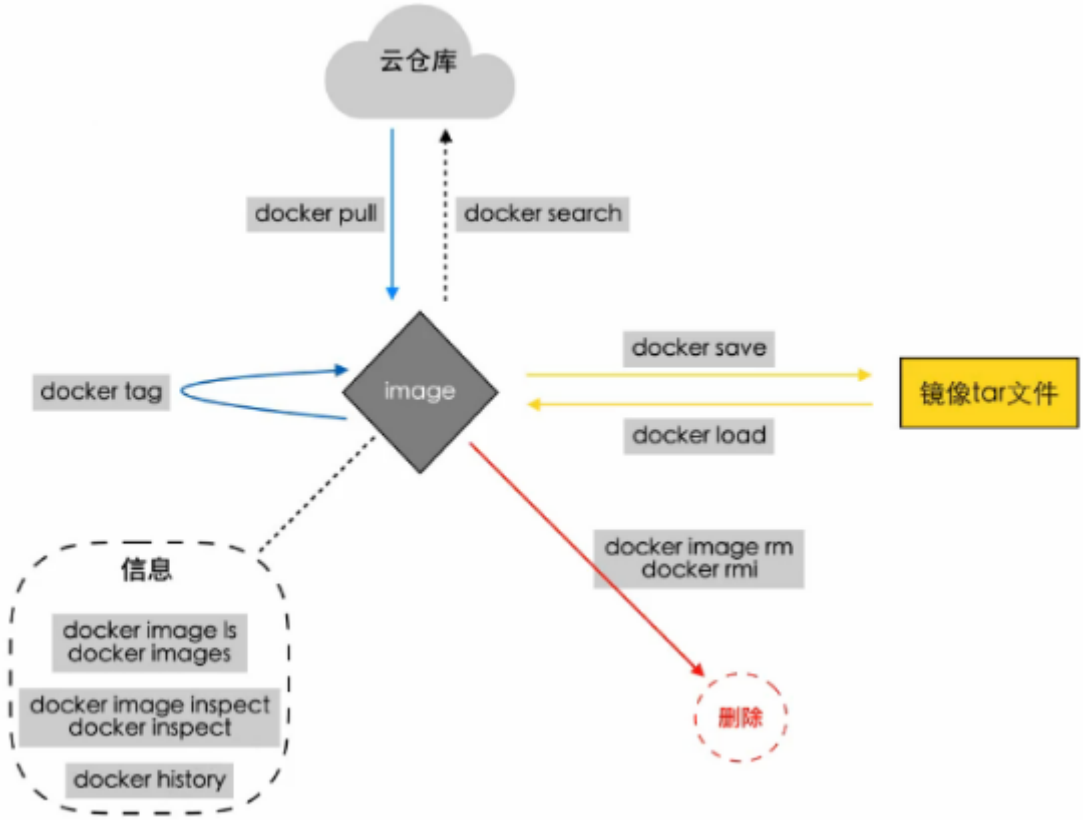
Docker语句

查看命令的作用和参数可以在命令后面加上 **--help 或 -h**(有时候不行,可能是-h是作为命令中的参数存在)

镜像操作



镜像搜索(搜索Docker Hub(镜像仓库)上的镜像)

docker search [OPTIONS] TERM

命令参数(OPTIONS):

-f, --filter filter 根据提供的格式筛选结果

--format string 利用Go语言的format格式化输出结果

--limit int 展示最大的结果数,默认25个

--no-trunc 内容全部显示

```
root@ubuntu:~# docker search mysql
NAME   DESCRIPTION                                     STARS  OFFICIAL AUTOMATED
mysql  MySQL is a widely used, open-source relati...  7943   [OK]
```

OFFICIAL 表示是否是官方镜像

TODO AUTOMATED

注意:

1.使用参数limit时,是先执行limit,然后再limit结果中执行filter,与正常思维有点不同

镜像查看(列出本地镜像)

docker images [OPTIONS] [REPOSITORY[:TAG]]

