Go version: go1.6.3

Git commit: 8eab29e

Built: Thu Jul 28 21:15:28

## Docker Toolbox

docker-machine upgrade

docker-machine [start|stop|restart]

应该使用docker-machine ssh xxx 登录到虚机上执行docker run命令

docker-machine ip

boot2docker用户和密码

| **用户** | **密码** | **进入方式** |
| --- | --- | --- |
| docker | tcuser | ssh |
| root |  | command：sudo -i (docker用户下执行) |

## dockerfile

### 基本示例

# This is a commit

FROM alpine:3.2

MAINTAINER Jasper Xu <sorex@163.com>

RUN apk update && apk add socat && rm -r /var/cache/

RUN chmod -R 755 /data

ADD execfile /etc/execfile

ADD data/\* /etc/data/

COPY xxx.tgz /home/xxx.tgz

RUN chmod 755 /etc/execfile

RUN ["/etc/execfile", "arg1", "arg1"]

ENV abc=hello

VOLUME ["/data"]

EXPOSE 8080

WORKDIR /data/tools

CMD service nginx start

CMD echo "Hello docker!"

### FROM

必须为第一个命令，指定基础镜像

FROM <image>

FROM <image>:<tag>

FROM <image>@<digest>

FROM mysql:5.6

### MAINTAINER

维护者信息

MAINTAINER <name>

MAINTAINER Jasper Xu

MAINTAINER sorex@163.com

MAINTAINER Jasper Xu <sorex@163.com>

### LABEL

给镜像添加信息。使用docker inspect可查看镜像的相关信息

LABEL "com.example.vendor"="ACME Incorporated"

LABEL com.example.label-with-value="foo"

LABEL version="1.0"

LABEL description="This text illustrates \

that label-values can span multiple lines."

### RUN

构建镜像时执行的命令

# 由shell启动，Linux默认为`/bin/sh -c`，Windows默认为`cmd /S /C`

RUN <command>

# 运行可执行文件

RUN ["executable", "param1", "param2"]

RUN apk update

RUN ["/etc/execfile", "arg1", "arg1"]

### ADD

将本地文件添加到容器中，identity, gzip, bzip2，xz，tar.gz，tgz等类型的文件将被添加tar -x命令，进行解压

ADD <src>... <dest>

ADD ["<src>",... "<dest>"] 用于支持包含空格的路径

ADD hom\* /mydir/ # 添加所有以"hom"开头的文件

ADD hom?.txt /mydir/ # ? 替代一个单字符,例如："home.txt"

ADD test relativeDir/ # 添加 "test" 到 `WORKDIR`/relativeDir/

ADD test /absoluteDir/ # 添加 "test" 到 /absoluteDir/

### COPY

同ADD，只是不会解压文件。

### CMD

CMD ["executable","param1","param2"] (执行可执行文件，优先)

CMD ["param1","param2"] (设置了ENTRYPOINT，则直接调用ENTRYPOINT添加参数)

### ENTRYPOINT

ENTRYPOINT ["executable", "param1", "param2"] (可执行文件, 优先)

ENTRYPOINT command param1 param2 (shell内部命令)

对于一个 docker 镜像，我们可以这么来理解  ENTRYPOINT 与 CMD 的关系

如果没有定义 ENTRYPOINT， CMD 将作为它的 ENTRYPOINT

定义了 ENTRYPOINT 的话，CMD 只为 ENTRYPOINT 提供参数

CMD 可由 docker run <image> 后的命令覆盖，同时覆盖参数

对于 #1 和  #2 更精致的理解是容器运行的最终入口由 ENTRYPOINT 和实际的 CMD 拼接而成。ENTRYPOINT 和 CMD 需转换为实际镜像中的 exec 格式来拼接，合并后的第一个元素是命令，其余是它的参数。

### ENV

设置环境变量

ENV <key> <value>

ENV <key>=<value> ...

ENV myName="John Doe" myDog=Rex\ The\ Dog \

myCat=fluffy

等同于

ENV myName John Doe

ENV myDog Rex The Dog

ENV myCat fluffy

### EXPOSE

指定于外界交互的端口，在容器启动时用-p传递参数，例如-p 3307:3306将容器内的3306绑定到本机的3307

EXPOSE <port> [<port>...]

