**[Docker概述](http://www.dockerinfo.net/document)**

## 在任何地方开发、部署和运行任何应用

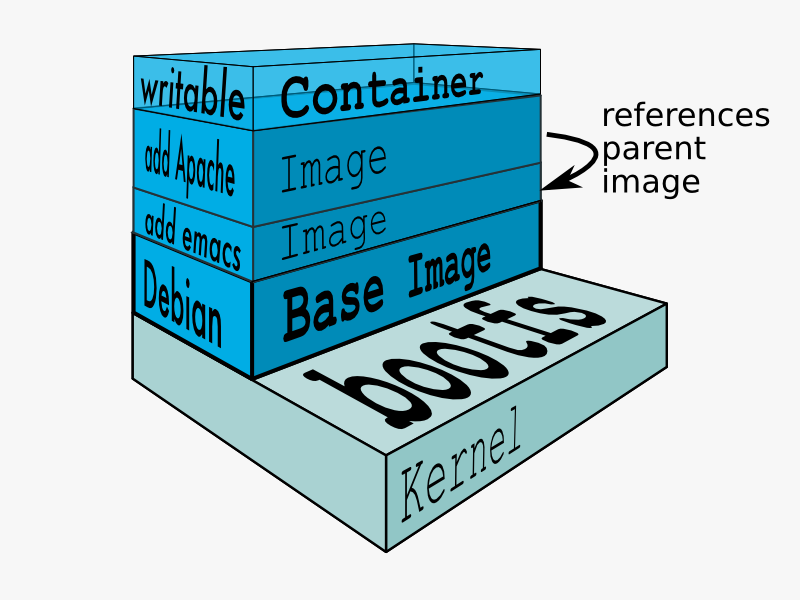
### 为什么选择Docker

### 快速交付应用程序

### 快速构建 轻松管理

# [Image镜像](http://www.dockerinfo.net/image%e9%95%9c%e5%83%8f)

所有的变更都发生顶层的可写层，而下层的原始的只读镜像文件并未变化。由于镜像不 可写，所以镜像是无状态的。



## 获取镜像

docker pull ubuntu:12.04

该命令实际上相当于 docker pull registry.hub.docker.com/ubuntu:12.04 命令，即从注册服务器registry.hub.docker.com 中的 ubuntu 仓库来下载标记为 12.04 的镜像。

## 列出本地镜像

docker images

## 创建镜像

### 修改已有镜像

### 利用 Dockerfile 来创建镜像

Dockerfile 基本的语法是

* 使用#来注释
* FROM 指令告诉 Docker 使用哪个镜像作为基础
* 接着是维护者的信息
* RUN开头的指令会在创建中运行，比如安装一个软件包，在这里使用 apt-get 来安装了一些软件

编写完成 Dockerfile 后可以使用 docker build 来生成镜像。

docker build -t="ouruser/sinatra:v2" .

\*注意一个镜像不能超过 127 层

还可以利用 ADD 命令复制本地文件到镜像；

用 EXPOSE 命令来向外部开放端口；

用 CMD 命令来描述容器启动后运行的程序等。

用 docker tag 命令来修改镜像的标签

### 从本地文件系统导入

### 上传镜像 docker push

## 存出和载入镜像

## 移除本地镜像

# [Docker容器](http://www.dockerinfo.net/docker%e5%ae%b9%e5%99%a8-2)

## 启动容器

docker run

docker run ubuntu:14.04 /bin/echo 'Hello world'

启动一个 bash 终端，允许用户进行交互, 在交互模式下，用户可以通过所创建的终端来输入命令

docker run -t -i ubuntu:14.04 /bin/bash

-t 选项让Docker分配一个伪终端（pseudo-tty）并绑定到容器的标准输入上， -i 则让容器的标准输入保持打开。

docker start

## 守护态运行

通过添加 -d 参数让 Docker 容器在后台以守护态（Daemonized）形式运行

docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 1; done"

