原 Docker——入门实战

2018年07月14日 15:35:24 等一杯咖啡 阅读数：63668

I. Docker

简介

Docker是一种新兴的虚拟化技术，能够一定程度上的代替传统虚拟机。不过，Docker 跟传统的虚拟化方式相比具有众多的优势。我也将Docker类 /thon虚拟环境，可以有效的配置各个版本的开发环境，比如深度学习与Java环境。

其他的Docker简介也不需要过多介绍，可以参考很流行的《Docker – 从入门到实践》。关于博客，文末列出了最近在掘金上看到的一些入门类型文章。

优势

本人主要想用来配置开发环境，由于实验室机器系统环境版本等冲突的问题。

先用一张Docker – 从入门到实践里的图整体感受一下其独特的优势：

特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个

由于本人才疏学浅，这里便再节选一些原文中话具体描述Docker强大所在。个人感觉在入门完Docker再回头重新认识一下下面所述的五个优势相关，会对Docker的认识有更深的理解。

- **更高效的利用系统资源**：由于容器不需要进行硬件虚拟以及运行完整操作系统等额外开销，Docker 对系统资源的利用率更高。无论是应用执行速度、内存损耗或者文件存储速度，都要比传统虚拟机技术更高效。因此，相比虚拟机技术，一个相同配置的主机，往往可以运行更多数量的应用。
- **更快速的启动时间**：传统的虚拟机技术启动应用服务往往需要数分钟，而Docker 容器应用，由于直接运行于宿主内核，无需启动完整的操作系统，因此可以做到秒级、甚至毫秒级的启动时间。大大的节约了开发、测试、部署的时间。
- **一致的运行环境**：开发过程中一个常见的问题是环境一致性问题。由于开发环境、测试环境、生产环境不一致，导致有些bug 并未在开发过程中被发现。而Docker 的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会再出现「这段代码在我机器上没问题啊」这类问题。
- **持续交付和部署**：Docker是build once，run everywhere. 使用Docker 可以通过定制应用镜像来实现持续集成、持续交付、部署。开发人员可以通过Dockerfile 来进行镜像构建，并结合持续集成(Continuous Integration) 系统进行集成测试，而运维人员则可以直接在生产环境中快速部署该镜像，甚至结合持续部署(Continuous Delivery/Deployment) 系统进行自动部署。
- **更轻松的迁移**：Docker 使用的分层存储以及镜像的技术，使得应用重复部分的复用更为容易，也使得应用的维护更新更加简单，基于基础镜像进一步扩展镜像也变得非常简单。此外，Docker 团队同各个开源项目团队一起维护了一大批高质量的官方镜像，既可以直接在生产环境使用，又可以作为基础进一步定制，大大的降低了应用服务的镜像制作成本。使用Dockerfile 使镜像构建透明化，不仅仅开发团队可以理解应用运行环境，也方便运维团队理解应用运行所需条件，帮助更好的生产环境中部署该镜像。

镜像(Image)

镜像，从认识上简单的来说，就是面向对象中的类，相当于一个模板。从本质上来说，镜像相当于一个文件系统。Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

容器(Container)

容器，从认识上来说，就是类创建的实例，就是依据镜像这个模板创建出来的实体。容器的实质是进程，但与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的命名空间。因此容器可以拥有自己的root 文件系统、自己的网络配置、自己的进程空间，甚至自己的用户ID 空间。容器内的进程是运行在一个隔离的环境里，使用起来，就好像是在一个独立于宿主的系统下操作一样。这种特性使得容器封装的应用比直接在宿主运行更加安全。

仓库(Repository)

仓库，从认识上来说，就好像软件包上传下载站，有各种软件的不同版本被上传供用户下载。镜像构建完成后，可以很容易的在当前宿主机上运行，但是，如果需要在其它服务器上使用这个镜像，我们就需要一个集中的存储、分发镜像的服务，Docker Registry 就是这样的服务。

Docker版本

Docker 划分为CE 和EE。CE 即社区版（免费，支持周期三个月），EE 即企业版，强调安全，付费使用。Docker在1.13 版本之后，从2017年的3月1日开始，版本命名规则变为如下：

项目	说明
版本格式	YY.MM
Stable版本	每个季度发行
Edge版本	每个月发型

Docker CE 每月发布一个Edge 版本(17.03, 17.04, 17.05…)，每三个月发布一个Stable 版本(17.03, 17.06, 17.09…)，Docker EE 和Stable 版本号保持一致，但每个版本提供一年维护。

分层存储

因为镜像包含操作系统完整的root 文件系统，其体积往往是庞大的，因此在Docker设计时，就充分利用Union FS 的技术，将其设计为... 严格来说，镜像并非是像一个ISO 那样的打包文件，镜像只是一个虚拟的概念，其实际体现并非由一个文件组成，而是由**一组文件系统组成**，或者说，由**多层文件系统联合组成**。

镜像构建时，会一层层构建，前一层是后一层的基础。每一层构建完就不会再发生改变，后一层上的任何改变只发生在自己这一层。比如，删除前... 文件的操作，实际不是真的删除前一层的文件，而是仅在当前层标记为该文件已删除。在最终容器运行的时候，虽然不会看到这个文件，但是实际上该文件会一直... 镜像。因此，在构建镜像的时候，需要额外小心，每一层尽量只包含该层需要添加的东西，**任何额外的东西应该在该层构建结束前清理掉。**

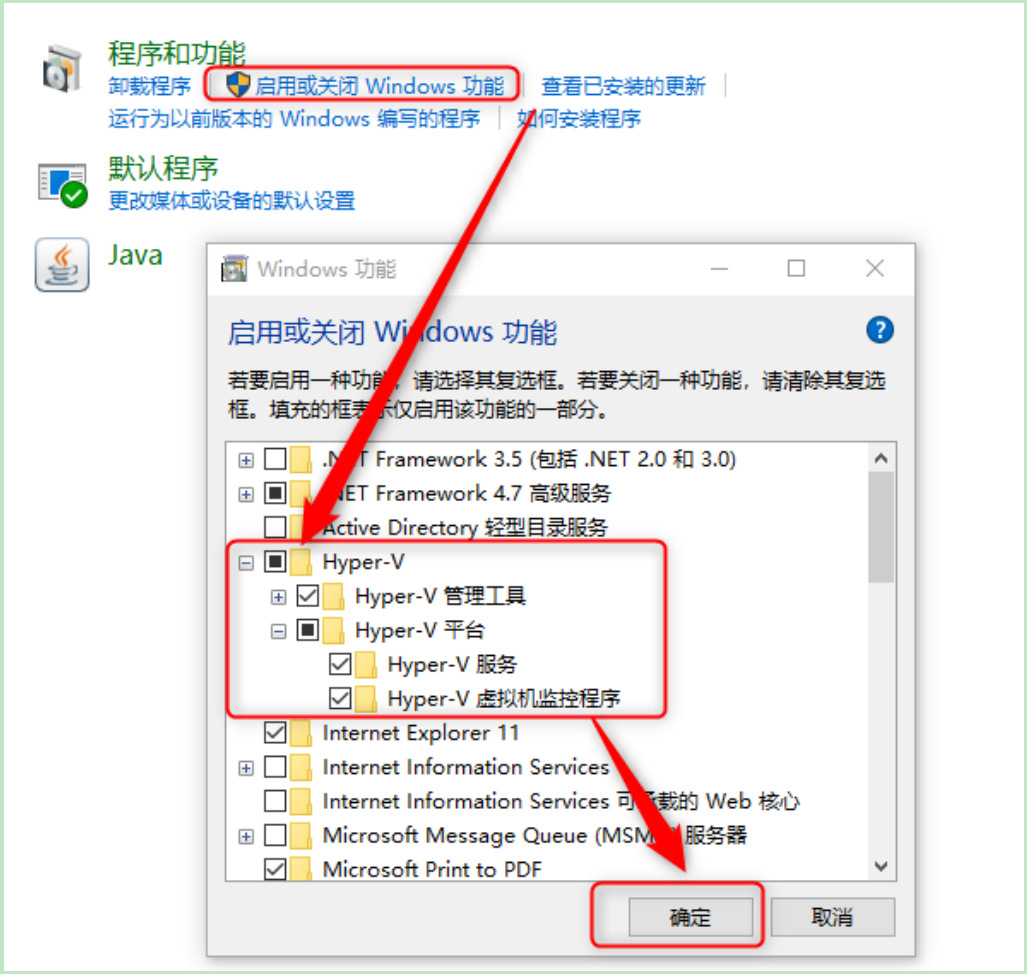
分层存储的特征还使得镜像的复用、定制变的更为容易。甚至可以用之前构建好的镜像作为基础层，然后进一步添加新的层，以定制自己所需的内... 构建新的镜像。

