

目錄

Introduction	1.1
第一章 基本概念与操作	1.2
第二章 镜像与容器管理	1.3
第三章 数据卷与网络管理	1.4
第四章 Docker 安全及维护	1.5
第五章 编写 Dockerfile	1.6
第六章 Docker 运维技术	1.7
第七章 Docker API 及其开发	1.8

Docker 极速入门教程

讨论 Docker 技术的一本极速、无废话教程。

引言

目标读者

本书目的

内容一览

- 第一章 基本概念与操作
- 第二章 镜像与容器管理
- 第三章 数据卷与网络管理
- 第四章 Docker 安全及维护
- 第五章 编写 Dockerfile
- 第六章 Docker 运维技术
 - Docker Compose
 - Docker Swarm
 - Kubernetes
- 第七章 Docker API 及其开发

交流

GitHub

版权声明



本教程系 [欧长坤](#) 原创，采用 [知识共享署名-非商业性使用-禁止演绎 4.0 国际许可协议](#) 进行许可。

第一章 基本概念和操作

建立以 Docker 为核心的概念

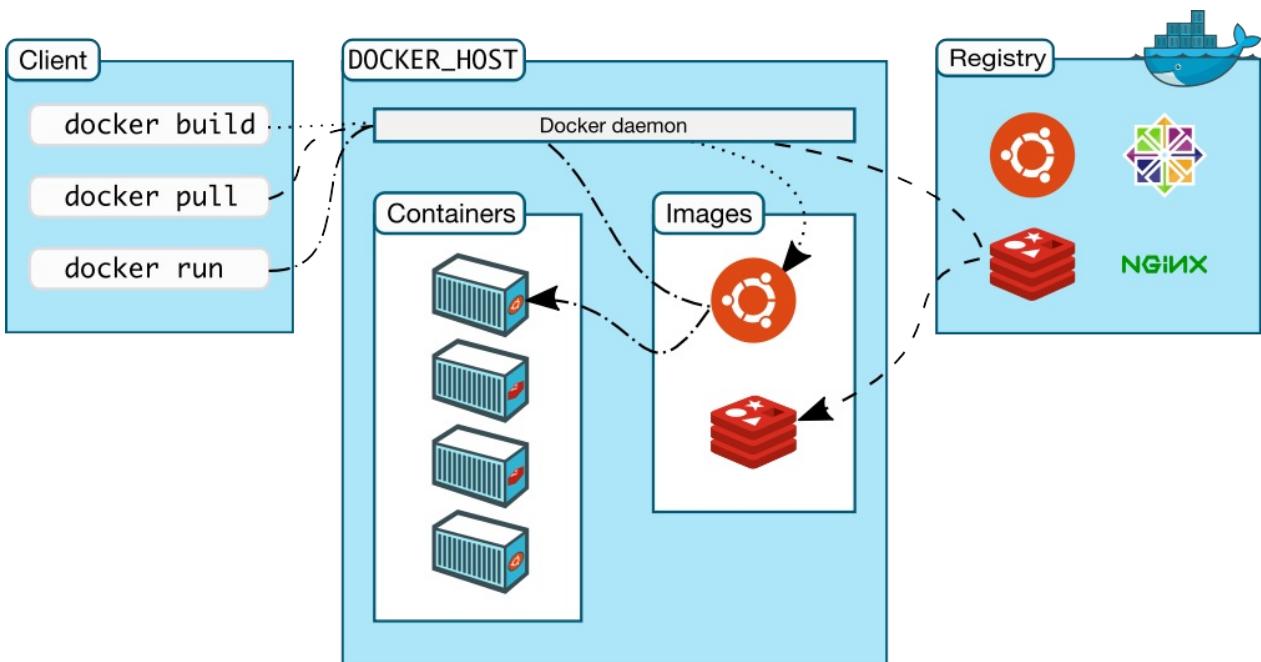
容器技术与 Docker 架构

容器通过对操作系统的资源访问进行限制，构建成独立的资源池，让应用运行在一个相对隔离的空间里，同时容器间也可以进行通信。容器技术对比虚拟化技术，容器比虚拟化更轻量级，对资源的消耗小很多。容器操作也更快捷，启动和停止都要比虚拟机快。但Docker容器需要与主机共享操作系统内核，不能像虚拟机那样运行独立的内核。

