Docker简介

Docker安装

文档

这里使用 Centos 7的root用户演示

1、确认系统版本

2、卸载旧版本

```
yum remove docker \
                  docker-client \
                  docker-client-latest \
                  docker-common \
                  docker-latest \
                  docker-latest-logrotate \
                  docker-logrotate \
                  docker-engine
```

3、yum安装gcc相关

```
yum install gcc -y
yum install gcc-c++ -y
```

4、安装需要的软件包

```
yum install -y yum-utils
```

5、设置stable仓库

```
yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
```

6、更新yum软件包索引

```
yum makecache fast
```

7、安装docker CE

```
yum install docker-ce docker-ce-cli containerd.io -y
```

8、启动docker

```
systemctl start docker
```

9、测试

```
docker version
```

```
Client: Docker Engine - Community
 Version:           20.10.12
 API version:       1.41
 Go version:        go1.16.12
 Git commit:        e91ed57
 Built:             Mon Dec 13 11:45:41 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.12
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.12
  Git commit:       459d0df
  Built:            Mon Dec 13 11:44:05 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.12
  GitCommit:        7b11cfaabd73bb80907dd23182b9347b4245eb5d
 runc:
  Version:          1.0.2
  GitCommit:        v1.0.2-0-g52b36a2
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

```
docker run hello-world
```

```
Unable to find image 'hello-world:latest' locally
https://registry-1.docker.io/v2/library/hello-
world/manifests/sha256latest: Pulling from library/hello-world
```

```
2db29710123e: Pull complete
Digest:
sha256:97a379f4f88575512824f3b352bc03cd75e239179eea0fecc38e597b2209f49
a
Status: Downloaded newer image for 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.
    (amd64)
 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://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

10、卸载

```
systemctl stop docker
yum remove docker-ce docker-ce-cli containerd.io
rm -rf /var/lib/docker
rm -rf /var/lib/containerd
```

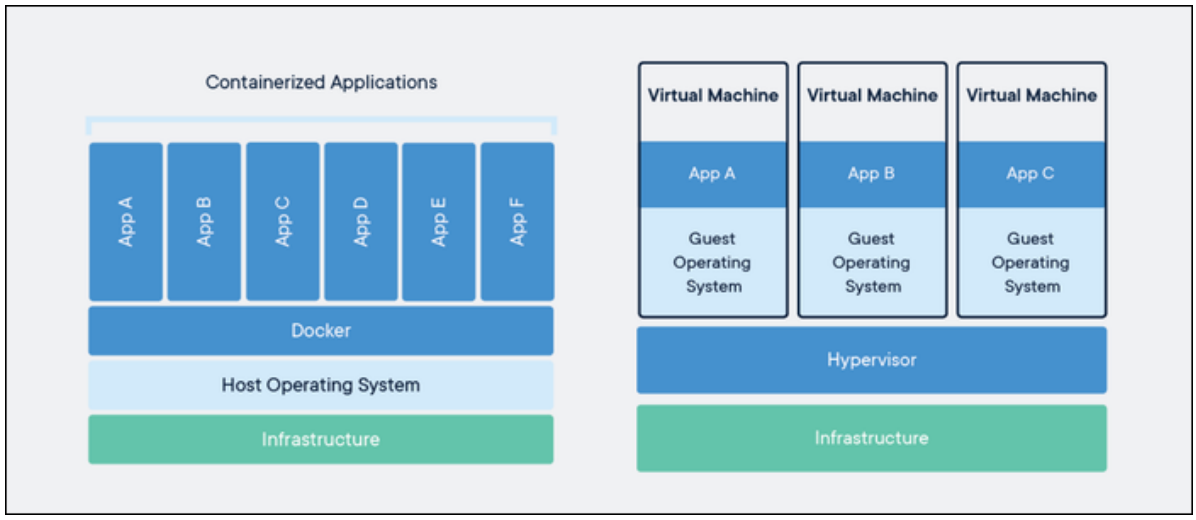
为什么Docker会比VM虚拟机快

docker有着比虚拟机更少的抽象层

由于docker不需要Hypervisor(虚拟机)实现硬件资源虚拟化,运行在docker容器上的程序直接使用的都是实际物理机的硬件资源。因此在CPU、内存利用率上docker将会在效率上有明显优势。

docker利用的是宿主机的内核,而不需要加载操作系统OS内核

当新建一个容器时,docker不需要和虚拟机一样重新加载一个操作系统内核。进而避免引寻、加载操作系统内核返回等比较费时费资源的过程,当新建一个虚拟机时,虚拟机软件需要加载OS,返回新建过程是分钟级别的。而docker由于直接利用宿主机的操作系统,则省略了返回过程,因此新建一个docker容器只需要几秒钟。



Docker常用命令

帮助启动类命令

用root用户演示，非root的注意加上`sudo`

启动docker

```
systemctl start docker
```

停止docker

```
systemctl stop docker
```

重启docker

```
systemctl restart docker
```

查看docker状态

```
systemctl status docker
```

开机自启

```
systemctl enable docker
```

查看docker概要信息

```
docker info
```

查看docker总体帮助文档

```
docker --help
```

查看docker命令帮助文档

```
docker 具体命令 --help
```

镜像命令

```
[root@simon ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    feb5d9fea6a5   5 months ago    13.3kB
```

各个选项说明：

- REPOSITORY　表示镜像的仓库源
- TAG　镜像的标签版本号
- IMAGE ID　镜像ID
- CREATED　镜像创建时间
- SIZE　镜像大小