III. 安装Docker

Win10

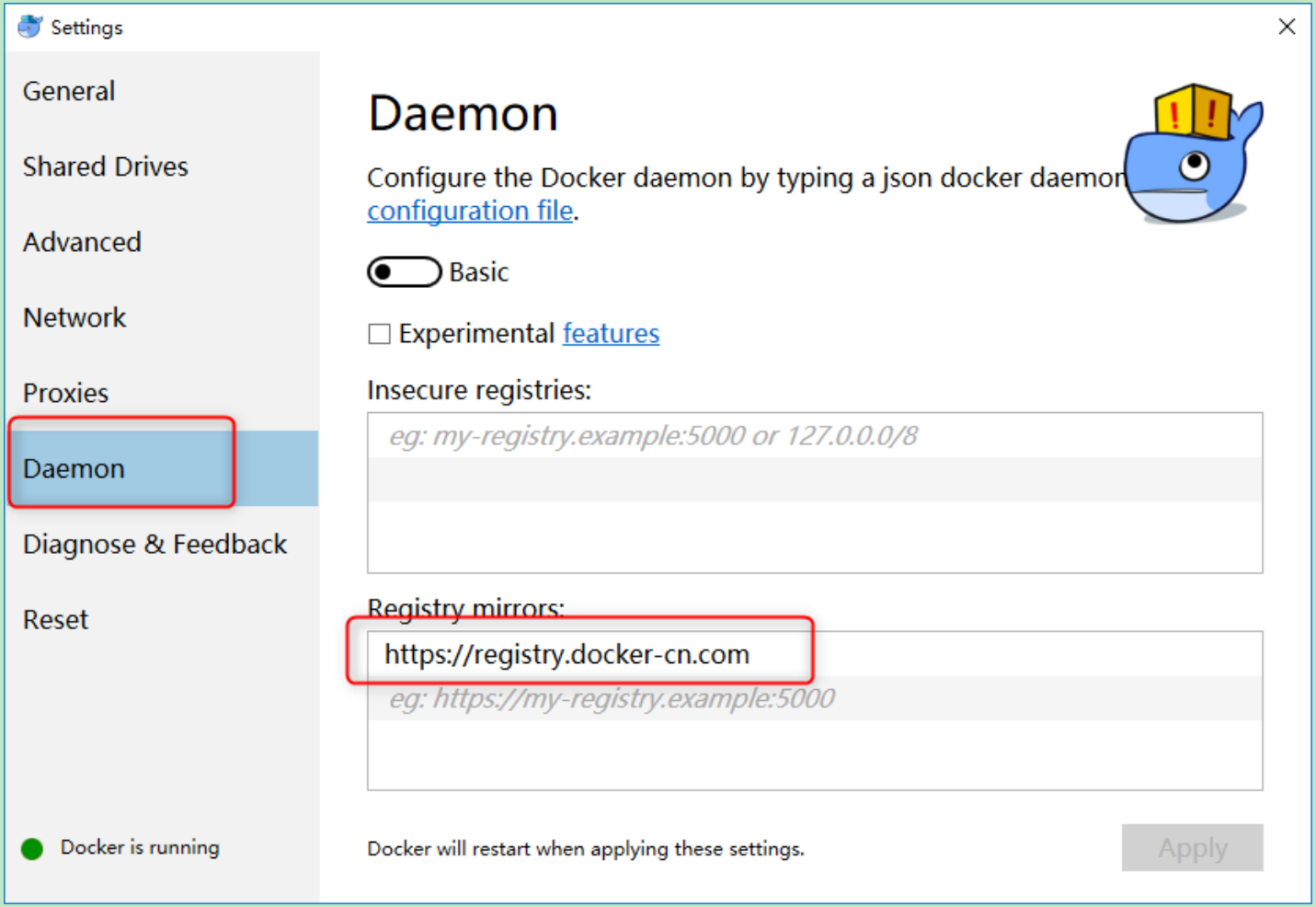
下载：https://docs.docker.com/docker-for-windows/install/

Docker支持64 位版本的Windows 10 Pro，且必须开启Hyper-V。开启方式为：打开“控制面板”->“程序”-> “启动或关闭Windows功能”，找到Hyper-V... ，确定重启电脑。



安装下载好的Docker for Windows Installer.exe，如下：

https://registry.docker-cn.com ，之后点击Apply保存后Docker就会重启并应用配置的镜像地址了。



Ubuntu16.04+

在Ubuntu系统中安装较为简单，官方提供了脚本供我们进行安装。

```
3 sudo sh get-docker.sh --mirror Aliyun
```


执行这个命令后，脚本就会自动的将一切准备工作做好，并且把Docker CE 的Edge版本安装在系统中。

启动Docker CE

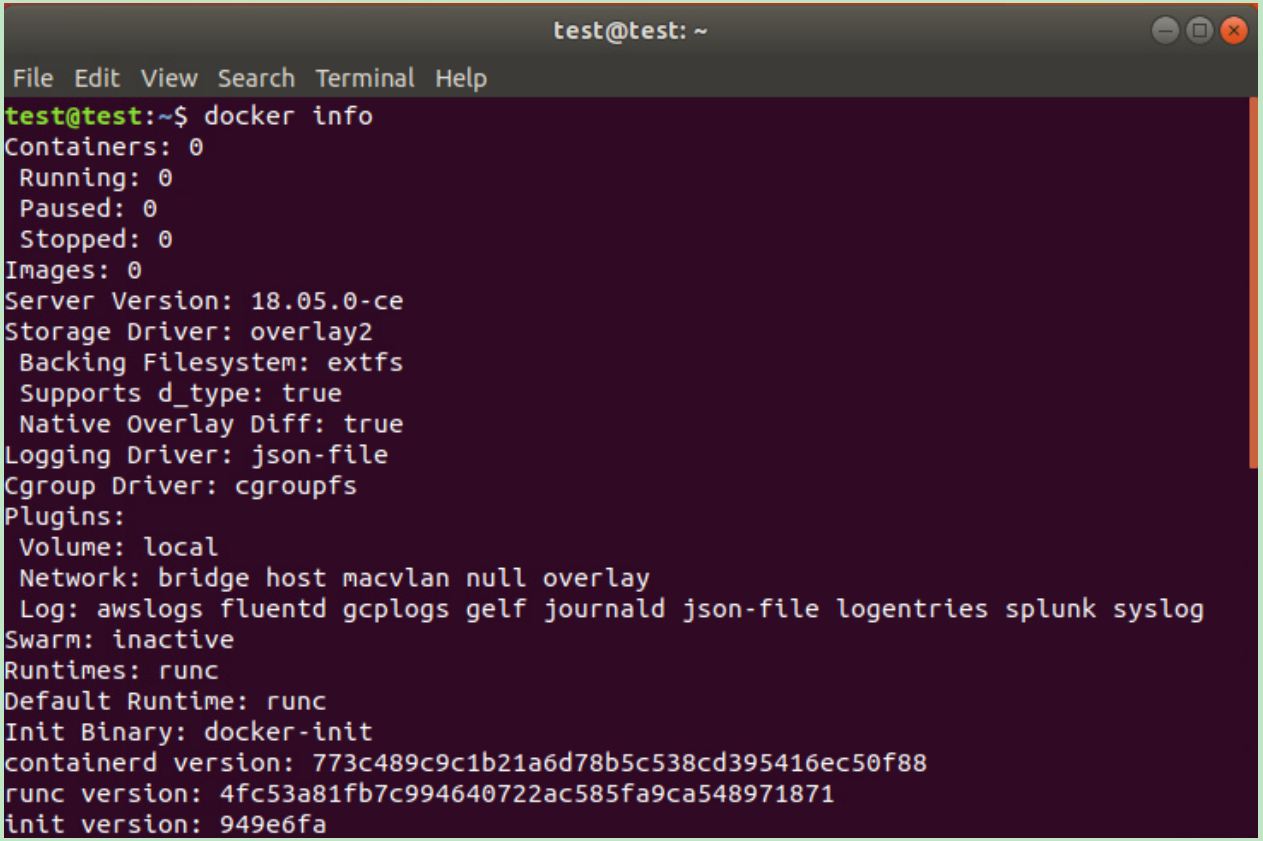
```
1 sudo systemctl enable docker
2 sudo systemctl start docker
```

建立docker 用户组

默认情况下，docker 命令会使用Unix socket 与Docker 引擎通讯。而只有root 用户和docker 组的用户才可以访问Docker 引擎的Unix socket。出于安全考虑，一般Ubuntu系统上不会直接使用root 用户。因此，更好地做法是将需要使用docker 的用户加入docker用户组。

```
1 # 建立docker组
2 sudo groupadd docker
3 # 将当前用户加入docker组
4 sudo usermod -aG docker $USER
```