或者 docker image ls [OPTIONS] [REPOSITORY[:TAG]]

命令参数(OPTIONS):

-a, --all 展示所有镜像(默认隐藏底层的镜像)

--no-trunc 不缩略显示

-q, --quiet 只显示镜像ID

```
root@ubuntu:~# docker image ls
REPOSITORY                  TAG      IMAGE ID        CREATED       SIZE
centos/python-35-centos7    latest   72949404c84a    6 weeks ago   641 MB
mysql                       5.7.25   e47e309f72c8    6 weeks ago   372 MB
```

注意:

1.相同镜像IMAGE ID相同, 不相同镜像IAMGE ID不同

2.可以使用*进行模糊匹配,如docker images ubu*,若有两个匹配的结果该命令会失效,应该在搜索后面加上版本号区别,如docker images ubu*:latest

镜像下载(下载远程仓库(如Docker Hub)中的镜像)

docker pull [OPTIONS] NAME:[:TAG|@DIGEST]

命令参数(OPTIONS):

-a, --all-tags 下载所有符合给定tag的镜像

```
root@ubuntu:~# docker pull elasticsearch
```

注意:

1.下载镜像不存在的版本会报错

2.下载镜像默认是下载latest最新版

3.下载镜像是根据IMAGE ID来判断下不下载的,如果本地有该镜像,就不从远程下载该镜像

镜像删除(将本地的一个或多个镜像删除)

docker rmi [OPTIONS] IMAGE [IMAGE...]

或者 docker image rm [OPTIONS] IMAGE [IMAGE...]

命令参数(OPTIONS):

-f, --force 强制删除

```
root@ubuntu:~# docker rmi -f centos/python-35-centos7
Untagged: centos/python-35-centos7:latest
Untagged: centos/python-35-centos7@sha256:c3269ddaf184b686d615778ed0f11a31da8bbcdaf5091f
```

```
Deleted: sha256:72949404c84a940bcdc344492dd367acfc7cd8024cf99fb21ce897326fd89271
```

注意:

1.删除的时候可以通过输入IMAGE ID来删除镜像,可以写部分的IMAGE ID,只要这部分的ID能在本地唯一识别该镜像就可以

2.若有多个引用指向这个镜像,每次删除只是删除其中的一个引用,直到一个引用删除后,才会在本地删除该镜像

3.当镜像上有容器时,不能删除该镜像,需加上参数-f

镜像保存备份(将本地的一个或多个镜像打包保存成本地tar文件(输出到STDOUT))

docker save [OPTIONS] IMAGE [IMAGE...]

命令参数(OPTIONS):

-o, --output string 指定写入的文件名和路径

```
root@ubuntu:~# docker save mysql:5.7.25 > mysql.tar
root@ubuntu:~# docker save mysql:5.7.25 -o mysql.tar
```

注意:

1.导出多个镜像时要指定重定向到哪个包或加-o参数然后指定打包到哪个包,否则会报错

2.save命令打包时要注意使用IMAGE ID打包的镜像,在load的时候版本号和名称会丢失,所以打包时最好是使用名称加版本号

镜像备份导入(将save命令打包的镜像导入本地镜像库中)

docker load [OPTIONS]

命令参数(OPTIONS):

-i, --input string 指定要打入的文件,如没有指定,默认时STDIN

-q, --quiet 不打印导入过程信息

```
root@ubuntu:~# docker load -i mysql.tar
```

镜像重命名(对本地镜像的NAME、TAG进行重命名,并新产生一个命名后镜像)

docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]

```
root@ubuntu:~# docker tag e47e mysql:5.7.25
```

注意:

1.对于NAME和TAG为none的镜像,该命令是重命名这个镜像的名字和标签,对于有NAME和TAG的镜像是新创建一个重命名后的引用指向该镜像

镜像详细信息(查看本地一个或多个镜像的详细信息)

docker image inspect [OPTIONS] IMAGE [IMAGE...]