Docker 则是一个基于 Linux 容器（LXC, Linux Container）技术构建的容器引擎。Docker 支持将应用打包进一个可以移植的容器中，重新定义了应用开发，测试，部署上线的过程。

使用 Docker，可以快速构建一个应用程序服务器、一个消息总线、一个持续集成测试环境，升值为生产或开发快速复制一套复杂的应用程序栈，可以说是让开发者成为 DevOps 的必备技能之一。

Docker 是一个 C/S 架构的程序，Docker 客户端通过向守护进程发出请求，守护进程和 Docker 容器将请求处理完成后返回响应的结果。由于 Docker 是基于 LXC 产生的，因此在其他（Mac/Windows）上不具备这样的技术时，Docker 依然能够运行则是得益于一个虚拟的宿主机，因此整个 Docker 的架构在 Linux 和其他平台上的架构略有不同，但随着如今 Docker 技术的发展，作为 Docker 的用户，我们完全不需要操心这一切，只需心中有下图即可：



此外，Docker 提供了一个命令行工具及一整套 RESTful API 让用户与守护进程进行交互，也就是我常说的 Docker API，这使得我们能够基于 Docker 来开发 Docker 相关的应用。

学习 Docker 需要先在脑子里建立三大基本概念，分别是：仓库、镜像、容器。

仓库(Registry)

仓库的概念和 git 里的仓库几乎完全一样。一个 Docker 仓库就相当于一个 Git 的代码仓库。DockerHub 的地位就如同 GitHub，用于托管和保存用户的镜像，因此用户可以在 DockerHub 上注册账号，管理或开源自己的镜像。

镜像(Image)

镜像是构建 Docker 的基础，就如同操作系统安装的 .iso 镜像一样，Docker 镜像就是这种类似物。镜像通过 Union 文件系统产生了层式的结构，从最基础的镜像，像盖房子一样一层层堆叠，从而完成整个镜像的构建。

容器(Container)

容器是从 Docker 镜像创建运行出来的一个实例（Instance），可以理解为一个虚拟的 Linux 环境。Docker 容器如同虚拟机，支持启动、停止、删除。每个容器之间互相隔离，隔离性弱于虚拟机。

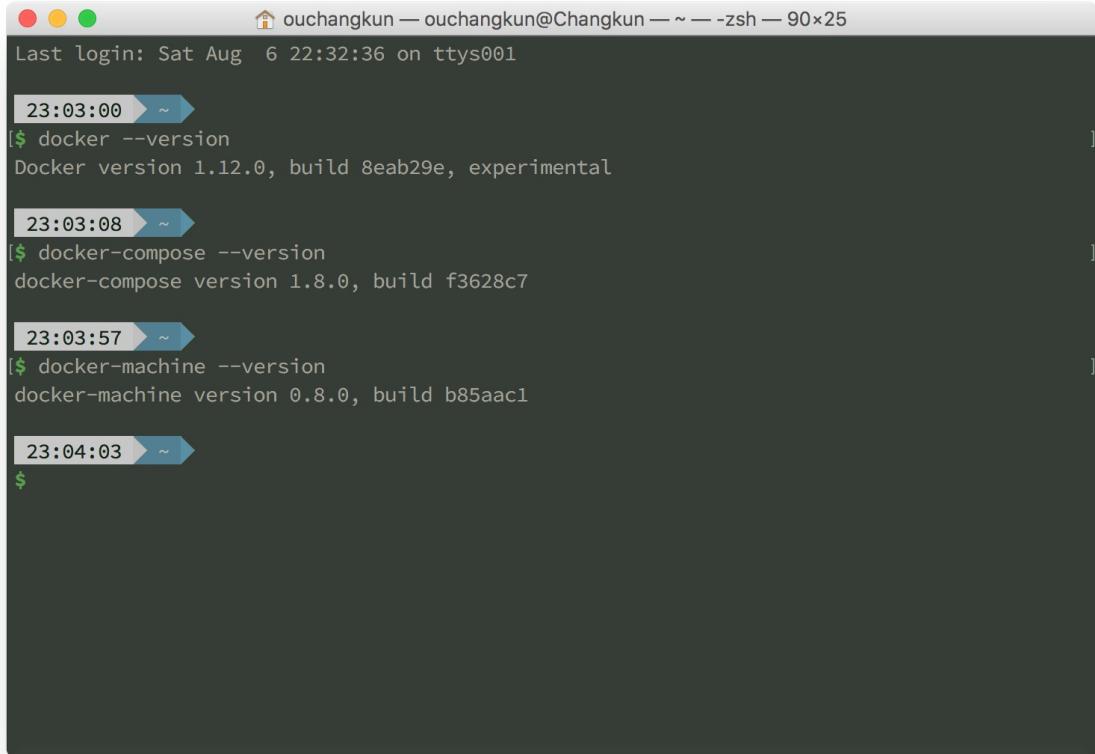
隔离的效果由 CGroups 和 Namespaces 实现，CGroups (Control Group, Linux 内核特性之一) 对 CPU、内存、磁盘资源等访问进行限制，而 Namespaces 提供了环境的隔离。