注销当前用户，重新登录Ubuntu，输入docker info，此时可以直接出现信息。



配置国内镜像加速

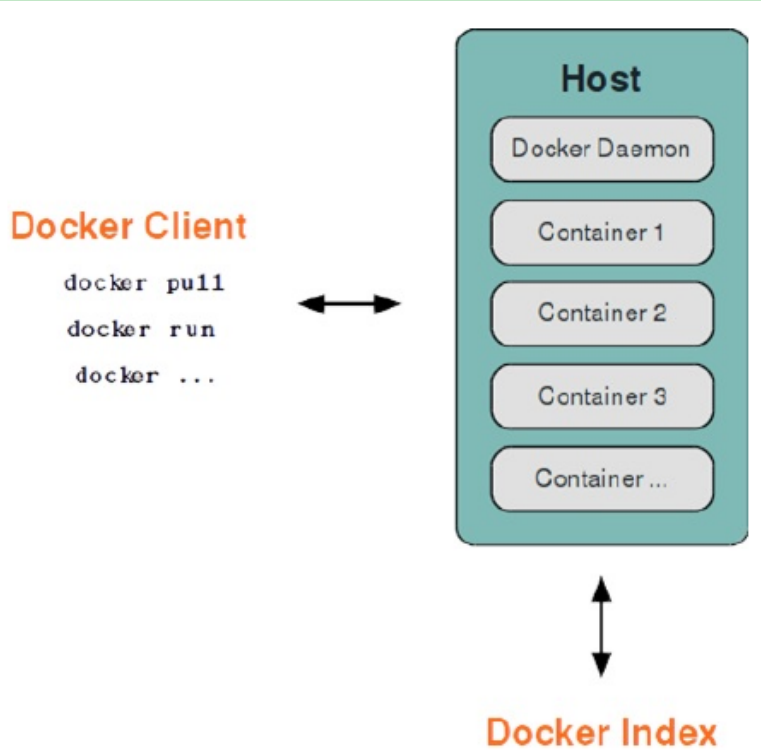
在/etc/docker/daemon.json 中写入如下内容（如果文件不存在请新建该文件）

```
{
      "registry-mirrors": [
    "https://registry.docker-cn.com"
  ]
}
```

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

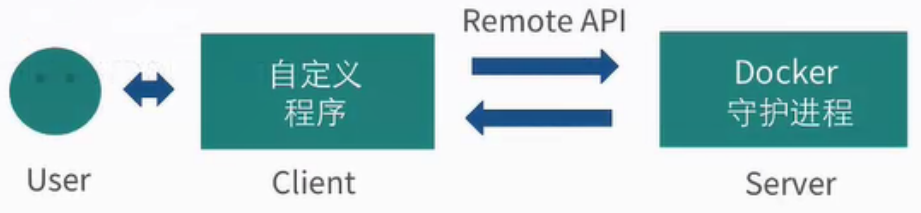
IV. Docker的C/S模式

Docker 采用了C/S 架构，包括客户端和服务端。Docker 守护进程（Daemon）作为服务端接受来自客户端的请求，并处理这些请求（创建、运行、分发容器）。



Docker 守护进程一般在宿主主机后台运行，等待接收来自客户端的消息；Docker 客户端则为用户提供一系列可执行命令，用户用这些命令实现跟Docker 守护进程交互。我们之前在Win10的命令行中便是最主要的客户端：

Docker也为我们提供了Remote API来操作Docker的守护进程，也意味着我们可以通过自己的程序来控制Docker的运行。客户端和服务端既可以运行在一个机器上，也可通过socket 或者RESTful API 来进行通信：



至于Docker的客户端与守护进程之间的通信，其连接方式为socket连接。主要有三种socket连接方式：

- unix:///var/run/docker.sock
- tcp://host:port
- fd://socketfd

完整的Docker的C/S连接方式的本质可以一般表示为如下：

V. 使用Docker

容器的基操

- **启动一次操作容器**：*docker run IMAGE_NAME [COMMAND] [ARG...]*

  例如，启动一个容器输出hello world。由于刚装上Docker，没有任何镜像，所以会先下载一个最新的ubuntu18.04的docker镜像。一次操作容器，处理完操作后会立即关闭容器。

```
1  docker run ubuntu echo 'hello world'
```

```
PS C:\Users\os> docker run ubuntu echo 'hello world'
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
6b98dfc16071: Pull complete
4001a1209541: Pull complete
6319fc68c576: Pull complete
b24603670dc3: Pull complete
97f170c87c6f: Pull complete
Digest: sha256:5f4bdc3467537cbbe563e80db2c3ec95d548a9145d64453b06939c4592d67b6d
Status: Downloaded newer image for ubuntu:latest
hello world
```

- **启动交互式容器**：*docker run -t -i –name=自定义名称 IMAGE_NAME /bin/bash*

  -i –interactive=true | false，默认是false

  -t –tty=true | false，默认是false

启动交互式的容器，就是类似虚拟机、云主机的操作方式，操作完一个命令后仍然可以继续：

```
1  docker run -i -t ubuntu /bin/bash
```

```
PS C:\Users\os> docker run -i -t ubuntu /bin/bash
root@5faa5c5ef9fa:/# ll
total 72
drwxr-xr-x   1 root root 4096 Jul  8 12:42 ./
drwxr-xr-x   1 root root 4096 Jul  8 12:42 ../
-rwxr-xr-x   1 root root    0 Jul  8 12:42 .dockerenv*
drwxr-xr-x   2 root root 4096 May 26 00:45 bin/
drwxr-xr-x   2 root root 4096 Apr 24 08:34 boot/
drwxr-xr-x   5 root root  360 Jul  8 12:42 dev/
drwxr-xr-x   1 root root 4096 Jul  8 12:42 etc/
drwxr-xr-x   2 root root 4096 Apr 24 08:34 home/
drwxr-xr-x   8 root root 4096 May 26 00:44 lib/
drwxr-xr-x   2 root root 4096 May 26 00:44 lib64/
drwxr-xr-x   2 root root 4096 May 26 00:44 media/
drwxr-xr-x   2 root root 4096 May 26 00:44 mnt/
drwxr-xr-x   2 root root 4096 May 26 00:44 opt/
dr-xr-xr-x 131 root root    0 Jul  8 12:42 proc/
drwx------   2 root root 4096 May 26 00:45 root/
drwxr-xr-x   1 root root 4096 Jun  5 21:20 run/
drwxr-xr-x   1 root root 4096 Jun  5 21:20 sbin/
drwxr-xr-x   2 root root 4096 May 26 00:44 srv/
dr-xr-xr-x  13 root root    0 Jul  8 12:42 sys/
drwxrwxrwt   2 root root 4096 May 26 00:45 tmp/
drwxr-xr-x   1 root root 4096 May 26 00:44 usr/
drwxr-xr-x   1 root root 4096 May 26 00:45 var/
root@5faa5c5ef9fa:/# cat /etc/issue
Ubuntu 18.04 LTS \n \l
```

- **查看容器**：*docker ps [-a] [-l]*

  省略 列出正在运行的容器

  -a all 列出所有容器

  -l latest 列出最近的容器

```
PS C:\Users\os> docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS               NAMES
5faa5c5ef9fa        ubuntu              "/bin/bash"              About a minute ago  Exited (0) 30 seconds ago                       thirsty_k
d5f651021402        ubuntu              "echo 'hello world'"     3 minutes ago       Exited (0) 3 minutes ago                        upbeat_al      i
PS C:\Users\os>
```


- **查看指定容器**：*docker inspect name | id*

  name指代具体的容器名称，id则是容器的唯一id标识。inspect命令可以详细的展示出容器的具体信息。

  ```
  1  docker inspect haha
  ```

```
PS C:\Users\os> docker inspect haha
[
    {
        "Id": "779483c689c317960d62830909c6d431687bb225d1dc09e20d04b3943448a5a4",
        "Created": "2018-07-08T12:47:06.46287Z",
        "Path": "/bin/bash",
        "Args": [],
        "State": {
            "Status": "exited",
            "Running": false,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 0,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2018-07-08T12:47:06.9090786Z",
            "FinishedAt": "2018-07-08T12:47:09.7980637Z"
        },
        "Image": "sha256:113a43faa1382a7404681f1b9af2f0d70b182c569aab71db497e33fa59ed87e6",
        "ResolvConfPath": "/var/lib/docker/containers/779483c689c317960d62830909c6d431687bb225d1dc09e20d04b3943448a5a4/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/779483c689c317960d62830909c6d431687bb225d1dc09e20d04b3943448a5a4/hostname",
        "HostsPath": "/var/lib/docker/containers/779483c689c317960d62830909c6d431687bb225d1dc09e20d04b3943448a5a4/hosts",
        "LogPath": "/var/lib/docker/containers/779483c689c317960d62830909c6d431687bb225d1dc09e20d04b3943448a5a4/779483c689c317960d62830909c6d                bb225d1d
c09e20d04b3943448a5a4-json.log",
        "Name": "/haha",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "default",
            "PortBindings": {},
            "RestartPolicy": {
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": false,
```