同一个仓库源可以有多个TAG版本，代表这个仓库源的不同个版本，我们使用 REPOSITORY:TAG 来定义不同的镜像

如果你不指定一个镜像的版本标签，假如你只使用ubuntu，docker将默认使用 ubuntu:latest 镜像

docker images

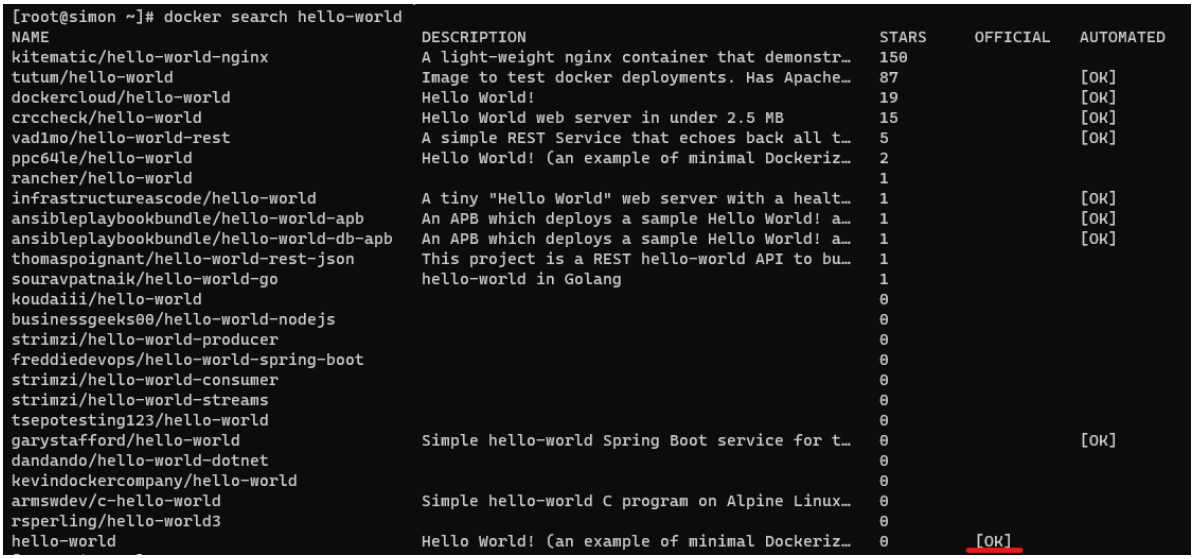
列出本地主机上的镜像

- -a　列出本地所有的镜像（含历史映像层）
- -q　只显示镜像ID

```
[root@simon ~]# docker images -a
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    feb5d9fea6a5   5 months ago    13.3kB
[root@simon ~]# docker images -q
feb5d9fea6a5
```

docker search 镜像名

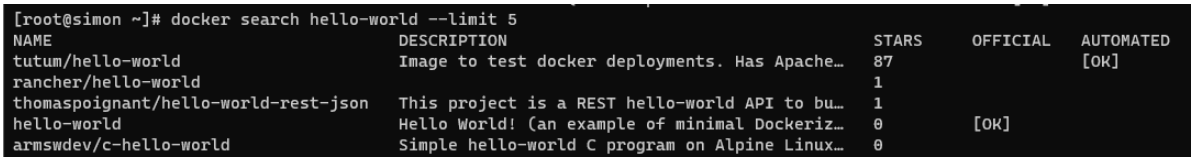
Search the Docker Hub for images



```
[root@simon ~]# docker search hello-world
NAME                                      DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
kitematic/hello-world-nginx               A light-weight nginx container that demonstr…   150
tutum/hello-world                         Image to test docker deployments. Has Apache…   87                   [OK]
dockercloud/hello-world                   Hello World!                                    19                   [OK]
crccheck/hello-world                      Hello World web server in under 2.5 MB          15                   [OK]
vad1mo/hello-world-rest                   A simple REST Service that echoes back all t…   5                    [OK]
ppc64le/hello-world                       Hello World! (an example of minimal Dockeriz…   2
rancher/hello-world                                                                       1
infrastructureascode/hello-world          A tiny "Hello World" web server with a healt…   1                    [OK]
ansibleplaybookbundle/hello-world-apb     An APB which deploys a sample Hello World! a…   1                    [OK]
ansibleplaybookbundle/hello-world-db-apb  An APB which deploys a sample Hello World! a…   1                    [OK]
thomaspoignant/hello-world-rest-json      This project is a REST hello-world API to bu…   1
souravpatnaik/hello-world-go              hello-world in Golang                           1
koudaiii/hello-world                                                                      0
businessgeeks00/hello-world-nodejs                                                        0
strimzi/hello-world-producer                                                              0
freddiedevops/hello-world-spring-boot                                                     0
strimzi/hello-world-consumer                                                              0
strimzi/hello-world-streams                                                               0
tsepotesting123/hello-world                                                               0
garystafford/hello-world                  Simple hello-world Spring Boot service for t…   0                    [OK]
dandando/hello-world-dotnet                                                               0
kevindockercompany/hello-world                                                            0
armswdev/c-hello-world                    Simple hello-world C program on Alpine Linux…   0
rsperling/hello-world3                                                                    0
hello-world                               Hello World! (an example of minimal Dockeriz…   0          [OK]
```

- NAME　镜像名称
- DESCRIPTION　镜像说明
- STARS　点赞数量
- OFFICIAL　是否是官方
- AUTOMATED　是否是自动构建的

```
--limit     只列出N个镜像，默认25个
```



```
[root@simon ~]# docker search hello-world --limit 5
NAME                                DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
tutum/hello-world                   Image to test docker deployments. Has Apache…   87                   [OK]
rancher/hello-world                                                                 1
thomaspoignant/hello-world-rest-json  This project is a REST hello-world API to bu…   1
hello-world                         Hello World! (an example of minimal Dockeriz…   0          [OK]
armswdev/c-hello-world              Simple hello-world C program on Alpine Linux…   0
```

docker pull 镜像名

下载镜像

```
docker pull 镜像名[:TAG]
```

没有TAG默认下载最新版

与`docker pull 镜像名:latest`等价

```
[root@simon ~]# docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
Digest: sha256:97a379f4f88575512824f3b352bc03cd75e239179eea0fecc38e597b2209f49a
Status: Image is up to date for hello-world:latest
docker.io/library/hello-world:latest
[root@simon ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    feb5d9fea6a5   5 months ago   13.3kB
[root@simon ~]# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
08c01a0ec47e: Pull complete
Digest: sha256:669e010b58baf5beb2836b253c1fd5768333f0d1dbcb834f7c07a4dc93f474be
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
[root@simon ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        latest    54c9d81cbb44   3 weeks ago    72.8MB
hello-world   latest    feb5d9fea6a5   5 months ago   13.3kB
[root@simon ~]# docker pull redis:6.0.8
6.0.8: Pulling from library/redis
bb79b6b2107f: Pull complete
1ed3521a5dcb: Pull complete
5999b99cee8f: Pull complete
3f806f5245c9: Pull complete
f8a4497572b2: Pull complete
eafe3b6b8d06: Pull complete
Digest: sha256:21db12e5ab3cc343e9376d655e8eabbdbe5516801373e95a8a9e66010c5b8819
Status: Downloaded newer image for redis:6.0.8
docker.io/library/redis:6.0.8
[root@simon ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
ubuntu        latest    54c9d81cbb44   3 weeks ago     72.8MB
hello-world   latest    feb5d9fea6a5   5 months ago    13.3kB
redis         6.0.8     16ecd2772934   16 months ago   104MB
```

docker system df

查看镜像/容器/数据卷所占的空间

```
[root@simon ~]# docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          3         1         177MB     177MB (99%)
Containers      1         0         0B        0B
Local Volumes   0         0         0B        0B
Build Cache     0         0         0B        0B
```

docker rmi 镜像ID

删除镜像remove image

删除单个

```
docker rmi 镜像ID
```

如果有提示需要强制卸载，那么就加个`-f`即可

```
[root@simon ~]# docker rmi 54c9d81cbb44
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:669e010b58baf5beb2836b253c1fd5768333f0d1dbcb834f7c07a4dc93f474be
Deleted: sha256:54c9d81cbb440897908abdcaa98674db83444636c300170cfd211e40a66f704f
Deleted: sha256:36ffdceb4c77bf34325fb695e64ea447f688797f2f1e3af224c29593310578d2
[root@simon ~]#
```