## 终止容器

docker stop

## 进入容器

docker attach

### nsenter 命令

## 导出和导入容器

## 删除容器

# [数据卷volumes](http://www.dockerinfo.net/%e6%95%b0%e6%8d%ae%e5%8d%b7volumes)

数据卷是一个可供一个或多个容器使用的特殊目录，它绕过 UFS，可以提供很多有用的特性：

* 数据卷可以在容器之间共享和重用
* 对数据卷的修改会立马生效
* 对数据卷的更新，不会影响镜像
* 卷会一直存在，直到没有容器使用

### 创建一个数据卷

使用 -v 标记来创建一个数据卷并挂载到容器里。在一次 run 中多次使用可以挂载多个数据卷。

创建一个 web 容器，并加载一个数据卷到容器的 /webapp 目录

docker run -d -P --name web -v /webapp training/webapp python app.py

也可以在 Dockerfile 中使用 VOLUME 来添加一个或者多个新的卷到由该镜像创建的任意容器

### 挂载一个主机目录作为数据卷

-v 标记也可以指定挂载一个本地主机的目录到容器中去

docker run -d -P --name web -v /src/webapp:/opt/webapp training/webapp python app.py

加载主机的 /src/webapp 目录到容器的 /opt/webapp 目录, 本地目录的路径必须是绝对路径，如果目录不存在 Docker 会自动为你创建它。

Dockerfile 中不支持这种用法，这是因为 Dockerfile 是为了移植和分享用的。然而，不同操作系统的路径格式不一样，所以目前还不能支持。

Docker 挂载数据卷的默认权限是读写，用户也可以通过 :ro 指定为只读。

docker run -d -P --name web -v /src/webapp:/opt/webapp:ro training/webapp python app.py

### 挂载一个本地主机文件作为数据卷

docker run --rm -it -v ~/.bash\_history:/.bash\_history ubuntu /bin/bash

## 数据卷容器

数据卷容器，其实就是一个正常的容器，专门用来提供数据卷供其它容器挂载的。如果你有一些持续更新的数据需要在容器之间共享，最好创建数据卷容器

1. 创建一个命名的数据卷容器 dbdata
2. 在其他容器中使用 --volumes-from 来挂载 dbdata 容器中的数据卷

## 利用数据卷容器来备份、恢复、迁移数据卷

# [使用网络](http://www.dockerinfo.net/%e4%bd%bf%e7%94%a8%e7%bd%91%e7%bb%9c)

## 外部访问容器

### 映射所有接口地址

docker run -d -p 5000:5000 training/webapp python app.py

hostPort:containerPort

### 映射到指定地址的指定端口

docker run -d -p 127.0.0.1:5000:5000 training/webapp python app.py

ip:hostPort:containerPort

### 映射到指定地址的任意端口

docker run -d -p 127.0.0.1::5000 training/webapp python app.py

### 查看映射端口配置

docker port

## 容器互联

容器的连接（linking）系统是除了端口映射外，另一种跟容器中应用交互的方式。该系统会在源和接收容器之间创建一个隧道，接收容器可以看到源容器指定的信息。

### 自定义容器命名

--name

使用 --link 参数可以让容器之间安全的进行交互。--link 参数的格式为 --link name:alias，其中 name 是要链接的容器的名称，alias 是这个连接的别名

docker run -d --name db training/postgres

docker run -d -P --name web --link db:db training/webapp python app.py

Docker 通过 2 种方式为容器公开连接信息：

* 环境变量
* 更新 /etc/hosts 文件

# [高级网络配置](http://www.dockerinfo.net/%e9%ab%98%e7%ba%a7%e7%bd%91%e7%bb%9c%e9%85%8d%e7%bd%ae)

# 只有在 Docker 服务启动的时候才能配置，而且不能马上生效

* -b BRIDGE or –bridge=BRIDGE –指定容器挂载的网桥
* –bip=CIDR –定制 docker0 的掩码
* -H SOCKET… or –host=SOCKET… –Docker 服务端接收命令的通道
* –icc=true|false –是否支持容器之间进行通信
* –ip-forward=true|false –请看下文容器之间的通信
* –iptables=true|false –禁止 Docker 添加 iptables 规则
* –mtu=BYTES –容器网络中的 MTU

**既可以在启动服务时指定，也可以 Docker 容器启动（docker run）时候指定。在 Docker 服务启动的时候指定则会成为默认值，后面执行 docker run 时可以覆盖设置的默认值。**

* –dns=IP\_ADDRESS… –使用指定的DNS服务器
* –dns-search=DOMAIN… –指定DNS搜索域

**只有在 docker run 执行时使用，因为它是针对容器的特性内容**

* -h HOSTNAME or –hostname=HOSTNAME –配置容器主机名
* –link=CONTAINER\_NAME:ALIAS –添加到另一个容器的连接
* –net=bridge|none|container:NAME\_or\_ID|host –配置容器的桥接模式
* -p SPEC or –publish=SPEC –映射容器端口到宿主主机
* -P or –publish-all=true|false –映射容器所有端口到宿主主机