- **重新启动停止的容器**：*docker start [-i] 容器名*

```
docker start -i haha
```

```
PS C:\Users\os> docker start -i haha
root@779483c689c3:/# exit
exit
PS C:\Users\os> docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
779483c689c3        ubuntu              "/bin/bash"         About a minute ago  Exited (0) 11 seconds ago                       haha
5faa5c5ef9fa        ubuntu              "/bin/bash"         6 minutes ago       Exited (0) 5 minutes ago                        thirsty_k
d5f651021402        ubuntu              "echo 'hello world'"  8 minutes ago     Exited (0) 8 minutes ago                        upbeat_al
```

- **删除停止的容器**：*docker rm name | id*

```
docker rm thirsty_kepler
docker rm upbeat_albattani
```

```
PS C:\Users\os> docker rm thirsty_kepler
thirsty_kepler
PS C:\Users\os> docker rm upbeat_albattani
upbeat_albattani
PS C:\Users\os> docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
779483c689c3        ubuntu              "/bin/bash"         2 minutes ago       Exited (0) 52 seconds ago                       haha
PS C:\Users\os>
```

守护式容器

交互式容器在运行完命令退出后即停止，而实际中我们常常需要能够长时间运行，即使退出也能后台运行的容器，而守护式容器具备这一功能。守护式容器具有：

1. 能够长期运行；
2. 没有交互式会话；
3. 适合于运行应用程序和服务。

以守护形式运行容器

我们执行完需要的操作退出容器时，不要使用*exit*退出，可以利用*Ctrl+P Ctrl+Q*代替，以守护式形式推出容器。


```
PS C:\Users\os> docker start -i haha
root@779483c689c3:/# read escape sequence
PS C:\Users\os> docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
779483c689c3        ubuntu              "/bin/bash"         38 minutes ago      Up 24 seconds                           haha
PS C:\Users\os>
```


附加到运行中的容器

退出正在运行的容器，想要再次进入，需要使用attach命令：*docker attach name | id*

```
1  docker attach haha
```

启动守护式容器

启动守护式容器，可以在后台为我们执行操作：*docker run -d IMAGE_NAME [COMMAND] [ARG...]*

当命令在后台执行完毕，容器还是会关闭。这里防止容器立刻退出，写一个脚本循环输出“hello world”。

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

```
PS C:\Users\os> docker run --name hiahia -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 1; done"
8aca97c97e1843fb012193a206143e444c29c04744a645daa90803648e53f16f
PS C:\Users\os> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
8aca97c97e18        ubuntu              "/bin/sh -c 'while t…"   18 seconds ago      Up 17 seconds                           hiahia
779483c689c3        ubuntu              "/bin/bash"              About an hour ago   Up 17 minutes                           haha
PS C:\Users\os>
```


查看容器日志

当守护式容器在后台运行时，我们可以利用docker的日志命令查看其输出：*docker logs [-f] [-t] [--tail] IMAGE_NAME*

-t –timestamps=true | false，默认是false，显示时间戳

–tail="all" | 行数，显示最新行数的日志

```
PS C:\Users\os> docker logs hiahia —tail 0 -tf
2018-07-08T13:54:41.668546600Z hello world
2018-07-08T13:54:42.670824000Z hello world
2018-07-08T13:54:43.671492200Z hello world
2018-07-08T13:54:44.672935400Z hello world
2018-07-08T13:54:45.675946800Z hello world
2018-07-08T13:54:46.676650100Z hello world
2018-07-08T13:54:47.677189800Z hello world
2018-07-08T13:54:48.678338200Z hello world
2018-07-08T13:54:49.679073200Z hello world
2018-07-08T13:54:50.679657400Z hello world
2018-07-08T13:54:51.680314400Z hello world
2018-07-08T13:54:52.681785400Z hello world
PS C:\Users\os>
```

查看容器内进程

对运行的容器查看其进程：*docker top IMAGE_NAME*

```
PS C:\Users\os> docker top hiahia
PID                 USER                TIME                COMMAND
3080                root                0:00                /bin/sh -c while true; do echo hello world; sleep 1; done
4142                root                0:00                sleep 1
PS C:\Users\os> docker top haha
PID                 USER                TIME                COMMAND
2960                root                0:00                /bin/bash
PS C:\Users\os>
```

运行中容器启动新进程

Docker的理念是一个容器运行一个服务，但是往往需要对一个服务进行监控，所以也需要在已经运行服务的容器启动新的进程：*docker exec [-d] [-i] [-t] IMAGE_NAME [COMMAND] [ARG...]*

```
PS C:\Users\os> docker exec -i -t hiahia /bin/bash
root@8aca97c97e18:/#
root@8aca97c97e18:/# read escape sequence
PS C:\Users\os> docker top hiahia
PID                 USER                TIME                COMMAND
3080                root                0:00                /bin/sh -c while true; do echo hello world; sleep 1; done
5924                root                0:00                /bin/bash
5960                root                0:00                sleep 1
PS C:\Users\os>
```

停止守护式容器

发送信号停止容器：*docker stop 容器名*

强制停止：*docker kill 容器名*

VI. 案例：在容器中部署静态网站

容器的端口映射

命令：*run [-P] [-p]*

-P，–publish-all=true | false，大写的P表示为容器暴露的所有端口进行映射；

-p，–publish=[]，小写的p表示为容器指定的端口进行映射，有四种形式：

- *containerPort*：只指定容器的端口，宿主机端口随机映射；
- *hostPort:containerPort*：同时指定容器与宿主机端口一一映射；
- *ip::containerPort*：指定ip和容器的端口；
- *ip:hostPort:containerPort*：指定ip、宿主机端口以及容器端口。

例如：

```
1  docker run -p 80 -i -t ubuntu /bin/bash
2  docker run -p 8080:80 -i -t ubuntu /bin/bash
```

```
4 docker run -p 0.0.0.0::80 -i -t ubuntu /bin/bash
  docker run -p 0.0.0.0:8080:80 -i -t ubuntu /bin/bash
```

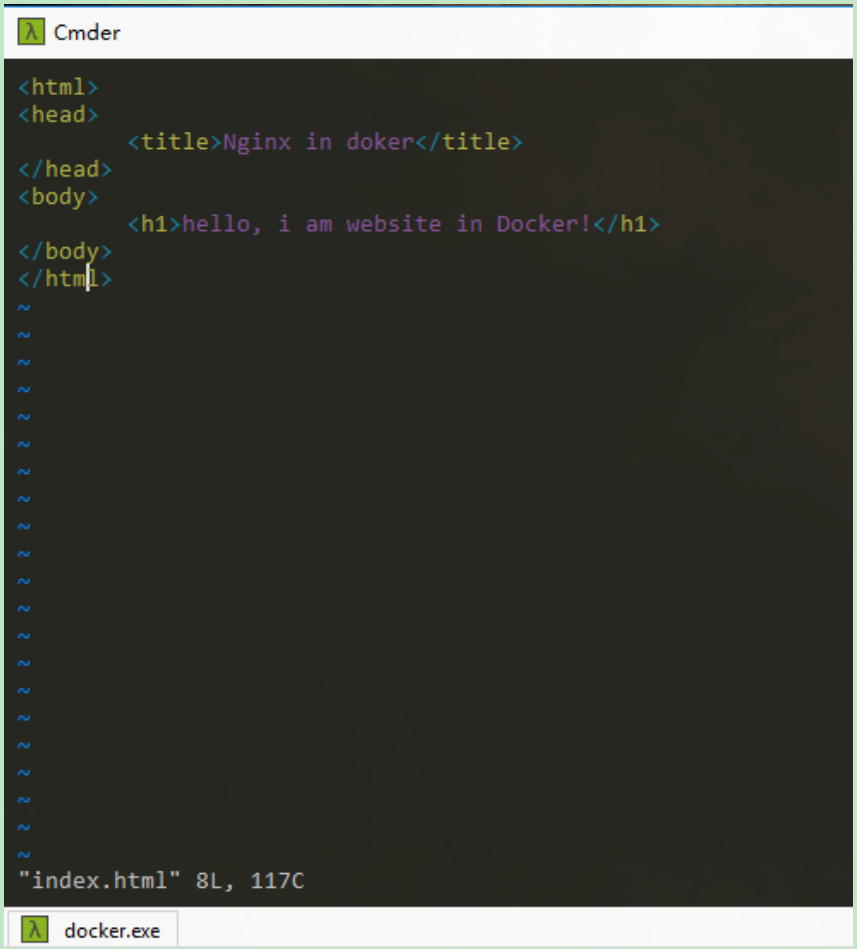
容器中部署Nginx服务

准备环境：

```
1 # 1. 创建映射80端口的交互式容器
2 docker run -p 80 --name web -i -t ubuntu /bin/bash
3 # 2. 更新源
4 apt-get update
5 # 3. 安装Nginx
6 apt-get install -y nginx
7 # 4. 安装Vim
8 apt-get install -y vim
```