安装 Docker

安装 Docker 的方法现在已经变得非常的简单了，这里不再叙述，下面是各个平台的安装方法：

- Mac: <https://docs.docker.com/docker-for-mac/>
- Linux: <https://docs.docker.com/engine/getstarted/>
- Window: <https://docs.docker.com/docker-for-windows/>

值得一提的是，现在(Docker v1.12) Mac 上的 Docker 已经不再需要使用 Docker ToolBox 了，只需要下载 Docker for Mac，就能够在命令行工具中使用了：



```
ouchangkun — ouchangkun@Changkun — ~ — zsh — 90x25
Last login: Sat Aug 6 22:32:36 on ttys001

23:03:00 ~
[$ docker --version
Docker version 1.12.0, build 8eab29e, experimental

23:03:08 ~
[$ docker-compose --version
docker-compose version 1.8.0, build f3628c7

23:03:57 ~
[$ docker-machine --version
docker-machine version 0.8.0, build b85aac1

23:04:03 ~
$
```

基本操作

首先要获得一个镜像，我们才能开始 Docker 的相关学习。

```
$ docker images          # 查看当前镜像
$ docker search ubuntu   # 搜索镜像
$ docker pull ubuntu      # 拉取一个镜像
```

```
ouchangkun — ouchangkun@Changkun — ~ — zsh — 105x48
[ 23:19:37 ~ ]$ docker images
REPOSITORY      TAG        IMAGE ID      CREATED       SIZE
ubuntu          latest     0f192147631d  5 weeks ago   132.8 MB
sebp/elk        latest     fd2c14c423b4  8 weeks ago   977.3 MB

[ 23:20:07 ~ ]$ docker search ubuntu
NAME              DESCRIPTION          STARS  OFFICIAL  AUTOMATED
ubuntu            Ubuntu is a Debian-based Linux operating s... 4416  [OK]
ubuntu-upstart    Upstart is an event-based replacement for ... 65   [OK]
rastasheep/ubuntu-sshd  Dockerized SSH service, built on top of of... 30   [OK]
torusware/speedus-ubuntu  Always updated official Ubuntu docker imag... 26   [OK]
ubuntu-debootstrap  debootstrap --variant=minbase --components...
nickistre/ubuntu-lamp  LAMP server on Ubuntu 8   [OK]
nuagebec/ubuntu   Simple always updated Ubuntu docker images... 6   [OK]
nimmis/ubuntu     This is a docker images different LTS vers... 5   [OK]
maxexcloo/ubuntu  Docker base image built on Ubuntu with Sup... 2   [OK]
admiringworm/ubuntu  Base ubuntu images based on the official u...
darksheer/ubuntu   Base Ubuntu Image -- Updated hourly 1   [OK]
jordi/ubuntu       Ubuntu Base Image 1   [OK]
teamrock/ubuntu   TeamRock's Ubuntu image configured with AW... 0   [OK]
lynntp/ubuntu     https://github.com/lynntp/docker-ubuntu 0   [OK]
datenbetrieb/ubuntu  custom flavor of the official ubuntu base ...
life360/ubuntu    Ubuntu is a Debian-based Linux operating s... 0   [OK]
esycat/ubuntu     Ubuntu LTS 0   [OK]
webhippie/ubuntu  Docker images for ubuntu 0   [OK]
widerplan/ubuntu  Our basic Ubuntu images. 0   [OK]
crocon/ubuntu     Crosconized Ubuntu 0   [OK]
smartentry/ubuntu  ubuntu with smartentry 0   [OK]
konstruktoeid/ubuntu  Ubuntu base image 0   [OK]
ustclug/ubuntu    ubuntu image for docker with USTC mirror 0   [OK]
dorapro/ubuntu    ubuntu image 0   [OK]
uvatbc/ubuntu     Ubuntu images with unprivileged user 0   [OK]

[ 23:20:17 ~ ]$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
43db9dbdcb30: Pull complete
2dc64e8f8d4f: Pull complete
670a583e1b50: Pull complete
183b0bfcd10e: Pull complete
Digest: sha256:c6674c44c6439673bf56536c1a15916639c47ea04c3d6296c5df938add67b54b
Status: Downloaded newer image for ubuntu:latest

[ 23:21:14 ~ ]$
```