EXPOSE 80 443

EXPOSE 8080

### VOLUME

用于指定持久化目录，在容器启动时用-v传递参数，例如-v ~/opt/data/mysql:/var/lib/mysql将本机的~/opt/data/mysql和容器内的/var/lib/mysql做持久化关联  
容器启动时会加载，容器关闭后会回写。

VOLUME ["/data"]

VOLUME ["/data"]

VOLUME ["/var/www", "/var/log/apache2", "/etc/apache2"]

### WORKDIR

工作目录

WORKDIR /path/to/workdir

WORKDIR /a (这时工作目录为/a)

WORKDIR b (这时工作目录为/a/b)

WORKDIR c (这时工作目录为/a/b/c)

### USER

用于设定容器的运行用户名或UID,USER 123或USER git。  
指定运行容器时的用户名或UID，后续的 RUN 也会使用指定用户。

当服务不需要管理员权限时，可以通过该命令指定运行用户。并且可以在之前创建所需要的用户，例如： RUN groupadd -r postgres && useradd -r -g postgres postgres 。要临时获取管理员权限可以使用 gosu ，而不推荐 sudo 。

### ARG

由外部启动时必须传入的参数，在容器启动时用--build-arg传递参数  
指定于外界交互的端口，在容器启动时用-p传递参数，例如--build-arg CONT\_IMG\_VER=v2.0.1

FROM ubuntu

2 ARG CONT\_IMG\_VER

3 ENV CONT\_IMG\_VER ${CONT\_IMG\_VER:-v1.0.0}

4 RUN echo $CONT\_IMG\_VER

有些默认参数，无需指定，也不用使用``传递，可直接传参  
HTTP\_PROXY,http\_proxy,HTTPS\_PROXY,https\_proxy,FTP\_PROXY,ftp\_proxy,NO\_PROXY,no\_proxy

## Docker命令

### docker ps

列出容器

### docker images

列出本地镜像

### docker run

创建一个新的容器并运行一个命令

语法

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

OPTIONS说明：

**-a stdin:** 指定标准输入输出内容类型，可选 STDIN/STDOUT/STDERR 三项；

**-d:** 后台运行容器，并返回容器ID；

**-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；

**--name="nginx-lb":** 为容器指定一个名称；

**--dns 8.8.8.8:** 指定容器使用的DNS服务器，默认和宿主一致；

**--dns-search example.com:** 指定容器DNS搜索域名，默认和宿主一致；

**-h "mars":** 指定容器的hostname；

**-e username="ritchie":** 设置环境变量；

**--env-file=[]:** 从指定文件读入环境变量；

**--cpuset="0-2" or --cpuset="0,1,2":** 绑定容器到指定CPU运行；

**-m :**设置容器使用内存最大值；

**--net="bridge":** 指定容器的网络连接类型，支持 bridge/host/none/container: 四种类型；

**--link=[]:** 添加链接到另一个容器；

通过link的方式创建容器，我们可以使用被Link容器的别名进行访问，而不是通过IP，解除了对IP的依赖

--link myzookeeper:localhost

**--expose=[]:** 开放一个端口或一组端口；

实例

使用docker镜像nginx:latest以后台模式启动一个容器,并将容器命名为mynginx。

docker run --name mynginx -d nginx:latest

使用镜像nginx:latest以后台模式启动一个容器,并将容器的80端口映射到主机随机端口。

docker run -P -d nginx:latest

使用镜像nginx:latest以后台模式启动一个容器,将容器的80端口映射到主机的80端口,主机的目录/data映射到容器的/data。

docker run -p 80:80 -v /data:/data -d nginx:latest

使用镜像nginx:latest以交互模式启动一个容器,在容器内执行/bin/bash命令。

runoob@runoob:~$ docker run -it nginx:latest /bin/bash

root@b8573233d675:/#

### docker pull

docker pull hub-mirror.c.163.com/library/nginx

### ****docker attach****

连接到正在运行中的容器。

### docker cp

docker cp nginx:/etc/nginx/nginx.conf /config/nginx.conf

### docker exec

docker exec -it mynginx /bin/bash