创建静态页面：

```
1 mkdir -p /var/www/html
2 cd /var/www/html
3 vim index.html
```



修改Nginx配置文件:

```
# 查看Nginx安装位置
whereis nginx
# 修改配置文件
vim /etc/nginx/sites-enabled/default
```

```
# Default server configuration
#
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        # SSL configuration
        #
        # listen 443 ssl default_server;
        # listen [::]:443 ssl default_server;
        #
        # Note: You should disable gzip for SSL traffic.
        # See: https://bugs.debian.org/773332
        #
        # Read up on ssl_ciphers to ensure a secure configuration.
        # See: https://bugs.debian.org/765782
        #
        # Self signed certs generated by the ssl-cert package
        # Don't use them in a production server!
        #
        # include snippets/snakeoil.conf;

        root /var/www/html;

        # Add index.php to the list if you are using PHP
        index index.html index.htm;

        server_name _;

        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                try_files $uri $uri/ =404;
        }

        # pass PHP scripts to FastCGI server
        #
        #location ~ \.php$ {
        #       include snippets/fastcgi-php.conf;
        #
        #       # With php-fpm (or other unix sockets):
:
```

```
# 启动nginx
nginx
# 查看进程
ps -ef
```

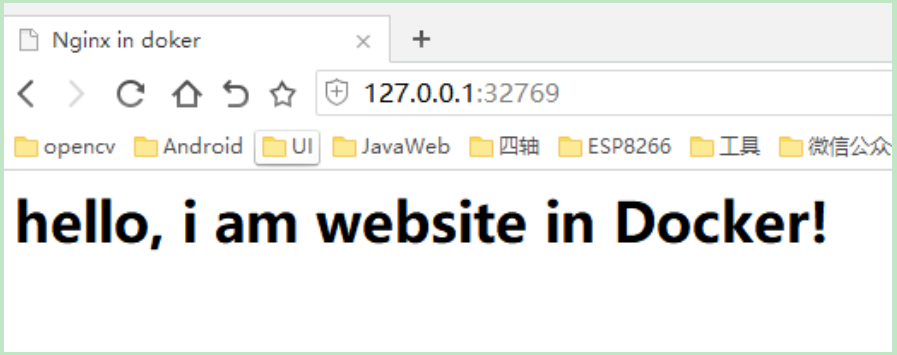
```
root@04b244947ce6:~# nginx
root@04b244947ce6:~# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 02:33 pts/0    00:00:00 /bin/bash
root          10       1  0 02:34 pts/0    00:00:00 vim index.html
root          23       1  0 02:46 ?        00:00:00 nginx: master process nginx
www-data      24      23  0 02:46 ?        00:00:00 nginx: worker process
www-data      25      23  0 02:46 ?        00:00:00 nginx: worker process
root          26       1  0 02:46 pts/0    00:00:00 ps -ef
root@04b244947ce6:~#
```

验证网站访问：

```
# 退出容器
Ctrl+P Ctrl+Q
# 查看容器进程
docker top web
# 查看容器端口映射情况
docker port web
```

```
C:\Users\os\Desktop
λ docker top web
PID                 USER                TIME                COMMAND
7619                root                0:00                /bin/bash
7659                root                0:00                vim index.html
7677                root                0:00                nginx: master process nginx
7678                xfs                 0:00                nginx: worker process
7679                xfs                 0:00                nginx: worker process

C:\Users\os\Desktop
λ docker port web
80/tcp -> 0.0.0.0:32769
```

通过宿主机地址加映射端口访问：



VII. 镜像基操

查看删除镜像

- **列出镜像**：*docker images [OPTIONS] [REPOSITORY]*

  -a，–all=false，显示所有镜像

–no-trunc=false，指定不使用截断的形式显示数据

-q，–quiet=false，只显示镜像的唯一id

```
C:\Users\os\Desktop
λ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              113a43faa138        4 weeks ago         81.2MB

C:\Users\os\Desktop
λ docker images --no-trunc
REPOSITORY          TAG                 IMAGE ID                                                                  CREATED             SIZE
ubuntu              latest              sha256:113a43faa1382a7404681f1b9af2f0d70b182c569aab71db497e33fa59ed87e6   4 weeks a           81.2MB

C:\Users\os\Desktop
λ docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              113a43faa138        4 weeks ago         81.2MB

C:\Users\os\Desktop
λ docker images -q
113a43faa138
```

- **查看镜像**：*docker inspect [OPTIONS] CONTAINER|IMAGE [CONTAINER|IMAGE]*

  -f，–format=“”

```
C:\Users\os\Desktop
λ docker inspect ubuntu
[
    {
        "Id": "sha256:113a43faa1382a7404681f1b9af2f0d70b182c569aab71db497e33fa59ed87e6",
        "RepoTags": [
            "ubuntu:latest"
        ],
        "RepoDigests": [
            "ubuntu@sha256:5f4bdc3467537cbbe563e80db2c3ec95d548a9145d64453b06939c4592d67b6d"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2018-06-05T21:20:54.310450149Z",
        "Container": "6713e927cc43b61a4ce3950a69907336ff55047bae9393256e32613a54321c70",
        "ContainerConfig": {
            "Hostname": "6713e927cc43",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "#(nop) ",
                "CMD [\"/bin/bash\"]"
            ],
            "ArgsEscaped": true,
            "Image": "sha256:c2775c69594daa3ee360d8e7bbca93c65d9c925e89bd731f12515f9bf8382164",
            "Volumes": null,
            "WorkingDir": "",
```

- **删除镜像**：*docker rmi [OPTIONS] IMAGE [IMAGE]*

–no-prune=false，保留未打标签的父镜像

- **虚悬镜像**：既没有仓库名，也没有标签，均为\

获取推送镜像

- **查找镜像**：*docker search [OPTIONS] TEAM*

  –automated=false，仅显示自动化构建的镜像

  –no-trunc=false，不以截断的方式输出

  –filter，添加过滤条件

```
C:\Users\os\Desktop
λ docker search ubuntu --filter=stars=3
NAME                                                DESCRIPTION                                     STARS               OFFICIAL            AUTOMATED
ubuntu                                              Ubuntu is a Debian-based Linux operating sys…   7927                [OK]
dorowu/ubuntu-desktop-lxde-vnc                      Ubuntu with openssh-server and NoVNC            193                                     [OK]
rastasheep/ubuntu-sshd                              Dockerized SSH service, built on top of offi…   156                                     [OK]
ansible/ubuntu14.04-ansible                         Ubuntu 14.04 LTS with ansible                   93                                      [OK]
ubuntu-upstart                                      Upstart is an event-based replacement for th…   87                  [OK]
neurodebian                                         NeuroDebian provides neuroscience research s…   50                  [OK]
ubuntu-debootstrap                                  debootstrap --variant=minbase --components=m…   38                  [OK]
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5   ubuntu-16-nginx-php-phpmyadmin-mysql-5       36                                      [OK]
nuagebec/ubuntu                                     Simple always updated Ubuntu docker images w…   23                                      [OK]
tutum/ubuntu                                        Simple Ubuntu docker images with SSH access     18
i386/ubuntu                                         Ubuntu is a Debian-based Linux operating sys…   13
ppc64le/ubuntu                                      Ubuntu is a Debian-based Linux operating sys…   12
1and1internet/ubuntu-16-apache-php-7.0              ubuntu-16-apache-php-7.0                        10                                      [OK]
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mariadb-10   ubuntu-16-nginx-php-phpmyadmin-mariadb-10     7                                       [OK]
eclipse/ubuntu_jdk8                                 Ubuntu, JDK8, Maven 3, git, curl, nmap, mc, …   6                                       [OK]
darksheer/ubuntu                                    Base Ubuntu Image -- Updated hourly             4                                       [OK]
codenvy/ubuntu_jdk8                                 Ubuntu, JDK8, Maven 3, git, curl, nmap, mc, …   4                                       [OK]
1and1internet/ubuntu-16-nginx-php-5.6-wordpress-4   ubuntu-16-nginx-php-5.6-wordpress-4             3                                       [OK]
```

- **拉取镜像**：*docker pull [OPTIONS] NAME [:TAG]*

  -a，–all-tags=false，下载所有的镜像（包含所有TAG）

```
C:\Users\os\Desktop
λ docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
b234f539f7a1: Pull complete
55172d420b43: Pull complete
5ba5bbeb6b91: Pull complete
43ae2841ad7a: Pull complete
f6c9c6de4190: Pull complete
Digest: sha256:b050c1822d37a4463c01ceda24d0fc4c679b0dd3c43e742730e2884d3c582e3a
Status: Downloaded newer image for ubuntu:16.04

C:\Users\os\Desktop
λ docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              16.04               5e8b97a2a082        4 weeks ago         114MB
ubuntu              latest              113a43faa138        4 weeks ago         81.2MB
```