删除多个

```
docker rmi 镜像ID1 镜像ID2 ... 镜像IDN
```

```
[root@simon ~]# docker rmi feb5d9fea6a5 16ecd2772934
Untagged: redis:6.0.8
Untagged: redis@sha256:21db12e5ab3cc343e9376d655e8eabbdbe5516801373e95a8a9e66010c5b8819
Deleted: sha256:16ecd277293476392b71021cdd585c40ad68f4a7488752eede95928735e39df4
Deleted: sha256:3746030fff867eb26a0338ad9d3ab832e6c19c7dc008090bcfa95c7b9f16f505
Deleted: sha256:1274ec54ad17d15ec95d2180cb1f791057e86dfcdfcc18cd58610a920e145945
Deleted: sha256:18d156147e54edec9a927080fdc0a53c4a8814b0c717b36dc62e637363c1a98d
Deleted: sha256:a8f09c4919857128b1466cc26381de0f9d39a94171534f63859a662d50c396ca
Deleted: sha256:2ae5fa95c0fce5ef33fbb87a7e2f49f2a56064566a37a83b97d3f668c10b43d6
Deleted: sha256:d0fe97fa8b8cefdffcef1d62b65aba51a6c87b6679628a2b50fc6a7a579f764c
Error response from daemon: conflict: unable to delete feb5d9fea6a5 (must be forced) - image is being used by stopped container e8d9a5a2a4
26
```

删除全部**（现实中不要这样做）**

```
docker rmi $(docker images -qa)
```

虚悬镜像(dangling image)

指的是没有仓库名或没有标签的镜像

直接删掉即可

查询显示虚悬镜像

```
docker images -f dangling=true
```

删除虚悬镜像

```
docker rmi $(docker images -q -f dangling=true)
```

容器命令

这里使用ubuntu镜像演示

首先先下载一个ubuntu镜像`docker pull ubuntu`

新建/启动容器

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

OPTIONS说明（常用）

--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(后台运行)；

-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；也即启动交互式容器(前台有伪终端，等待交互)；

-P: 随机端口映射，大写P

-p: 指定端口映射，小写p

交互式容器

使用镜像ubuntu:latest以交互模式启动一个容器，在容器内执行`/bin/bash`命令，不指定的话默认为`bash`。

`docker run -it ubuntu /bin/bash`

守护进程式容器

以启动ubuntu为例，引出一个问题

```
docker run -d ubuntu
```

```
[root@simon ~]# docker run -d ubuntu
dd1ad780a47ca2d5417f6cb85c88dc76b217460a8a8bf8a8a35b750bc4efc722
[root@simon ~]# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@simon ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS                      PORTS     NAMES
dd1ad780a47c   ubuntu    "bash"    23 seconds ago   Exited (0) 21 seconds ago             elastic_hypatia
```

可以使用`docker ps -a`后可以发现，容器一经推出

这是因为docker容器后台运行，就必须有一个前台进程

容器运行的命令如果不是那些一直挂起的命令（如top，tail），就是会自动退出的

这个是docker的机制问题,比如你的web容器,我们以nginx为例，正常情况下,

我们配置启动服务只需要启动响应的service即可。例如service nginx start

但是,这样做,nginx为后台进程模式运行,就导致docker前台没有运行的应用,

这样的容器后台启动后,会立即自杀因为他觉得他没事可做了.

所以，最佳的解决方案是,将你要运行的程序以前台进程的形式运行，

常见就是命令行模式，表示我还有交互操作，别中断

运行redis则没有这个问题

```
[root@simon ~]# docker run -d redis:6.0.8
a6334b3eb0d33a5d46481736a11337f4beaa37e48355884deabb0bf95841190a
[root@simon ~]# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS      NAMES
a6334b3eb0d3   redis:6.0.8    "docker-entrypoint.s…"   6 seconds ago   Up 4 seconds   6379/tcp   frosty_williamson
[root@simon ~]#
```

列出当前所有正在运行的容器

```
docker ps [OPTIONS]
```

OPTIONS说明（常用）：

-a :列出当前所有正在运行的容器+历史上运行过的

-l :显示最近创建的容器。

-n：显示最近n个创建的容器。

-q :静默模式，只显示容器编号。

退出容器

`exit`或快捷键`ctrl+d`

run进去容器，`exit`退出，容器**停止**

快捷键`ctrl+p+q`

run进去容器，`ctrl+p+q`退出，容器**不停止**

启动已经停止的容器

```
docker start 容器ID或者容器名
```

```
[root@simon ~]# docker start 08fd2ada7e8e
08fd2ada7e8e
[root@simon ~]# docker ps -n2
CONTAINER ID   IMAGE     COMMAND       CREATED             STATUS             PORTS     NAMES
08fd2ada7e8e   ubuntu    "bash"        11 minutes ago      Up 3 seconds                 myu1
7e1b33c77f2d   ubuntu    "/bin/bash"   About an hour ago   Up About an hour             upbeat_ganguly
[root@simon ~]#
```

停止容器

```
docker stop 容器ID或者容器名
```

强制停止容器

```
docker kill 容器ID或者容器名
```

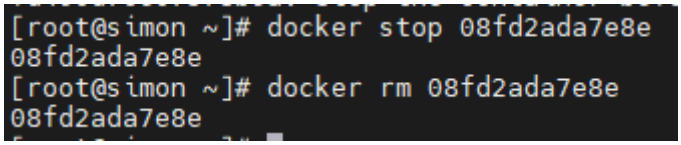
删除已经停止的容器

首先停止容器运行

```
docker stop 容器ID
```

然后删除

```
docker rm 容器ID
```



当然也可以强制删除正在运行的容器，加个`-f`

```
docker rm -f 容器ID
```

一次性删除多个容器（不要在现实中使用）

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

或

```
docker ps -a -q | xargs docker rm
```

查看容器日志

```
docker logs 容器ID
```

查看容器内运行的进程

```
docker top 容器ID
```

查看容器内部细节

```
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
```

如，`docker exec -it e1a956e0b08a /bin/bash`，必须要指定shell

重新进入

```
docker attach 容器ID
```

`exec -it`与`attach`区别

- attach直接进入容器启动命令的终端，不会启动新的进程，用exit退出会导致容器的停止

- exec实在容器中打开新的终端，并且可以启动新的进程，用exit退出不会导致容器的停止

推荐大家使用`docker exec`命令，因为退出容器终端不会导致容器的停止

从容器内拷贝文件到主机上

容器→主机

```
docker cp 容器ID:容器内路径 目的主机路径
```

```
root@e97d9bb15f6f:~# cd /tmp/
root@e97d9bb15f6f:/tmp# echo 123 > a.txt
root@e97d9bb15f6f:/tmp# [root@simon ~]# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS         PORTS     NAMES
e97d9bb15f6f   ubuntu    "bash"    6 minutes ago   Up 6 minutes             confident_engelbart
[root@simon ~]# docker cp e97d9bb15f6f:/tmp/a.txt .
[root@simon ~]# cat a.txt
123
```