拿到镜像后，我们可以开始进行一些基本的操作了。

```
$ # 运行 docker 容器
$ docker run ubuntu echo "hello docker"
$ # 查看运行的容器列表,
$ # docker 在执行完命令后会主动停止,
$ # 所以使用 -a 参数可以查看到已经停止的容器. 此外, -l 可以查看最后一次启动的容器, -q 只查看容器 ID
$ docker ps
$ # 对于 run 命令可以使用 -it 参数进入 bash 来保持容器运行
$ docker run -i -t ubuntu /bin/bash
```

`docker run` 的 `-i (interactive)` 参数保证了容器中的 `STDIN` 开启，要让交互式 `shell` 长期运行，就必须要开启持久的标准输入；`-t (tty)` 指定让 Docker 创建一个伪 `tty` 终端。

The screenshot shows a terminal window with four command-line sessions:

- Session 1 (23:31:05):

```
[~]$ docker run ubuntu echo "hello docker"
hello docker
```
- Session 2 (23:31:17):

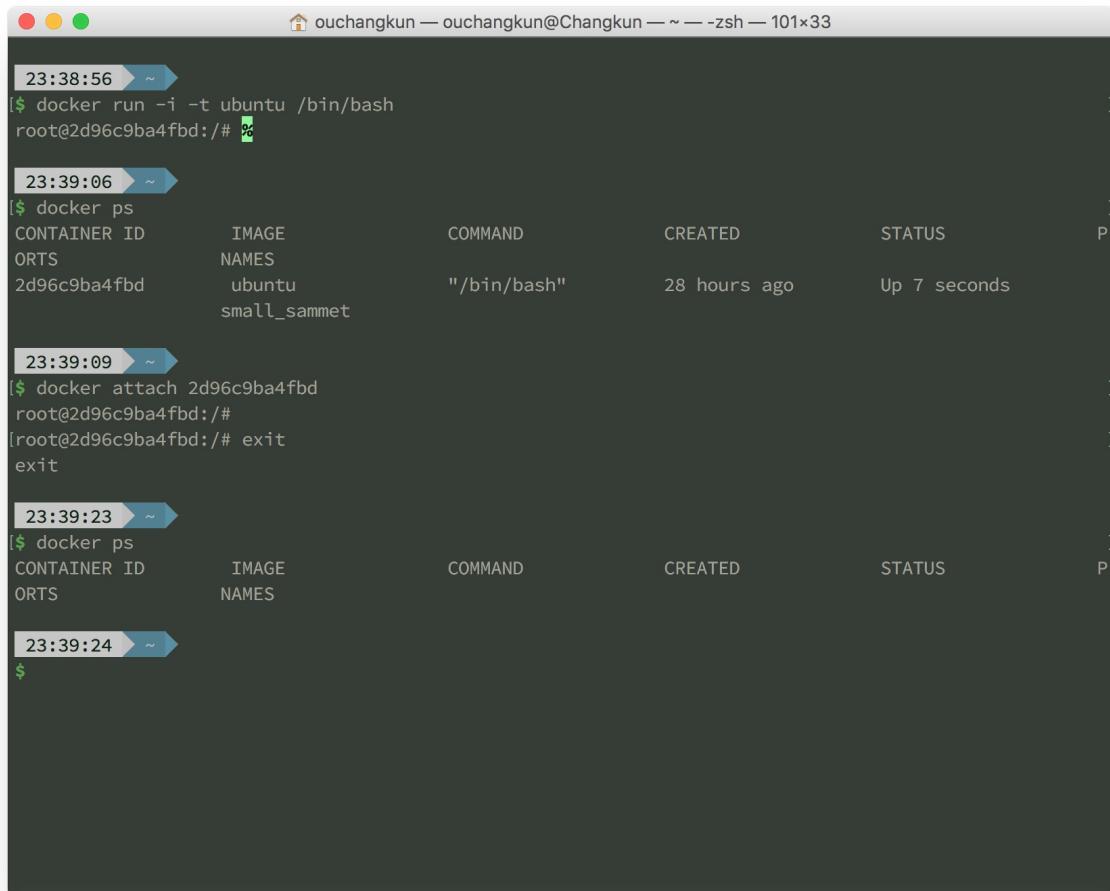
```
[~]$ docker ps -a
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS          NAMES
cb5a5901b04        ubuntu      "echo 'hello docker'"  28 hours ago    Exited (0) 5 sec   elegant_goodall
6a88ed92e257       ubuntu      "echo 'hello docker'"  28 hours ago    Exited (0) 5 min   evil_kirch
5b9ef060ce2b       sebp/elk    "/usr/local/bin/start"  4 weeks ago     Exited (0) 3 wee  elk-test
```
- Session 3 (23:31:22):

```
[~]$ docker ps
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS          NAMES
```
- Session 4 (23:31:27):

```
[~]$ docker run -i -t ubuntu /bin/bash
[root@c1b57c7dcac7:/]# ls
bin  dev  home  lib64  mnt  proc  run  srv  tmp  var
boot  etc  lib  media  opt  root  sbin  sys  usr
root@c1b57c7dcac7:/#
```

在 Docker 中，涉及到磁盘、网络、设备等 Linux 特权命令都无法执行，因此我们不能够执行诸如 `reboot` 之类的命令，可以通过 `exit` 退出 `bash`。

如果希望退出后依然保持容器运行，可以使用 `Ctrl+p` 及 `Ctrl+q` 两组快捷键，然后使用 `docker attach` 能够再次进入 `bash`。