或者 docker inspect [OPTIONS] IMAGE [IMAGE...]

命令参数(OPTIONS):

-f, --format string 利用特定GO语言的format格式输出结果

```
# 查看镜像全部信息
root@ubuntu:~# docker image inspect mysql:5.7.25
[
    {
        "Id": "sha256:e47e309f72c831cf880cc0e1990b9c5ac427016acdc71346a36c83806ca79bb4",
        "RepoTags": [
            "mysql:5.7.25"
        ],
        ...
]

# 查看匹配的某行信息
root@ubuntu:~# docker image inspect mysql:5.7.25 | grep Created
        "Created": "2019-02-06T07:06:37.28335381Z",

# 查看匹配的某块信息
root@ubuntu:~# docker image inspect -f "{{json .RootFS.Layers}}" mysql:5.7.25
["sha256:0a07e81f5da36e4cd6c89d9bc3af643345e56bb2ed74cc8772e42ec0d393aee3","sha256:3d3393
```

注意:

1.docker inspect查看的是docker对象,而docker image inspect查看的是docker镜像

镜像历史信息(查看本地一个镜像的历史(历史分层)信息)

docker history [OPTIONS] IMAGE

命令参数(OPTIONS):

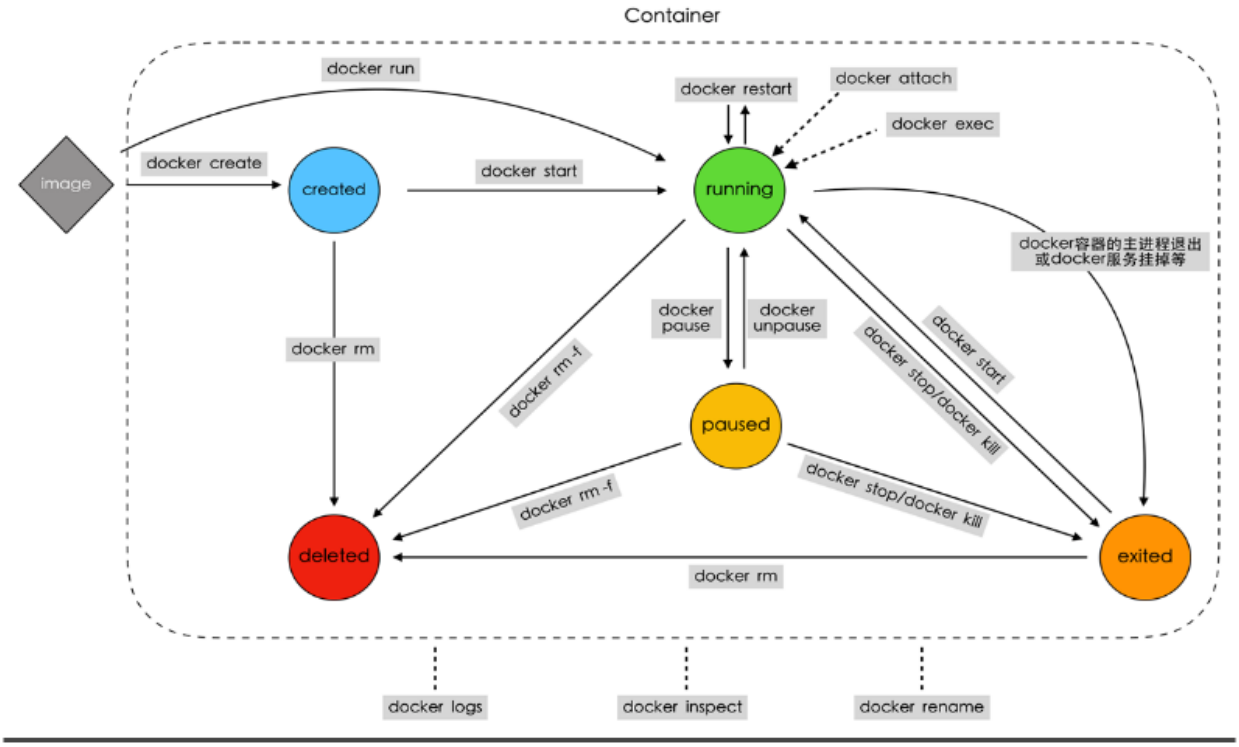
-H, --human 将创建时间、大小进行优化打印(默认为true)