导入和导出容器

导出

导出容器的内容为一个tar归档文件[对应import命令]

```
docker export 容器ID > 文件名.tar
```

```
[root@simon ~]# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS         PORTS     NAMES
e97d9bb15f6f   ubuntu    "bash"    8 minutes ago   Up 8 minutes             confident_engelbart
[root@simon ~]# docker export e97d9bb15f6f > abcd.tar
[root@simon ~]# ls -al
total 73424
dr-xr-x---.  3 root root      163 Mar  1 05:17 .
dr-xr-xr-x. 17 root root      224 Feb 24 04:34 ..
-rw-r--r--.  1 root root 75156992 Mar  1 05:17 abcd.tar
```

导入

iport从tar包中的内容创建一个新的文件系统再导入为镜像[对应export]

```
cat 文件名.tar |docker import - 镜像用户/镜像名:镜像版本号
```

```
[root@simon ~]# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS         PORTS     NAMES
e97d9bb15f6f   ubuntu    "bash"    13 minutes ago  Up 13 minutes            confident_engelbart
[root@simon ~]# cat abcd.tar |docker import - simon/ubuntu:3.7
sha256:50ee72b1d2b6d4d3d345b8d20984b50731f92832f7e608175a1be768d500706a
[root@simon ~]# docker images
REPOSITORY     TAG       IMAGE ID       CREATED          SIZE
simon/ubuntu   3.7       50ee72b1d2b6   17 seconds ago   72.8MB
postgres       latest    6a3c44872108   13 days ago      374MB
ubuntu         latest    54c9d81cbb44   3 weeks ago      72.8MB
hello-world    latest    feb5d9fea6a5   5 months ago     13.3kB
redis          6.0.8     16ecd2772934   16 months ago    104MB
[root@simon ~]# docker run -it 50ee72b1d2b6 /bin/bash
root@cd8ef9f34cec: /root@cd8ef9f34cec:/#
```

并且原本在/tmp创建的a.txt也在

```
root@cd8ef9f34cec: /root@cd8ef9f34cec:/# cat /tmp/a.txt
123
root@cd8ef9f34cec: /root@cd8ef9f34cec:/#
```

Docker镜像

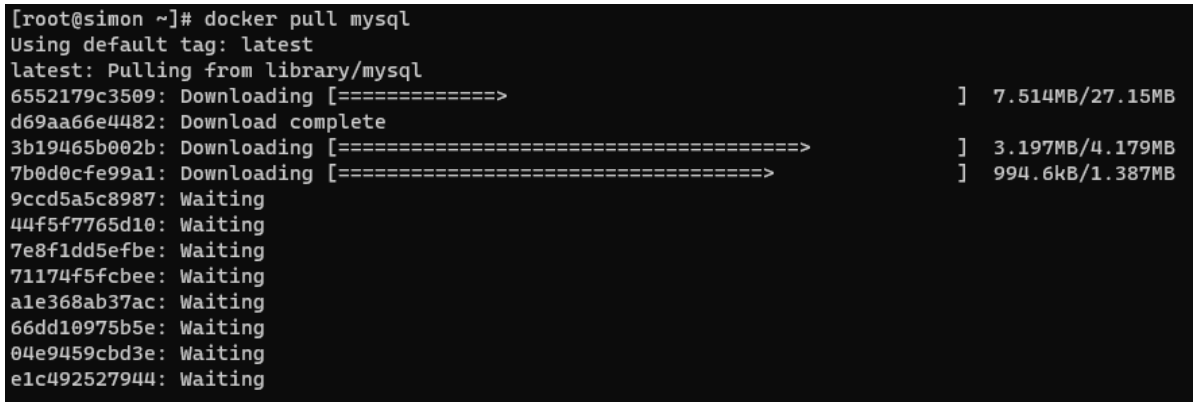
是什么

镜像是一种轻量级、可执行的独立软件包，它包含运行某个软件所需的所有内容，我们把应用程序和配置依赖打包好形成一个可交付的运行环境(包括代码、运行时需要的库、环境变量和配置文件等)，这个打包好的运行环境就是image镜像文件。

只有通过这个镜像文件才能生成Docker容器实例(类似Java中new出来一个对象)。

分层的镜像

以我们的pull为例，在下载的过程中我们可以看到docker的镜像好像是在一层一层的在下载



UnionFS

Union文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)。Union 文件系统是 Docker 镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。

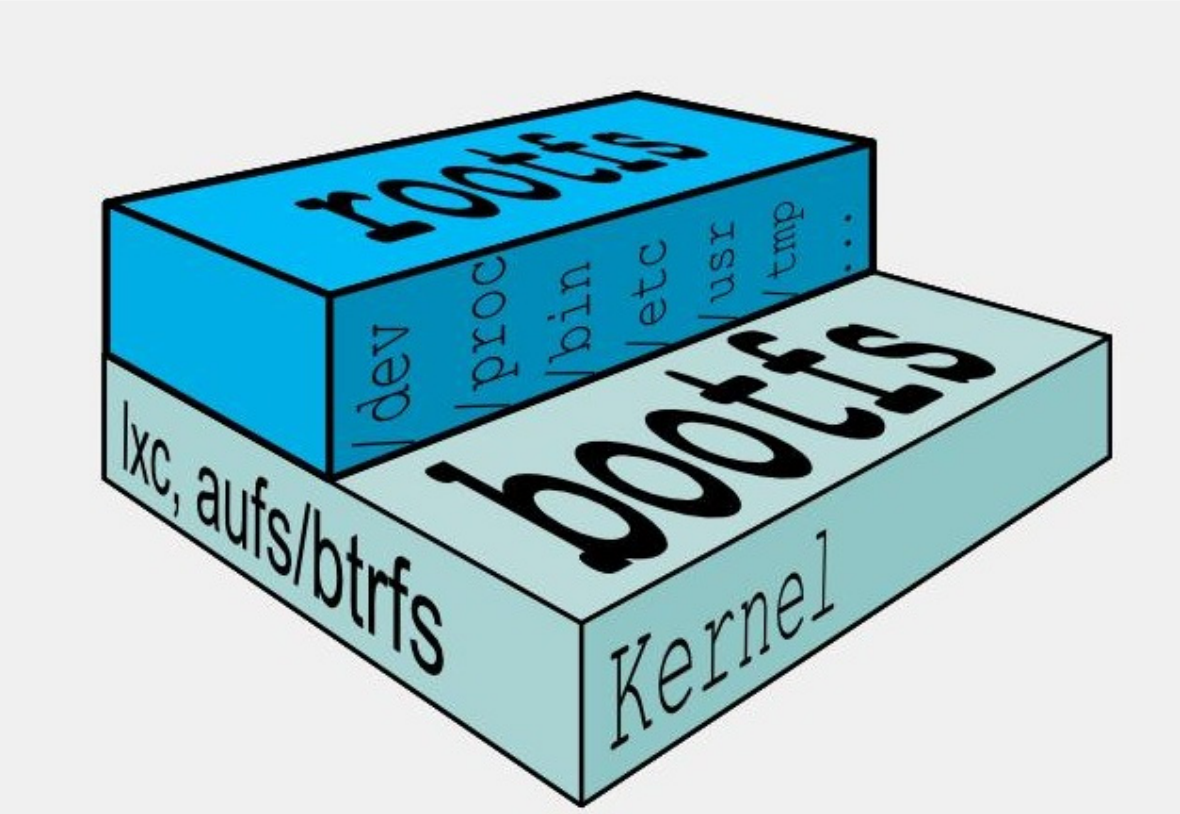
特性：一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录