- 推送镜像：*docker push NAME [:TAG]*

  Docker允许上传我们自己构建的镜像，需要注册DockerHub的账户。也可以上传到阿里云，地址：https://cr.console.aliyun.com/#/namespace/index

构建镜像

构建Docker镜像，可以保存对容器的修改，并且再次使用。构建镜像提供了自定义镜像的能力，以软件的形式打包并分发服务及其运行环境。Docker中提供了两种方式来构建镜像：

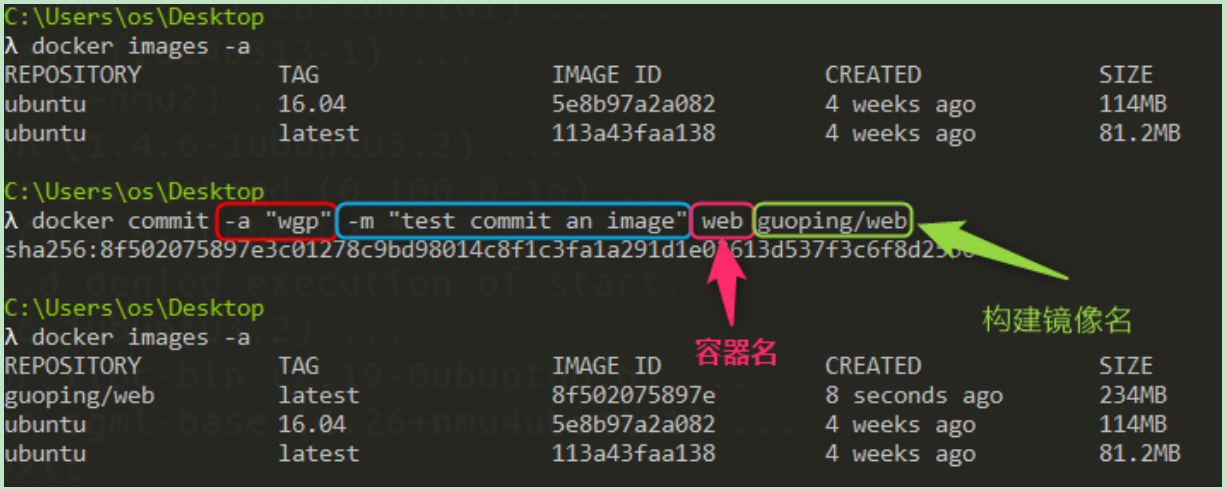
- 通过容器构建：*docker commit*
- 通过Dockerfile：*docker build*

使用commit命令构建镜像

命令：*docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]*

参数：-a，–author=“”，指定镜像的作者信息

-m，–message=“”，提交信息



使用Dockerfile文件构建镜像

Docker允许我们利用一个类似配置文件的形式来进行构建自定义镜像，在文件中可以指定原始的镜像，自定义镜像的维护人信息，对原始镜像采取的操作以及暴露的端口等信息。比如：

```
1 # Sample Dockerfile
2 FROM ubuntu:16.04
3 MAINTAINER wgp "Kingdompin@163.com"
4 RUN apt-get update
5 RUN apt-get install -y nginx
6 EXPOSE 80
```

命令：docker build [OPTIONS] DockerFile_PATH | URL | -

参数：–force-rm=false

–no-cache=false

–pull=false

–rm=true

-t，tag=“”，指定输出的镜像名称信息


```
C:\Users\os\Desktop
λ touch Dockerfile

C:\Users\os\Desktop
λ notepad "Dockerfile"

C:\Users\os\Desktop
λ docker build -t="guoping/web2" .
Sending build context to Docker daemon  2.927GB
Step 1/5 : FROM ubuntu:16.04
 ---> 5e8b97a2a082
Step 2/5 : MAINTAINER wgp "Kingdompin@163.com"
 ---> Running in 4b452f2a7986
Removing intermediate container 4b452f2a7986
 ---> be4f5dbe8421
Step 3/5 : RUN apt-get update
 ---> Running in d00cc48e7885
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [107 kB]
Get:3 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [83.5 kB]
Get:4 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
Get:5 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [660 kB]
Get:6 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]
Get:7 http://archive.ubuntu.com/ubuntu xenial/universe Sources [9802 kB]
Get:8 http://security.ubuntu.com/ubuntu xenial-security/restricted amd64 Packages [12.7 kB]
Get:9 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 Packages [452 kB]
Get:10 http://security.ubuntu.com/ubuntu xenial-security/multiverse amd64 Packages [3735 B]
Get:11 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages [1558 kB]
Get:12 http://archive.ubuntu.com/ubuntu xenial/restricted amd64 Packages [14.1 kB]
Get:13 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [9827 kB]
Get:14 http://archive.ubuntu.com/ubuntu xenial/multiverse amd64 Packages [176 kB]
Get:15 http://archive.ubuntu.com/ubuntu xenial-updates/universe Sources [260 kB]
Get:16 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [1038 kB]
Get:17 http://archive.ubuntu.com/ubuntu xenial-updates/restricted amd64 Packages [13.1 kB]
Get:18 http://archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages [825 kB]
Get:19 http://archive.ubuntu.com/ubuntu xenial-updates/multiverse amd64 Packages [18.8 kB]
Get:20 http://archive.ubuntu.com/ubuntu xenial-backports/main amd64 Packages [7321 B]
Get:21 http://archive.ubuntu.com/ubuntu xenial-backports/universe amd64 Packages [8088 B]
Fetched 25.3 MB in 2min 6s (200 kB/s)
Reading package lists...
Removing intermediate container d00cc48e7885
 ---> 685012120c30
```

创建编辑Dockerfile

```
Reading state information...
The following additional packages will be installed:
  fontconfig-config fonts-dejavu-core geoip-database libexpat1 libfontconfig1
  libfreetype6 libgd3 libgeoip1 libicu55 libjbig0 libjpeg-turbo8 libjpeg8
  libpng12-0 libssl1.0.0 libtiff5 libvpx3 libx11-6 libx11-data libxau6 libxcb1
  libxdmcp6 libxml2 libxpm4 libxslt1.1 nginx-common nginx-core sgml-base ucf
  xml-core
Suggested packages:
  libgd-tools geoip-bin fcgiwrap nginx-doc ssl-cert sgml-base-doc debhelper
The following NEW packages will be installed:
  fontconfig-config fonts-dejavu-core geoip-database libexpat1 libfontconfig1
```