-q, --quiet 只显示镜像ID

--no-trunc 不缩略显示

```
root@ubuntu:~# docker history mysql:5.7.25
IMAGE          CREATED       CREATED BY                                      SIZE  COMMENT
e47e309f72c8   6 weeks ago   /bin/sh -c #(nop)  CMD ["mysqld"]               0 B
...
```

容器操作



容器查看(列出本地容器)

docker container ls [OPTIONS]

或者 docker ps [OPTIONS]

命令参数(OPTIONS):

-a, --all 展示所有容器(默认只展示运行的容器)

--no-trunc 不缩略显示

-q, --quiet 只显示容器ID

```
# 使用container ls列出容器
root@ubuntu:~# docker container ls -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS
e33d36404bca   mysql:5.7.25   "docker-entrypoint..."   2 minutes ago   Exited (1) 2 minute

# 使用ps流出容器
root@ubuntu:~# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS
```

```
e33d36404bca  mysql:5.7.25  "docker-entrypoint..."   11 seconds ago  Exited (1) 5 seconds
```

容器创建(利用镜像创建出一个Created状态的待启动容器)

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

命令参数(OPTIONS):

-t, --tty 分配一个伪TTY,也就是分配虚拟终端

-i, --interactive 即使没有连接,也要保持STDIN打开

--name 为容器起名,如果没有指定将会随机产生一个名称

命令参数(COMMAND\ARG):

COMMAND 表示容器启动后,需要在容器中执行的命令,如ps、ls等命令

ARG 表示执行COMMAND时需要提供的一些参数,如ps命令的aux、ls命令的-a等等

```
root@ubuntu:~# docker create --name mysql -e MYSQL_ROOT_PASSWORD=mysql -p 13306:3306 mysql
69ac32f2aabee5646539c7852213746a1fb1e0cbe9c8ba263e55c3736bcb1c31
```

-e：配置信息，此处配置mysql的root用户的登陆密码

-p：端口映射，此处映射主机13306端口到容器mysql的3306端口

注意:

1.如果创建容器没有传入COMMAND\ARG,默认命令为docker inspect查看详细信息中的Cmd的值

容器删除(删除一个或多个容器)

docker container rm [OPTIONS] CONTAINER [CONTAINER...]

或者 docker rm [OPTIONS] CONTAINER [CONTAINER...]

命令参数(OPTIONS):

-f, --force 强制删除容器(会使用SIGKILL信号)

-v, --volumes 同时删除绑定在容器上的数据卷

```
# 使用container rm删除容器
root@ubuntu:~# docker container rm mysql
mysql

# 使用rm删除容器
root@ubuntu:~# docker rm 6ceb
6ceb
```

注意:

1.删除容器时CONTAINER处可以使用容器的CONTAINER ID或NAME

2.如果想删除正在运行或暂停的容器,需要加上-f参数

容器启动(将一个或多个处于创建状态或关闭状态的容器启动起来)

docker start [OPTSIONS] CONTAINER [CONTAINER...]