Docker镜像加载原理

docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统UnionFS。

bootfs(boot file system)主要包含bootloader和kernel, bootloader主要是引导加载kernel, Linux刚启动时会加载bootfs文件系统，在Docker镜像的最底层是引导文件系统bootfs。这一层与我们典型的Linux/Unix系统是一样的，包含boot加载器和内核。当boot加载完成之后整个内核就都在内存中了，此时内存的使用权已由bootfs转交给内核，此时系统也会卸载bootfs。

rootfs (root file system)，在bootfs之上。包含的就是典型 Linux 系统中的 /dev, /proc, /bin, /etc 等标准目录和文件。rootfs就是各种不同的操作系统发行版，比如Ubuntu，Centos等等。



对于一个精简的OS，rootfs可以很小，只需要包括最基本的命令、工具和程序库就可以了，因为底层直接用Host的kernel，自己只需要提供 rootfs 就行了。由此可见对于不同的linux发行版, bootfs基本是一致的, rootfs会有差别, 因此不同的发行版可以公用bootfs。

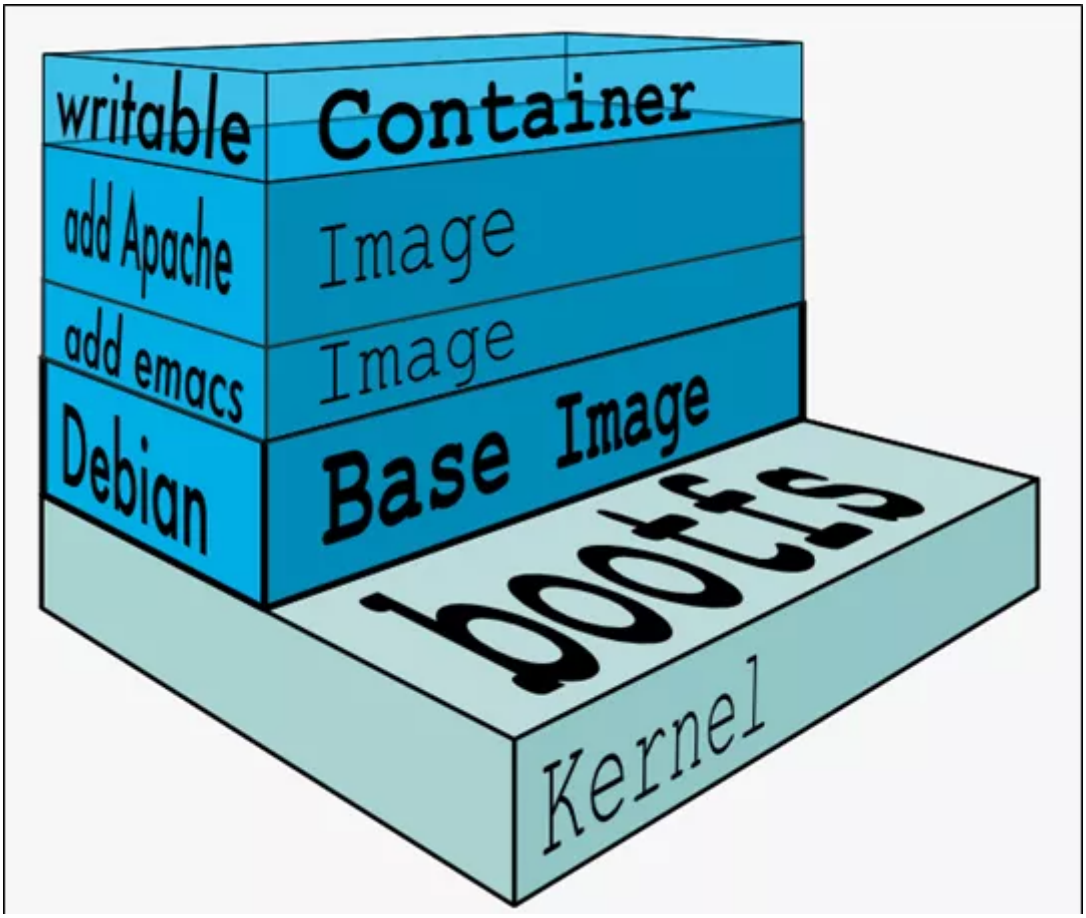
镜像分层最大的一个好处就是**共享资源，方便复制迁移，就是为了复用。**

比如说有多个镜像都从相同的 base 镜像构建而来，那么 Docker Host 只需在磁盘上保存一份 base 镜像；

同时内存中也只需加载一份 base 镜像，就可以为所有容器服务了。而且镜像的每一层都可以被共享。

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

所有对容器的改动 - 无论添加、删除、还是修改文件都只会发生在容器层中。只有容器层是可写的，容器层下面的所有镜像层都是只读的。



Docker镜像commit操作案例

`docker commit` 提交容器副本使之成为一个新的镜像

```
docker commit -m="提交的描述信息" -a="作者" 容器ID 要创建的目标镜像名:[标签名]
```

给Ubuntu安装vim为例

- 从Hub上下载ubuntu镜像到本地并成功运行
- 原始的默认ubuntu镜像是不带vim
- 外网联通的情况下，安装vim

```
apt update
apt install vim -y
```

```
root@e45b6d7efeba:/# vim a.txt
root@e45b6d7efeba:/# cat a.txt
docker!
root@e45b6d7efeba:/#
```

- 安装完成后commit自己的新镜像

```
docker commit -m="add vim" -a="simon" e45b6d7efeba simon/myubuntu:1.3
```