A screenshot of a macOS terminal window titled "ouchangkun — ouchangkun@Changkun — ~ — -zsh — 101x33". The terminal shows the following sequence of commands:

```
23:38:56 ~
[$ docker run -i -t ubuntu /bin/bash
root@2d96c9ba4fb:/# %

23:39:06 ~
[$ docker ps
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS           PORTS
ORTS                NAMES
2d96c9ba4fb        ubuntu              "/bin/bash"   28 hours ago    Up 7 seconds

23:39:09 ~
[$ docker attach 2d96c9ba4fb
root@2d96c9ba4fb:/# exit
exit

23:39:23 ~
[$ docker ps
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS           PORTS
ORTS                NAMES

23:39:24 ~
$
```

```
$ docker start <id>      # 根据 ID 启动一个容器
$ docker stop <id>        # 根据 ID 停止一个容器
$ docker restart <id>     # 根据 ID 重启一个容器
```

```
ouchangkun — ouchangkun@Changkun — ~ — -zsh — 101x60
[$ docker ps -l
CONTAINER ID        IMAGE           COMMAND      CREATED        STATUS
          PORTS     NAMES
2d96c9ba4fbdb      ubuntu          "/bin/bash"   28 hours ago   Exited (0) 48 seconds
ago
ago
23:52:12 ~
[$ docker start 2d96c9ba4fbdb
2d96c9ba4fbdb

23:52:20 ~
[$ docker ps -l
CONTAINER ID        IMAGE           COMMAND      CREATED        STATUS
          PORTS     NAMES
2d96c9ba4fbdb      ubuntu          "/bin/bash"   28 hours ago   Up 3 seconds
ago
ago
23:52:23 ~
[$ docker attach 2d96c9ba4fbdb
root@2d96c9ba4fbdb:/#
[root@2d96c9ba4fbdb:/# exit
exit

23:52:34 ~
[$ docker ps -l
CONTAINER ID        IMAGE           COMMAND      CREATED        STATUS
          PORTS     NAMES
2d96c9ba4fbdb      ubuntu          "/bin/bash"   28 hours ago   Exited (0) 5 seconds
ago
ago
23:52:39 ~
[$ docker restart 2d96c9ba4fbdb
2d96c9ba4fbdb

23:52:46 ~
[$ docker attach 2d96c9ba4fbdb
root@2d96c9ba4fbdb:/#
[root@2d96c9ba4fbdb:/# exit
exit

23:52:55 ~
[$ docker ps -l
CONTAINER ID        IMAGE           COMMAND      CREATED        STATUS
          PORTS     NAMES
2d96c9ba4fbdb      ubuntu          "/bin/bash"   28 hours ago   Exited (0) 4 seconds
ago
ago
23:53:03 ~
[$ docker start 2d96c9ba4fbdb
2d96c9ba4fbdb

23:53:17 ~
[$ docker stop 2d96c9ba4fbdb
2d96c9ba4fbdb

23:53:21 ~
[$ docker ps -l
CONTAINER ID        IMAGE           COMMAND      CREATED        STATUS
          PORTS     NAMES
2d96c9ba4fbdb      ubuntu          "/bin/bash"   28 hours ago   Exited (0) 3 seconds
```

```
$ docker inspect <id>      # 查看容器或镜像内部信息
$ docker top <id>          # 查看容器中运行进程等信息
```

```
ouchangkun — ouchangkun@Changkun — ~ — zsh — 101x30
23:56:46 ~
[$ docker ps -l
CONTAINER ID        IMAGE       COMMAND      CREATED     STATUS
PORTS               NAMES
2d96c9ba4fbdb      ubuntu      "/bin/bash"   29 hours ago   Exited (0) 3 minutes
ago                  small_sammet

23:56:50 ~
[$ docker start 2d96c9ba4fbdb
2d96c9ba4fbdb

23:56:56 ~
[$ docker inspect 2d96c9ba4fbdb
[
  {
    "Id": "2d96c9ba4fb7c5f18321c37f806c8393ff07f09d98c91256ca9d0e968fb1543",
    "Created": "2016-08-05T10:53:51.359476657Z",
    "Path": "/bin/bash",
    "Args": [],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 7790,
      "ExitCode": 0,
      "Error": ""
    }
  }
]
```

```
ouchangkun — ouchangkun@Changkun — ~ — zsh — 101x30
{
  "IPPrefixLen": 16,
  "IPv6Gateway": "",
  "MacAddress": "02:42:ac:11:00:02",
  "Networks": {
    "bridge": {
      "IPAMConfig": null,
      "Links": null,
      "Aliases": null,
      "NetworkID": "c107ee7ef57841bce66e88d9eeacb10c1db82aee6af1b30653a83ff46866ad4",
      "EndpointID": "11ad9107ce9d81678bb9e8c99f748d5c3b3ff964f3d77527b570be61c5e2034d",
      "Gateway": "172.17.0.1",
      "IPAddress": "172.17.0.2",
      "IPPrefixLen": 16,
      "IPv6Gateway": "",
      "GlobalIPv6Address": "",
      "GlobalIPv6PrefixLen": 0,
      "MacAddress": "02:42:ac:11:00:02"
    }
  }
}
]

23:57:02 ~
[$ docker top 2d96c9ba4fbdb
PID      USER      TIME      COMMAND
7790     root      0:00      /bin/bash

23:57:21 ~
$
```

```
$ docker rm <id>          # 删除容器
$ docker rmi <id>          # 删除镜像
```