命令参数(OPTIONS):

-a, --attach 将当前shell的STDOUT/STDERR连接到容器上

-i, --interactive 将当前shell的STDIN连接到容器上

```
root@ubuntu:~# docker create --name python -ti python
7c19c22fca5f663fff8a71ffc052c90cfeb1a929354ebd6c6513535332d76045
root@ubuntu:~# docker start -ai python
Python 3.7.2 (default, Mar  5 2019, 06:22:51)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

注意:

1.如果想要看到容器内终端的信息,需要加上-a参数

2.如果想要在容器内的终端输入,需要加上-i参数,并且在创建容器的时候也要加上-t和-i参数

容器创建并启动(利用镜像创建并启动一个容器)

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

命令参数(OPTIONS):

-t, --tty 分配一个伪TTY,也就是分配虚拟终端

-i, --interactive 即使没有连接,也要保持STDIN打开

--name 为容器起名,如果没有指定将会随机产生一个名称

-d, --detach 在后台运行容器并打印出容器ID

--rm 当容器退出运行后,自动删除容器

命令参数(COMMAND\ARG):

COMMAND 表示容器启动后,需要在容器中执行的命令,如ps、ls等命令

ARG表示执行COMMAND时需要提供的一些参数,如ps命令的aux、ls命令的-a等等

```
root@ubuntu:~# docker run centos ls -l
total 56
-rw-r--r--   1 root root 12082 Mar  5 17:36 anaconda-post.log
lrwxrwxrwx   1 root root     7 Mar  5 17:34 bin -> usr/bin
...
```

```
root@ubuntu:~# docker run -ti python python
Python 3.7.2 (default, Mar  5 2019, 06:22:51)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

注意:

1.docker run 相当于 docker create + docker start -a (前台模式)

2.docker run -d 相当于 docker create + docker start (后台模式)

容器关闭(关闭一个或多个处于暂停状态或者运行状态的容器)

docker stop [OPTIONS] CONTAINER [CONTAINER...]

命令参数(OPTIONS):

-t, --time int 关闭前,等待的时间,单位秒(默认10s)

```
root@ubuntu:~# docker stop -t 5 a9df
a9df
```

注意:

1.一旦输入了docker stop命令,在规定的秒数内使用CTRL+C中断命令时,docker stop命令的操作并不会被中断,只要秒数到了,就会关闭容器

2.即使在关闭的状态也能输入docker stop命令,而不会报错

容器终止(强制并立即关闭一个或多个处于暂停状态或者运行状态的容器)

docker kill [OPTIONS] CONTAINER [CONTAINER...]

命令参数(OPTIONS):

-s, --signal string 指定发送给容器的关闭信号(默认"KILL"信号)

```
root@ubuntu:~# docker kill a9df
a9df
```

前提知识点:

1.Linux(终止进程信号有很多种, 可以通过kill -l 查看)其中两种终止进程的信号是:SIGTERM和SIGKILL

SIGKILL信号:无条件终止进程信号.进程接收到该信号会立即终止,不进行清理和暂存工作.该信号不能被忽略、处理和阻塞,它向系统管理员提供了可以杀死任何进程的方法

SIGTERM信号:程序终止信号.可以由kill命令产生.与SIGKILL不同的是,SIGTERM信号可以被阻塞和终止,以便程序在退出前可以保存工作或清理临时文件等

```
# 本次操作需开启两个终端
# SIGTERM的情况
    # 终端1
    root@ubuntu:~# python2.7
    Python 2.7.12 (default, Nov 12 2018, 14:36:49)
    [GCC 5.4.0 20160609] on linux2
    Type "help", "copyright", "credits" or "license" for more information.
    >>> 已终止

    # 终端2
    root@ubuntu:~# kill 10255

# SIGKILL的情况
    # 终端1
    root@ubuntu:~# python2.7
    Python 2.7.12 (default, Nov 12 2018, 14:36:49)
    [GCC 5.4.0 20160609] on linux2
    Type "help", "copyright", "credits" or "license" for more information.
    >>> 已杀死

    # 终端2
    root@ubuntu:~# kill -9 10619
```

docker stop 和 docker kill的区别

1.docker stop 会先发出SIGTERM信号给进程,告诉进程即将会被关闭.在-t指定的等待时间过了之后,将会立即发出SIGKILL信号,直接关闭容器

2.docker kill 直接发出SIGKILL信号关闭容器.但也可以通过-s参数修改发出的信号

3.因此会发现在docker stop的等待过程中,如果终止docker stop的执行,容器最终没有被关闭.而docker kill几乎是立刻发生,无法撤销

4.此外还有些异常原因也会导致容器被关闭,比如docker daemon重启、容器内部进程运行发生错误等等"异常原因"