## 配置 DNS

## 容器访问控制

容器的访问控制，主要通过 Linux 上的 iptables 防火墙来进行管理和实现

### 容器访问外部网络

### 容器之间访问

# [安全介绍](http://www.dockerinfo.net/%e5%ae%89%e5%85%a8%e4%bb%8b%e7%bb%8d)

# [Dockerfile介绍](http://www.dockerinfo.net/dockerfile%e4%bb%8b%e7%bb%8d)

Dockerfile 分为四部分：基础镜像信息、维护者信息、镜像操作指令和容器启动时执行指令。

RUN 指令将对镜像执行跟随的命令。每运行一条 RUN 指令，镜像添加新的一层，并提交。

最后是 CMD 指令，来指定运行容器时的操作命令。

### FROM

FROM <image>或FROM <image>:<tag>

### MAINTAINER

MAINTAINER <name>，指定维护者信息

### RUN

RUN <command> 或 RUN ["executable", "param1", "param2"]

### CMD

* CMD ["executable","param1","param2"] 使用 exec 执行，推荐方式；
* CMD command param1 param2 在 /bin/sh 中执行，提供给需要交互的应用；
* CMD ["param1","param2"] 提供给 ENTRYPOINT 的默认参数；

指定启动容器时执行的命令，每个 Dockerfile 只能有一条 CMD 命令。如果指定了多条命令，只有最后一条会被执行。

如果用户启动容器时候指定了运行的命令，则会覆盖掉 CMD 指定的命令。

### EXPOSE

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

告诉 Docker 服务端容器暴露的端口号，供互联系统使用。

### ENV

ENV <key> <value> 。 指定一个环境变量，会被后续 RUN 指令使用，并在容器运行时保持。

ENV PG\_MAJOR 9.3

ENV PG\_VERSION 9.3.4

RUN curl -SL http://example.com/postgres-$PG\_VERSION.tar.xz | tar -xJC /usr/src/postgress && …

ENV PATH /usr/local/postgres-$PG\_MAJOR/bin:$PATH

### ADD

ADD <src> <dest>

复制指定的 <src> 到容器中的 <dest>。 其中 <src> 可以是Dockerfile所在目录的一个相对路径；也可以是一个 URL；还可以是一个 tar 文件（自动解压为目录）。

### COPY

COPY <src> <dest>

复制本地主机的 <src>（为 Dockerfile 所在目录的相对路径）到容器中的 <dest>。当使用本地目录为源目录时，推荐使用 COPY。

### ENTRYPOINT

* ENTRYPOINT ["executable", "param1", "param2"]
* ENTRYPOINT command param1 param2（shell中执行）

配置容器启动后执行的命令，并且不可被 docker run 提供的参数覆盖。

每个 Dockerfile 中只能有一个 ENTRYPOINT，当指定多个时，只有最后一个起效。

### VOLUME

VOLUME ["/data"]

创建一个可以从本地主机或其他容器挂载的挂载点，一般用来存放数据库和需要保持的数据等。

### USER ?

USER daemon

### WORKDIR

WORKDIR /path/to/workdir

为后续的 RUN、CMD、ENTRYPOINT 指令配置工作目录。

可以使用多个 WORKDIR 指令，后续命令如果参数是相对路径，则会基于之前命令指定的路径。

### ONBUILD

ONBUILD [INSTRUCTION]

配置当所创建的镜像作为其它新创建镜像的基础镜像时，所执行的操作指令

使用 ONBUILD 指令的镜像，推荐在标签中注明，例如 ruby:1.9-onbuild。

## 创建镜像

docker build [选项] 路径

.dockerignore

# [底层实现](http://www.dockerinfo.net/%e5%ba%95%e5%b1%82%e5%ae%9e%e7%8e%b0)

## 基本架构

## 命名 空间

每个容器都有自己单独的名字空间，运行在其中的应用都像是在独立的操作系统中运行一样。名字空间保证了容器之间彼此互不影响。

## 控制组（cgroups）

## 联合文件系统

## 容器格式

## Docker 网络实现