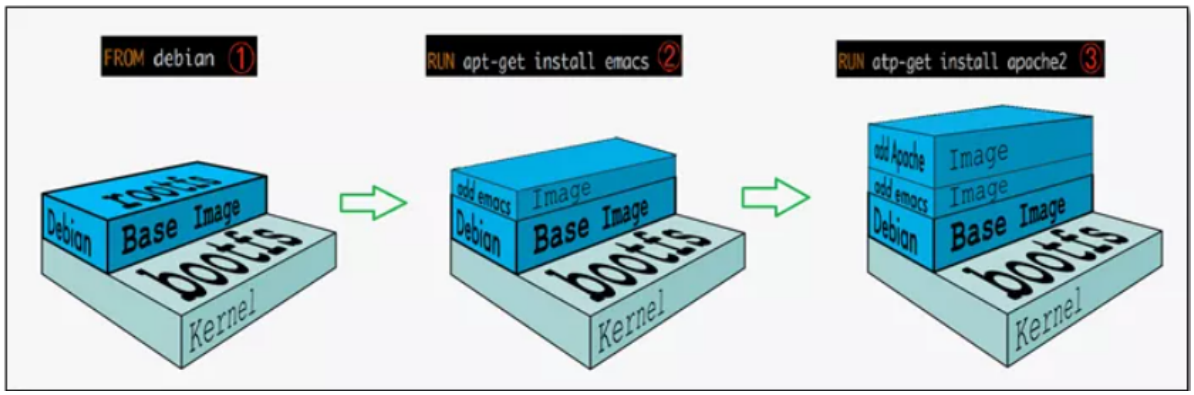
```
[root@simon ~]# docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
e45b6d7efeba   ubuntu    "/bin/bash"   22 minutes ago   Up 22 minutes             exciting_goldwasser
[root@simon ~]# docker commit -m="add vim" -a="simon" e45b6d7efeba simon/myubuntu:1.3
sha256:8fd6c9b90558d20e10b63944ec747434272e479daa2885b252dfc55cef509974
[root@simon ~]# docker images
REPOSITORY       TAG       IMAGE ID       CREATED         SIZE
simon/myubuntu   1.3       8fd6c9b90558   6 seconds ago   182MB
simon/ubuntu     3.7       50ee72b1d2b6   17 hours ago    72.8MB
mysql            latest    6126b4587b1b   5 days ago      519MB
postgres         latest    6a3c44872108   2 weeks ago     374MB
ubuntu           latest    54c9d81cbb44   3 weeks ago     72.8MB
hello-world      latest    feb5d9fea6a5   5 months ago    13.3kB
redis            6.0.8     16ecd2772934   16 months ago   104MB
[root@simon ~]#
```

- 启动新镜像并和原来的对比

```
[root@simon ~]# docker run -it ubuntu /bin/bash
root@eaaf1cf475b2:/# vim
bash: vim: command not found
root@eaaf1cf475b2:/# exit
[root@simon ~]# docker run -it 8fd6c9b90558 /bin/bash
root@7deb9413da5d:/# vim a.txt
root@7deb9413da5d:/# cat a.txt
docker!!!!
root@7deb9413da5d:/#
```

Docker中的镜像分层，支持通过扩展现有镜像，创建新的镜像。类似Java继承于一个Base基础类，自己再按需扩展。

新镜像是从 base 镜像一层一层叠加生成的。每安装一个软件，就在现有镜像的基础上增加一层



推送到仓库

什么docker hub、阿里云、腾讯云这里就不演示了，有各自命令与教程提供

推送到私有库

启动registry

Docker Registry是官方提供的工具，可以用于构建私有镜像仓库

```
docker pull registry
```

运行私有库

默认情况，仓库被创建在容器的/var/lib/registry目录下，建议自行用容器卷映射，方便于宿主机联调

```
docker run -d -p 5000:5000  -v /simon/myregistry/:/tmp/registry --
privileged=true registry
```

用在ubuntu上安装ifconfig为例演示

```
apt update
apt install net-tools -y
```

在外部执行

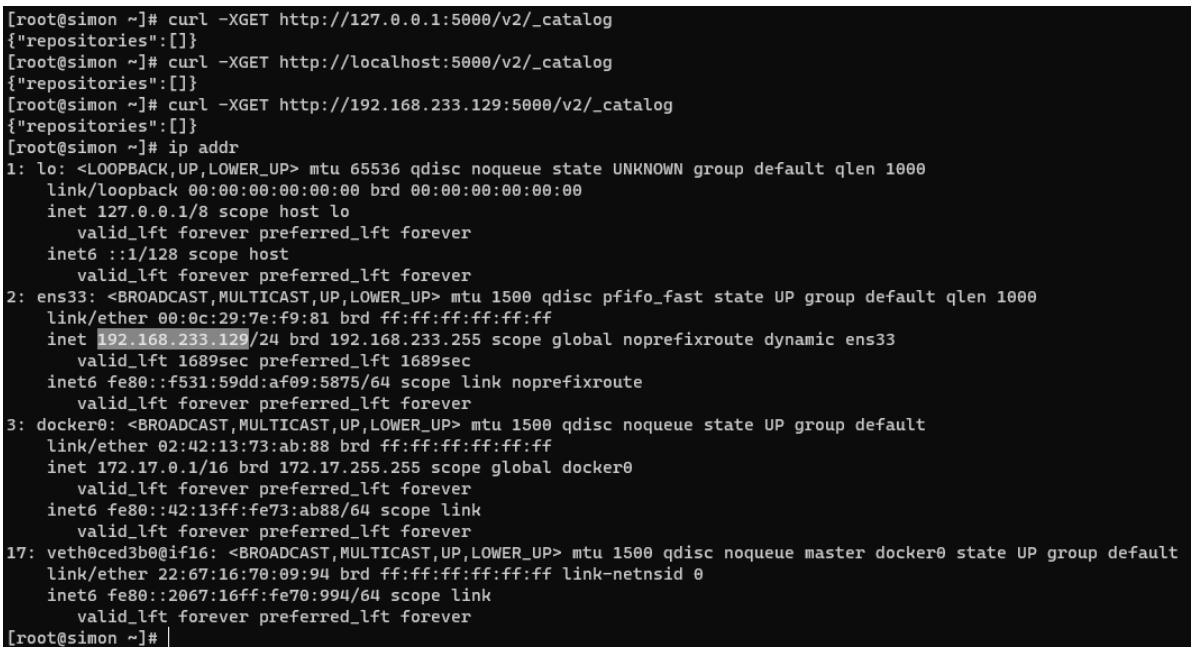
```
docker commit -m "add ifconfig" -a="simon" 99b985cf1999
simonubuntu:1.2
```

curl验证私服库上有什么镜像

发送请求，IP是宿主机的对外IP

当然，本地用的话也可以改成localhost

```
curl -XGET http://192.168.233.129:5000/v2/_catalog
```