The screenshot shows a terminal window with a dark background and light-colored text. It displays a series of Docker commands and their outputs:

- Container Removal:**
 - Line 1: `00:00:05 ~` followed by the command `[\$ docker ps -a]` which lists several running containers.
 - Line 2: `[\$ docker rm c1b]` followed by `c1b`.
 - Line 3: `[\$ docker rm cbd 6a]` followed by `cbd` and `6a`.
 - Line 4: `[\$ docker ps -a]` showing the container `5b9ef060ce2b` has exited.
 - Line 5: `[\$ docker images]` showing three images: `ubuntu` (latest), `` (latest), and `sebp/elk` (latest).
 - Line 6: `[\$ docker rmi 0f]` followed by a long list of deleted images, starting with `Deleted: sha256:0f192147631d72486538039c51ef9557be11865030be2951a0fbe94ef66db618`.
- Image Removal:**
 - Line 7: `[\$ docker images]` showing the same set of images as before.
 - Line 8: `[\$ docker rmi sebp/elk]` followed by `sebp/elk`.

初识 Dockerfile

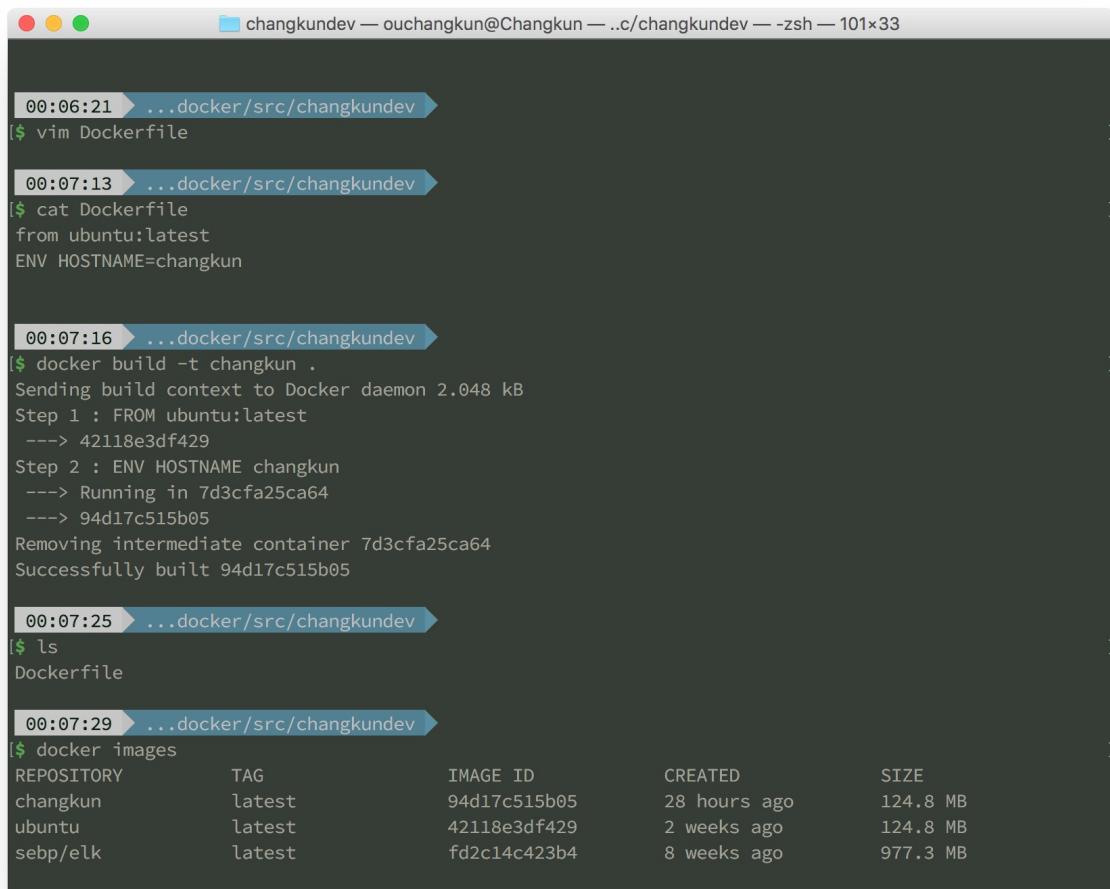
Dockerfile 用于编写一个镜像，下面创建了一个新的 Docker 镜像：

```
from ubuntu:latest      # 基于 ubuntu:latest
ENV HOSTNAME=changkun  # 设置 HOSTNAME 环境变量
```