容器暂停(暂停一个或多个处于运行状态的容器)

docker pause CONTAINER [CONTAINER...]

```
root@ubuntu:~# docker pause a9df
a9df
root@ubuntu:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS                     PORTS   NAMES
a9dff49cd21f   python    "python"    25 minutes ago   Up 22 seconds (Paused)             gracious
```

注意:

1.对已暂停的容器使用docker pause命令,会报错

容器取消暂停(取消一个或多个处于暂停状态的容器,恢复运行)

docker unpause CONTAINER [CONTAINER...]

```
root@ubuntu:~# docker unpause a9df
a9df
root@ubuntu:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS          PORTS   NAMES
a9dff49cd21f   python    "python"    25 minutes ago   Up 33 seconds           gracious_sinoussi
```

注意:

1.对运行中的容器使用docker unpause命令,会报错

容器重启(重启一个或多个处于运行状态、暂停状态、关闭状态或者新建状态的容器该命令相当于stop和start命令的结合)

docker restart [OPTIONS] CONTAINER [CONTAINER...]

命令参数(OPTIONS):

-t, --time int 重启前,等待的时间,单位秒(默认10s)实则是关闭前等待的时间

```
root@ubuntu:~# docker restart a9df
a9df
```

容器详细信息(查看本地一个或多个容器的详细信息)

docker container inspect [OPTIONS] CONTAINER [CONTAINER...]

或者 docker inspect [OPTIONS] CONTAINER [CONTAINER...]

命令参数(OPTIONS):

-f, --format string 利于特定Go语言的format格式输出结果

-s, --size 显示总大小

```
# 展示容器所有详细信息
root@ubuntu:~# docker container inspect a9df
[
    {
        "Id": "a9dff49cd21f330cf56cb652ed7a6c7c124a69810a64e95d332b4ecd2fdfa72f",
        ...
]

# 展示容器部分详细信息
 root@ubuntu:~# docker container inspect -f "{{json .State}}" a9df
{"Status":"running","Running":true,"Paused":false,"Restarting":false,"OOMKilled":false,"D
```

容器日志信息(查看容器的日志信息)

docker logs [OPTIONS] CONTAINER

命令参数(OPTIONS):

--details 显示日志的额外信息

-f, --follow 动态跟踪显示日志信息

--since string 只显示某事件时间节点之后的

--tail string 显示倒数的行数(默认全部)

-t, --timeestamps 显示timestamps时间

--until string 只显示某事时间节点之前的

```
root@ubuntu:~# docker logs -t a9df
2019-03-23T07:55:20.354197102Z Python 3.7.2 (default, Mar  5 2019, 06:22:51)
2019-03-23T07:55:20.354246858Z [GCC 6.3.0 20170516] on linux
2019-03-23T07:55:20.354257873Z Type "help", "copyright", "credits" or "license" for more
2019-03-23T07:55:49.523495592Z >>> 2019-03-23T08:08:11.539027347Z Python 3.7.2 (default,
2019-03-23T08:08:11.539081445Z [GCC 6.3.0 20170516] on linux
2019-03-23T08:08:11.539139961Z Type "help", "copyright", "credits" or "license" for more
2019-03-23T08:08:28.483313520Z >>> 2019-03-23T08:20:14.316609653Z Python 3.7.2 (default,
```

注意:

1.容器日志中记录的是容器主进程的输出STDOUT\STDERR,子进程的输出是不会记录的

容器重命名(修改容器的名称)

docker rename CONTAINER NEW_NAME

```
root@ubuntu:~# docker rename a9df python
root@ubuntu:~# docker ps -a
CONTAINER ID   IMAGE    COMMAND    CREATED        STATUS        PORTS   NAMES
a9dff49cd21f   python   "python"   18 hours ago   Up 17 hours           python
```