将新镜像simonubuntu:1.2修改成符合私服规范的TAG

格式

```
docker tag 镜像:Tag Host:Port/Repository:Tag
```

示例

```
docker tag simonubuntu:1.2 192.168.233.129:5000/simonubuntu:1.2
```

```
[root@simon ~]# docker images
REPOSITORY                                                TAG       IMAGE ID       CREATED          SIZE
simonubuntu                                               1.2       c68e0f0d1cbc   10 minutes ago   132MB
registry.cn-guangzhou.aliyuncs.com/simon-canton/myubuntu  1.3       8fd6c9b90558   5 hours ago      182MB
simon/ubuntu                                              3.7       50ee72b1d2b6   22 hours ago     72.8MB
mysql                                                     latest    6126b4587b1b   5 days ago       519MB
postgres                                                  latest    6a3c44872108   2 weeks ago      374MB
registry                                                  latest    9c97225e83c8   3 weeks ago      24.2MB
ubuntu                                                    latest    54c9d81cbb44   3 weeks ago      72.8MB
hello-world                                               latest    feb5d9fea6a5   5 months ago     13.3kB
redis                                                     6.0.8     16ecd2772934   16 months ago    104MB
[root@simon ~]# docker tag simonubuntu:1.2 192.168.233.129:5000/simonubuntu:1.2
[root@simon ~]# docker images
REPOSITORY                                                TAG       IMAGE ID       CREATED          SIZE
192.168.233.129:5000/simonubuntu                          1.2       c68e0f0d1cbc   10 minutes ago   132MB
simonubuntu                                               1.2       c68e0f0d1cbc   10 minutes ago   132MB
registry.cn-guangzhou.aliyuncs.com/simon-canton/myubuntu  1.3       8fd6c9b90558   5 hours ago      182MB
simon/ubuntu                                              3.7       50ee72b1d2b6   22 hours ago     72.8MB
mysql                                                     latest    6126b4587b1b   5 days ago       519MB
postgres                                                  latest    6a3c44872108   2 weeks ago      374MB
registry                                                  latest    9c97225e83c8   3 weeks ago      24.2MB
ubuntu                                                    latest    54c9d81cbb44   3 weeks ago      72.8MB
hello-world                                               latest    feb5d9fea6a5   5 months ago     13.3kB
redis                                                     6.0.8     16ecd2772934   16 months ago    104MB
[root@simon ~]#
```

修改配置文件使之支持http协议

docker默认不允许http方式推送镜像，通过修改配置文件来取消这个限制

如果修改后不生效，尝试重启docker

```
nano /etc/docker/daemon.json
如是提示文件原本不存在，则添加
{
  "insecure-registries": ["192.168.111.162:5000"]
}
若存在则加上即可
"insecure-registries": ["192.168.111.162:5000"]
```

```
 GNU nano 2.3.1                文件:  /etc/docker/daemon.json                已更改

{
  "registry-mirrors": ["https://aa25jngu.mirror.aliyuncs.com"],
  "insecure-registries": ["192.168.233.129:5000"]
}
```

push推送到私服库

```
docker run -d -p 5000:5000  -v /simon/myregistry/:/tmp/registry --
privileged=true registry
```

```
[root@simon ~]# systemctl restart docker
[root@simon ~]# docker run -d -p 5000:5000  -v /simon/myregistry/:/tmp/registry --privileged=true registry
c933b652dbd64e0bc74a2069cf0acc88267f53c619e98c19de5b04c867f9a88b
[root@simon ~]# docker push 192.168.233.129:5000/simonubuntu:1.2
The push refers to repository [192.168.233.129:5000/simonubuntu]
0283db78a4e3: Pushing [==========================>                        ]  32.02MB/59.12MB
36ffdceb4c77: Pushing [====================>                              ]  32.01MB/72.78MB
```

再次curl验证私服库上有什么镜像

```
curl -XGET http://192.168.233.129:5000/v2/_catalog
```

```
[root@simon ~]# curl -XGET http://192.168.233.129:5000/v2/_catalog
{"repositories":["simonubuntu"]}
[root@simon ~]#
```

pull到本地并运行

先删掉本地的镜像

```
[root@simon ~]# docker images
REPOSITORY                                                TAG       IMAGE ID       CREATED          SIZE
registry.cn-guangzhou.aliyuncs.com/simon-canton/myubuntu  1.3       8fd6c9b90558   6 hours ago      182MB
simon/ubuntu                                              3.7       50ee72b1d2b6   23 hours ago     72.8MB
mysql                                                     latest    6126b4587b1b   5 days ago       519MB
postgres                                                  latest    6a3c44872108   2 weeks ago      374MB
registry                                                  latest    9c97225e83c8   3 weeks ago      24.2MB
ubuntu                                                    latest    54c9d81cbb44   3 weeks ago      72.8MB
hello-world                                               latest    feb5d9fea6a5   5 months ago     13.3kB
redis                                                     6.0.8     16ecd2772934   16 months ago    104MB
[root@simon ~]#
```

pull

```
docker pull 192.168.233.129:5000/simonubuntu:1.2
```

```
[root@simon ~]# docker pull 192.168.233.129:5000/simonubuntu:1.2
1.2: Pulling from simonubuntu
08c01a0ec47e: Already exists
e2eab536af38: Already exists
Digest: sha256:4ea182a240e83d6c9670f36e7c0d00682b9c0f678bba8fa7d40a7261ef29c9ce
Status: Downloaded newer image for 192.168.233.129:5000/simonubuntu:1.2
192.168.233.129:5000/simonubuntu:1.2
[root@simon ~]# docker images
REPOSITORY                                                TAG       IMAGE ID       CREATED          SIZE
192.168.233.129:5000/simonubuntu                          1.2       c68e0f0d1cbc   24 minutes ago   132MB
registry.cn-guangzhou.aliyuncs.com/simon-canton/myubuntu  1.3       8fd6c9b90558   6 hours ago      182MB
simon/ubuntu                                              3.7       50ee72b1d2b6   23 hours ago     72.8MB
mysql                                                     latest    6126b4587b1b   5 days ago       519MB
postgres                                                  latest    6a3c44872108   2 weeks ago      374MB
registry                                                  latest    9c97225e83c8   3 weeks ago      24.2MB
ubuntu                                                    latest    54c9d81cbb44   3 weeks ago      72.8MB
hello-world                                               latest    feb5d9fea6a5   5 months ago     13.3kB
redis                                                     6.0.8     16ecd2772934   16 months ago    104MB
[root@simon ~]# docker run -it c68e0f0d1cbc /bin/bash
root@8e2b80707229:/# ifconfig
```

Docker容器数据卷

Docker挂载主机目录访问如果出现**cannot open directory. : Permission denied**

解决办法：在挂载目录后多加一个`--privileged=true`参数即可

如果是CentOS7安全模块比之前系统版本加强，不安全的会先禁止，所以目录挂载的情况被默认为不安全的行为，

在SELinux里面挂载目录被禁止掉了额，如果要开启，我们一般使用--privileged=true命令，扩大容器的权限解决挂载目录没有权限的问题，也即

使用该参数，container内的root拥有真正的root权限，否则，container内的root只是外部的一个普通用户权限。

是什么

卷就是目录或文件，存在于一个或多个容器中，由docker挂载到容器，但不属于联合文件系统，因此能够绕过Union File System提供一些用于持续存储或共享数据的特性：

卷的设计目的就是**数据的持久化**，完全独立于容器的生存周期，因此Docker不会在容器删除时删除其挂载的数据卷

命令解析

```
docker run -d -p 5000:5000  -v /simon/myregistry:/tmp/registry --
privileged=true registry
```