登录 注册 ×

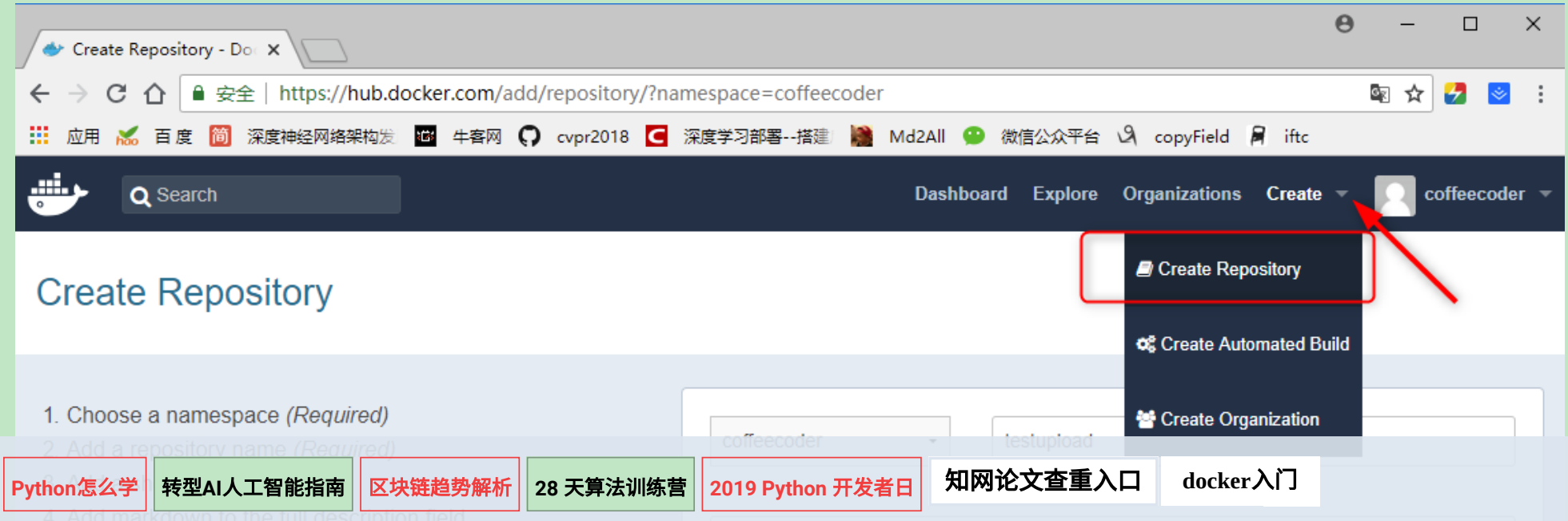
32 20

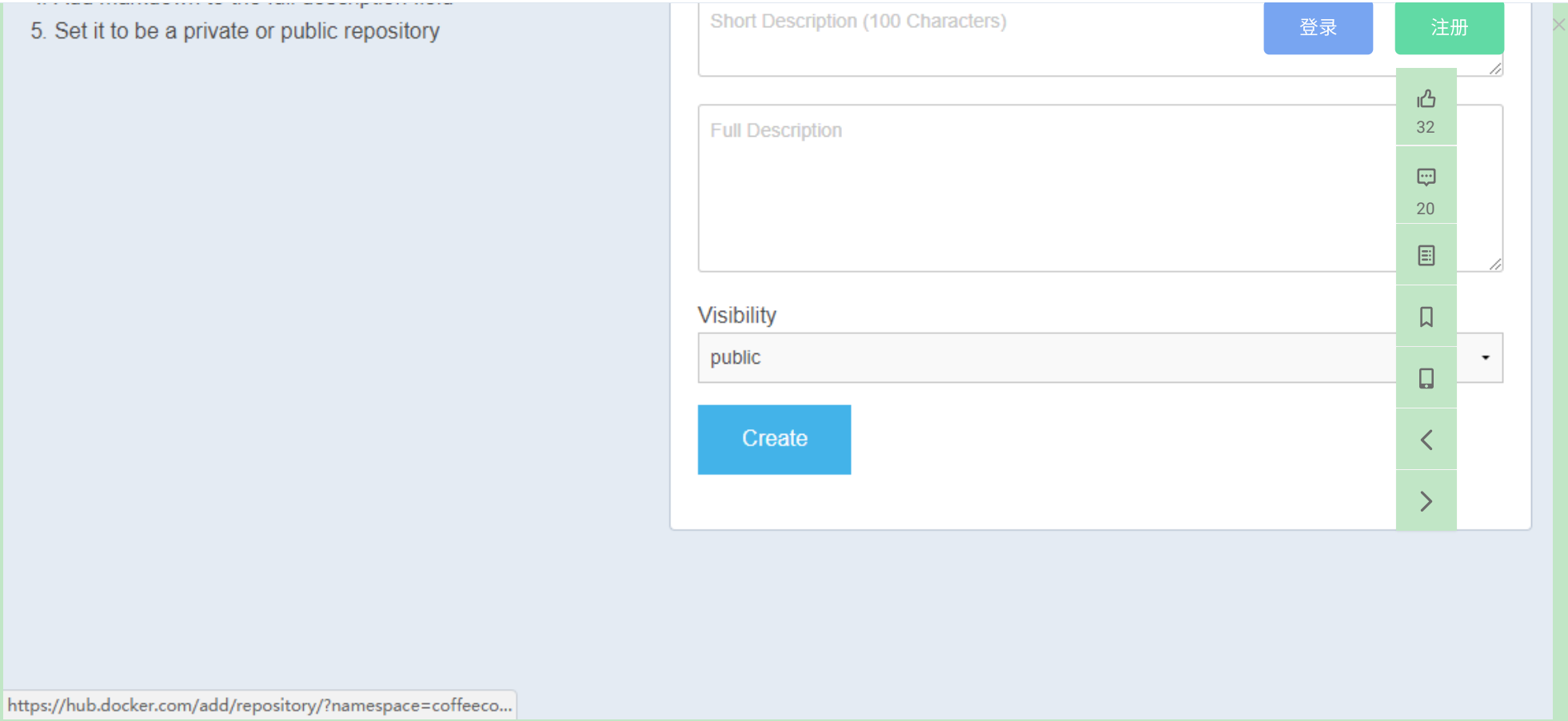
VIII. 镜像迁移

我们制作好的镜像，一般会迁移或分享给其他需要的人。Docker提供了几种将我们的镜像迁移、分享给其他人的方式。推荐镜像迁移应该直接使用...er Registry，无论是直接使用Docker Hub还是使用内网私有Registry都可以。使用镜像频率不高，镜像数量不多的情况下，我们可以选择以下两种方式。

上传Docker Hub

首先，需要在Docker Hub上申请注册一个帐号（人机验证时需要科学上网）。然后我们需要创建仓库，指定仓库名称。





在终端中登录你的Docker Hub账户，输入 `docker login` ，输入用户名密码即可登录成功。

```
C:\Users\os
λ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.

Username: coffeecoder
Password:
Login Succeeded
```

查看需要上传的镜像，并将选择的镜像打上标签，标签名需和Docker Hub上新建的仓库名称一致，否则上传失败。给镜像打标签的命令如下。

```
docker tag <existing-image> <hub-user>/<repo-name>[:<tag>]
```

其中 existing-image 代表本地待上传的镜像名加tag，后面 <hub-user>/<repo-name>[:<tag>] 则是为上传更改的标签名，tag不指定则为latest。

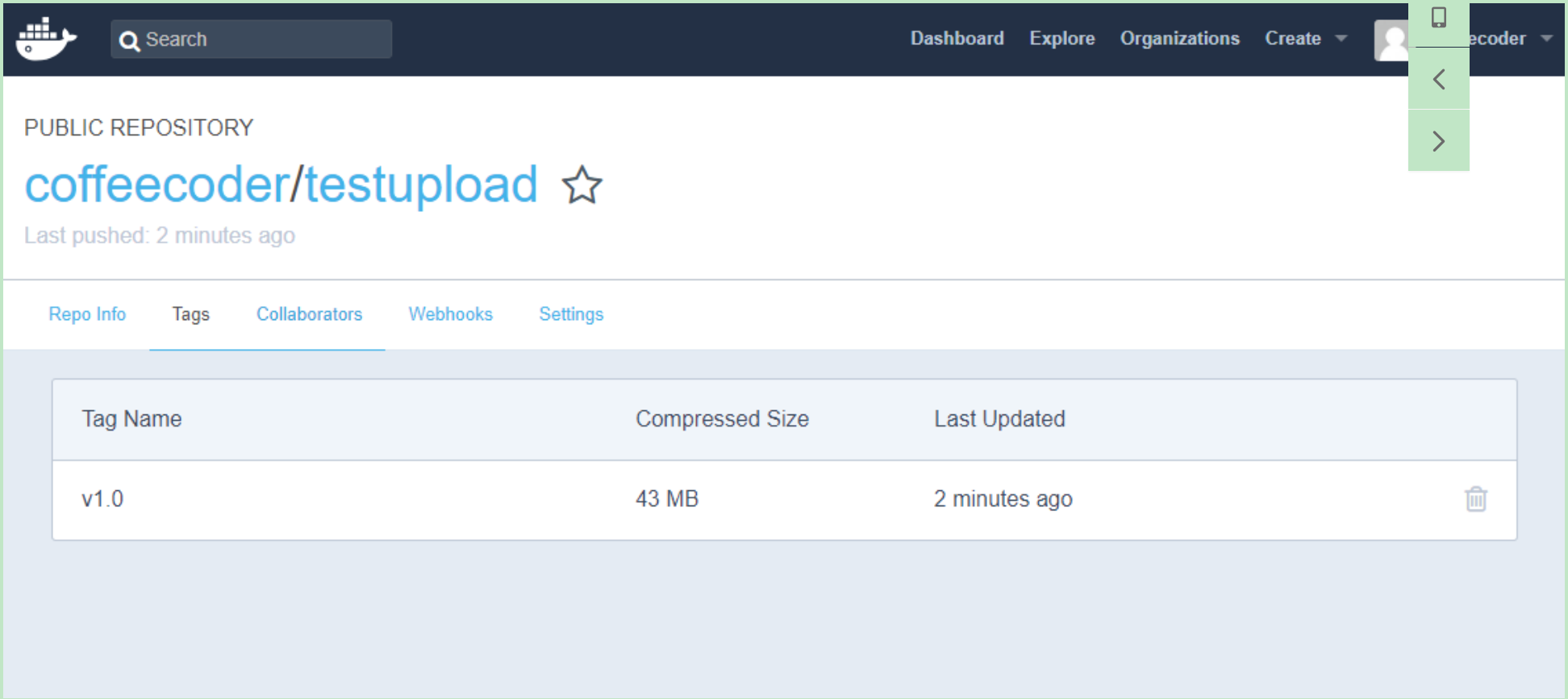
```
C:\Users\os
λ docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              16.04               5e8b97a2a082        5 weeks ago         114MB
ubuntu              latest              113a43faa138        5 weeks ago         81.2MB

C:\Users\os
λ docker tag ubuntu:16.04 coffeecoder/testupload:v1.0

C:\Users\os
λ docker images -a
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
ubuntu                      16.04               5e8b97a2a082        5 weeks ago         114MB
coffeecoder/testupload      v1.0                5e8b97a2a082        5 weeks ago         114MB
ubuntu                      latest              113a43faa138        5 weeks ago         81.2MB
```