容器连接(将当前终端的STDIN、STDOUT、STDERR绑定到正在运行的容器的主进程上实现连接)

docker attach [OPTIONS] CONTAINER

命令参数(OPTIONS):

--no-stdin 不绑定STDIN

```
# 连接到centos容器中
root@ubuntu:~# docker run -dti centos bash
fbbc7b4b98c8fb59295220fa171a221625195867c1922261bd9c20c68dcd1950
root@ubuntu:~# docker attach fbbc
[root@fbbc7b4b98c8 /]# ps -A
   PID TTY          TIME CMD
     1 ?        00:00:00 bash
    16 ?        00:00:00 ps
[root@fbbc7b4b98c8 /]# exit
exit
root@ubuntu:~# docker ps -a
CONTAINER ID   IMAGE    COMMAND   CREATED         STATUS                     PORTS   NAMES
fbbc7b4b98c8   centos   "bash"    6 minutes ago   Exited (0) 5 minutes ago           trusting_l

# 连接到python容器中
root@ubuntu:~# docker attach a9df
>>> print('hello docker')
hello docker
>>> quit()
root@ubuntu:~# docker ps -a
CONTAINER ID   IMAGE    COMMAND    CREATED        STATUS                     PORTS   NAMES
a9dff49cd21f   python   "python"   18 hours ago   Exited (0) 11 seconds ago          python
```

注意:

1.容器的状态必须是运行状态

2.若要在进程中输入命令,该运行的容器在创建时需加上-ti参数

容器中执行新命令(在容器中运行一个命令)

docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

命令参数(OPTIONS):

-d, --detach 后台运行命令

-i, --interactive 即使没连接容器,也将当前的STDIN绑定上

-t, --tty 分配一个虚拟终端

-w, --workdir string 指定在容器中的工作目录

-e, --env list 设置容器中运行时的环境变量

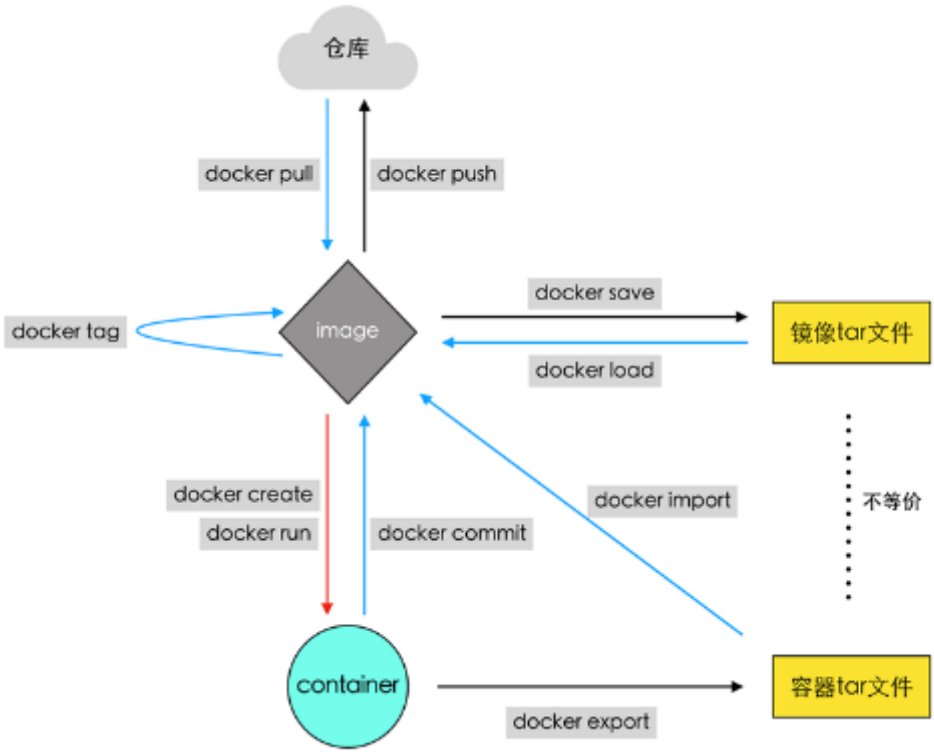
```
# 在centos容器中执行bash命令
root@ubuntu:~# docker exec -ti fbbc bash
[root@fbbc7b4b98c8 /]# ps -A
   PID TTY          TIME CMD
     1 ?        00:00:00 bash
    15 ?        00:00:00 bash
    30 ?        00:00:00 ps
[root@fbbc7b4b98c8 /]# exit
exit
root@ubuntu:~# docker ps -a
CONTAINER ID  IMAGE   COMMAND   CREATED         STATUS             PORTS   NAMES
fbbc7b4b98c8  centos  "bash"    17 minutes ago  Up About a minute          trusting_bose

# 在python容器中执行bash命令
root@ubuntu:~# docker exec -ti python bash
root@a9dff49cd21f:/# ps -A
   PID TTY          TIME CMD
     1 ?        00:00:00 python
    23 ?        00:00:00 bash
    29 ?        00:00:00 ps
root@a9dff49cd21f:/# exit
exit
root@ubuntu:~# docker ps -a
CONTAINER ID  IMAGE   COMMAND   CREATED        STATUS             PORTS  NAMES
a9dff49cd21f  python  "python"  19 hours ago   Up About a minute         python
```