-v就是指定将本机的/simon/myregistry(绝对路径)映射到镜像中的/tmp/registry

作用是将docker容器内的数据保存进宿主机的磁盘中

作用

将运用与运行的环境打包镜像，run后形成容器实例运行 ，但是我们对数据的要求希望是**持久化**的

Docker容器产生的数据，如果不备份，那么当容器实例删除后，容器内的数据自然也就没有了。

为了能保存数据在docker中我们使用卷。

特点：

- 数据卷可在容器之间共享或重用数据
- 卷中的更改可以直接实时生效
- 数据卷中的更改不会包含在镜像的更新中
- 数据卷的生命周期一直持续到没有容器使用它为止

案例

宿主与容器之间映射添加容器卷

语法

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

使用（若目录不存在，会自动创建）

```
docker run -it --privileged=true -v /tmp/host_data:/tmp/docker_data --
name=u1 ubuntu
```

在容器的`/tmp/docker_data`里面新建一个文件

返回到宿主机后查看`/tmp/host_data`

```
[root@simon ~]# docker run -it --privileged=true -v /tmp/host_data:/tmp/docker_data --name=u1 ubuntu
root@6a320c1c01b9:/# cd /tmp/docker_data/
root@6a320c1c01b9:/tmp/docker_data# touch dockerin.txt
root@6a320c1c01b9:/tmp/docker_data# ls
dockerin.txt
root@6a320c1c01b9:/tmp/docker_data# [root@simon ~]# cd /tmp/host_data
[root@simon host_data]# ls
dockerin.txt
```

在宿主机`/tmp/host_data`下新建一个文件

返回到容器查看`/tmp/docker_data`目录

```
[root@simon host_data]# touch hostin.txt
[root@simon host_data]# docker exec -it 6a320c1c01b9 /bin/bash
root@6a320c1c01b9:/# ls /tmp/docker_data/
dockerin.txt  hostin.txt
root@6a320c1c01b9:/#
```

即使宿主机在容器停止的情况下往`/tmp/host_data`目录写入数据，当容器重新启动后，也能看到最新的数据

读写规则映射添加说明

下面该命令默认是允许读与写

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

只读

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

该命令只限制容器，而不限制宿主机。意味着宿主机写入新数据后，容器能读取但不能修改

示例

```
docker run -it --privileged=true -v /mydocker/u:/tmp/u:ro --name=u2 ubuntu
```

```
[root@simon host_data]# docker run -it --privileged=true -v /mydocker/u:/tmp/u:ro --name=u2 ubuntu
root@678290d3cdc9:/# [root@simon host_data]#
[root@simon host_data]# cd /mydocker/u/
[root@simon u]# echo hell > a.txt
[root@simon u]# docker attach 678290d3cdc9
root@678290d3cdc9:/# cd /tmp/u/
root@678290d3cdc9:/tmp/u# cat a.txt
hell
root@678290d3cdc9:/tmp/u# touch b.txt
touch: cannot touch 'b.txt': Read-only file system
root@678290d3cdc9:/tmp/u#
```

卷的继承和共享

先启动一个容器

```
docker run -it --privileged=true -v /mydocker/u:/tmp/u --name=u1
ubuntu
```

容器2继承容器1的卷规则

语法

```
docker run -it --privileged=true --volumes-from 父类 --name u2 ubuntu
```

示例

```
docker run -it --privileged=true --volumes-from u1 --name u2 ubuntu
```

```
[root@simon ~]# docker run -it --privileged=true --volumes-from u1 --name u2 ubuntu
root@a4bc0e1f88c6:/# cd /tmp/u/
root@a4bc0e1f88c6:/tmp/u# ls -l
total 4
-rw-r--r--. 1 root root 5 Mar  2 14:39 a.txt
-rw-r--r--. 1 root root 0 Mar  2 14:44 hosts.txt
-rw-r--r--. 1 root root 0 Mar  2 14:43 u1data.txt
root@a4bc0e1f88c6:/tmp/u# touch u2data.txt
root@a4bc0e1f88c6:/tmp/u# ls -l
total 4
-rw-r--r--. 1 root root 5 Mar  2 14:39 a.txt
-rw-r--r--. 1 root root 0 Mar  2 14:44 hosts.txt
-rw-r--r--. 1 root root 0 Mar  2 14:43 u1data.txt
-rw-r--r--. 1 root root 0 Mar  2 14:46 u2data.txt
root@a4bc0e1f88c6:/tmp/u#
```

在容器u1上查看能否获取新数据

```
root@ac34b51cfeb6:/tmp/u# ll
total 4
drwxr-xr-x. 2 root root 72 Mar  2 14:46 ./
drwxrwxrwt. 1 root root 15 Mar  2 14:43 ../
-rw-r--r--. 1 root root  5 Mar  2 14:39 a.txt
-rw-r--r--. 1 root root  0 Mar  2 14:44 hosts.txt
-rw-r--r--. 1 root root  0 Mar  2 14:43 u1data.txt
-rw-r--r--. 1 root root  0 Mar  2 14:46 u2data.txt
root@ac34b51cfeb6:/tmp/u#
```

尝试停止u1后在宿主机与u2添加数据

```
root@ac34b51cfeb6:/tmp/u# exit
[root@simon u]# docker stop ac34b51cfeb6
ac34b51cfeb6
[root@simon u]# touch host2.txt
[root@simon u]#
```

```
root@a4bc0e1f88c6:/tmp/u# ls -l
total 4
-rw-r--r--. 1 root root 5 Mar  2 14:39 a.txt
-rw-r--r--. 1 root root 0 Mar  2 14:49 host2.txt
-rw-r--r--. 1 root root 0 Mar  2 14:44 hosts.txt
-rw-r--r--. 1 root root 0 Mar  2 14:43 u1data.txt
-rw-r--r--. 1 root root 0 Mar  2 14:46 u2data.txt
root@a4bc0e1f88c6:/tmp/u#
```

成功，可以得出结论，u2继承的只是规则与u1是否启动毫无关系

尝试启动u1

```
[root@simon u]# docker start ac34b51cfeb6
ac34b51cfeb6
[root@simon u]# docker exec -it ac34b51cfeb6 /bin/bash
root@ac34b51cfeb6:/# cd /tmp/u/
root@ac34b51cfeb6:/tmp/u# ls -l
total 4
-rw-r--r--. 1 root root 5 Mar  2 14:39 a.txt
-rw-r--r--. 1 root root 0 Mar  2 14:49 host2.txt
-rw-r--r--. 1 root root 0 Mar  2 14:44 hosts.txt
-rw-r--r--. 1 root root 0 Mar  2 14:43 u1data.txt
-rw-r--r--. 1 root root 0 Mar  2 14:46 u2data.txt
root@ac34b51cfeb6:/tmp/u#
```

能看到新增的文件