可以看到，我们重新为ubuntu:16.04的镜像打上标签，观察IMAGE ID可知，**同一镜像可以拥有不同的标签名**。接下来，我们利用 push 命令直接上传镜像。

```
docker push <hub-user>/<repo-name>:<tag>
```

如图，我们已经上传成功。由于之前介绍的分层存储系统，我们这里是直接对已有的ubuntu镜像进行上传，只是重新打了标签，所以真正上传的只是变化的部分。

```
C:\Users\os
λ docker push coffeecoder/testupload:v1.0
The push refers to repository [docker.io/coffeecoder/testupload]
2de391e51d73: Mounted from library/ubuntu
d73dd9e65295: Mounted from library/ubuntu
686245e78935: Mounted from library/ubuntu
d7ff1dc646ba: Mounted from library/ubuntu
644879075e24: Mounted from library/ubuntu
v1.0: digest: sha256:689aa49d87d325f951941d789f7f7c8fae3394490cbcf084144caddba9c1be12 size: 1357
```

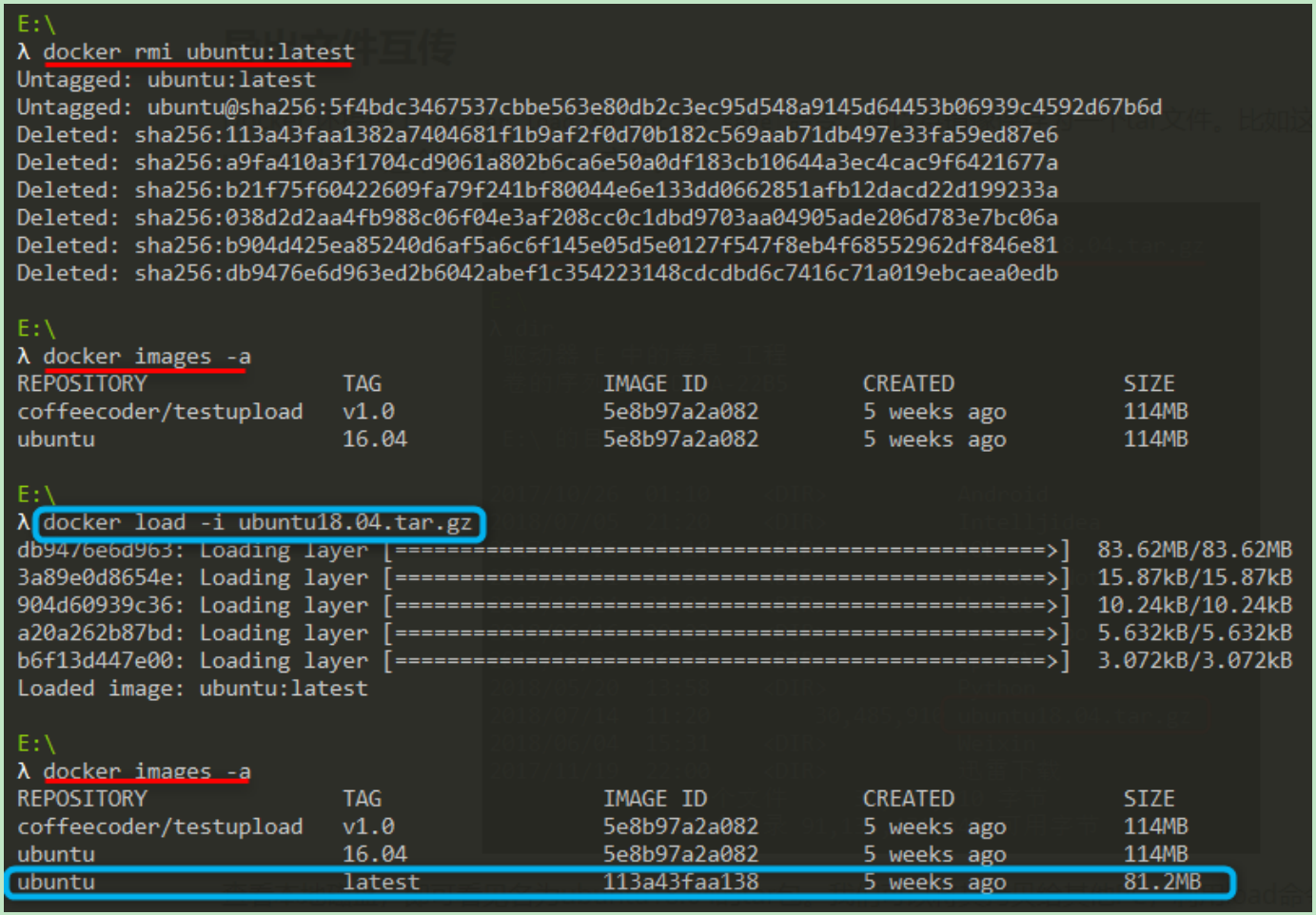


导出文件互传

Docker 还提供了 `docker load` 和 `docker save` 命令，用以将镜像保存为一个tar文件。比如这次我们将ubuntu:latest这个镜像保存为tar文件。

```
E:\
λ docker save ubuntu:latest | gzip > ubuntu18.04.tar.gz

E:\
λ dir
 驱动器 E 中的卷是 工程
 卷的序列号是 D80A-22B5

 E:\ 的目录

2017/10/26  01:10    <DIR>          Android
2018/07/05  21:20    <DIR>          Intelljidea
2017/10/26  21:11    <DIR>          LOL
2017/10/31  21:59    <DIR>          MarkdownNote
2017/10/24  21:04    <DIR>          Matlab
2018/06/16  20:22    <DIR>          maven_repo
2017/10/23  16:35    <DIR>          OpenCV
2018/05/20  13:58    <DIR>          Python
2018/07/14  11:20        30,485,910 ubuntu18.04.tar.gz
2018/06/04  15:31    <DIR>          Weixin
2017/11/19  22:00    <DIR>          迅雷下载
               1 个文件     30,485,910 字节
              10 个目录 91,136,471,040 可用字节
```

查看本地磁盘，即可看见名为ubuntu18.04的tar包。我们可以将其拷贝给其他PC，利用load命令重新导入。



```
E:\
λ docker rmi ubuntu:latest
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:5f4bdc3467537cbbe563e80db2c3ec95d548a9145d64453b06939c4592d67b6d
Deleted: sha256:113a43faa1382a7404681f1b9af2f0d70b182c569aab71db497e33fa59ed87e6
Deleted: sha256:a9fa410a3f1704cd9061a802b6ca6e50a0df183cb10644a3ec4cac9f6421677a
Deleted: sha256:b21f75f60422609fa79f241bf80044e6e133dd0662851afb12dacd22d199233a
Deleted: sha256:038d2d2aa4fb988c06f04e3af208cc0c1dbd9703aa04905ade206d783e7bc06a
Deleted: sha256:b904d425ea85240d6af5a6c6f145e05d5e0127f547f8eb4f68552962df846e81
Deleted: sha256:db9476e6d963ed2b6042abef1c354223148cdcdbd6c7416c71a019ebcaea0edb

E:\
λ docker images -a
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
coffeecoder/testupload  v1.0                 5e8b97a2a082        5 weeks ago         114MB
ubuntu                  16.04               5e8b97a2a082        5 weeks ago         114MB

E:\
λ docker load -i ubuntu18.04.tar.gz
db9476e6d963: Loading layer [==================================================>]  83.62MB/83.62MB
3a89e0d8654e: Loading layer [==================================================>]  15.87kB/15.87kB
904d60939c36: Loading layer [==================================================>]  10.24kB/10.24kB
a20a262b87bd: Loading layer [==================================================>]  5.632kB/5.632kB
b6f13d447e00: Loading layer [==================================================>]  3.072kB/3.072kB
Loaded image: ubuntu:latest

E:\
λ docker images -a
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
coffeecoder/testupload  v1.0                 5e8b97a2a082        5 weeks ago         114MB
ubuntu                  16.04               5e8b97a2a082        5 weeks ago         114MB
ubuntu                  latest              113a43faa138        5 weeks ago         81.2MB
```

推荐阅读

- 从 0 开始了解 Docker
- 如何把 Java Web 应用放在 docker 容器中运行
- 使用 Docker 搭建前端 Java 开发环境