通过 `docker build` 构建新镜像：

```
$ docker build -t changkun .
```

其中 `-t changkun` 指定了镜像的名字，`.` 标志从当前目录查找 Dockerfile。



The screenshot shows a terminal window on a Mac OS X desktop. The title bar says "changkundev — ouchangkun@Changkun — ..c/changkundev — zsh — 101x33". The terminal history is as follows:

- 00:06:21 ...docker/src/changkundev
- [` vim Dockerfile
- 00:07:13 ...docker/src/changkundev
- [` cat Dockerfile
- from ubuntu:latest
ENV HOSTNAME=changkun
- 00:07:16 ...docker/src/changkundev
- [` docker build -t changkun .
- Sending build context to Docker daemon 2.048 kB
- Step 1 : FROM ubuntu:latest
--> 42118e3df429
- Step 2 : ENV HOSTNAME changkun
--> Running in 7d3cfa25ca64
--> 94d17c515b05
- Removing intermediate container 7d3cfa25ca64
- Successfully built 94d17c515b05
- 00:07:25 ...docker/src/changkundev
- [` ls
- Dockerfile
- 00:07:29 ...docker/src/changkundev
- [` docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
changkun	latest	94d17c515b05	28 hours ago	124.8 MB
ubuntu	latest	42118e3df429	2 weeks ago	124.8 MB
sebp/elk	latest	fd2c14c423b4	8 weeks ago	977.3 MB

删除我们刚才创建的镜像：

```
changkundev ~ ouchangkun@Changkun ~ ..c/changkundev ~ -zsh ~ 101x34

00:07:47 ...docker/src/changkundev
[$ docker run -it changkun /bin/bash
[root@697f55694945:/# echo $HOSTNAME
changkun
[root@697f55694945:/# exit
exit

00:08:21 ...docker/src/changkundev
[$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
           PORTS
697f55694945      changkun            "/bin/bash"         28 hours ago       Exited (0) 13 se   gigantic_leakey
5b9ef060ce2b      sebp/elk           "/usr/local/bin/start" 4 weeks ago        Exited (0) 3 wee
ks ago            5000/tcp, 5044/tcp, 9200/tcp, 9300/tcp, 0.0.0.0:5601->5601/tcp   elk-test

00:08:34 ...docker/src/changkundev
[$ docker rm 6
6

00:08:40 ...docker/src/changkundev
[$ docker rmi 94
Untagged: changkun:latest
Deleted: sha256:94d17c515b0561b68ab46d0b66e126e0e4f360401ff77d94a951f14a70eeaa6b

00:08:57 ...docker/src/changkundev
[$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
ubuntu              latest   42118e3df429  2 weeks ago   124.8 MB
sebp/elk            latest   fd2c14c423b4  8 weeks ago   977.3 MB

00:09:28 ...docker/src/changkundev
$
```

注意: 删除 Docker 镜像时, 尽管可以使用 `-f` 参数强制删除镜像, 为了养成良好的运维习惯, 应该先使用 `docker rm` 删除容器后, 再删除镜像。

延伸阅读

- 理解 Docker
- Docker 核心技术预览