注意:

1.docker exec和docker attach区别在于, docker exec是在容器内执行命令,该命令会以子进程方式运行,而docker attach连接的是主进程

容器与镜像的操作



容器提交(根据容器生成一个新的镜像)

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

命令参数(OPTIONS):

-a, --author string 作者

-c, --change list 为创建的镜像加入Dockerfile命令

-m, --message string 提交信息,类似git commit -m

-p, --pause 提交时暂停容器(defalut true)

```
root@ubuntu:~# docker run -dti centos bash
d68d9bc421ae11495feb0f4e597fad7efcc97ba1e0adbc0dc422fa5dc9101709
root@ubuntu:~# docker exec -ti d68d ifconfig
rpc error: code = 2 desc = oci runtime error: exec failed: container_linux.go:247: start
# 给容器安装net-tools
root@ubuntu:~# docker exec -ti d68d yum install net-tools
Loaded plugins: fastestmirror, ovl
Determining fastest mirrors
...
Complete!
```

```
root@ubuntu:~# docker exec -ti d68d ifconfig
```

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.3  netmask 255.255.0.0  broadcast 0.0.0.0
        inet6 fe80::42:acff:fe11:3  prefixlen 64  scopeid 0x20<link>
...
root@ubuntu:~# docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
centos       latest    9f38484d220f   9 days ago      202 MB
root@ubuntu:~# docker commit -a xiaoma -m 'install net-tools' d68d centos-net:v1.0
sha256:3361ee547702b540c0e9d84cb76644bcb45ecfa641a7084360174f1670b4aa14
root@ubuntu:~# docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
centos-net   v1.0      3361ee547702   5 seconds ago    285 MB
centos       latest    9f38484d220f   9 days ago       202 MB
```

容器导出(将容器当前的文件系统导出成一个tar文件)

docker export [OPTIONS] CONTAINER

命令参数(OPTIONS):

-o, --output string 指定写入的文件,默认是STDOUT

```
root@ubuntu:~# docker exec -ti d68d ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.3  netmask 255.255.0.0  broadcast 0.0.0.0
        inet6 fe80::42:acff:fe11:3  prefixlen 64  scopeid 0x20<link>
...
root@ubuntu:~# docker export -o net-tools-image.tar d68d
```

容器打包的导入(从一个tar文件中导入内容创建一个镜像)

docker import [OPTIONS] file|URL|- [REPOSITORY[:TAG]]

命令参数(OPTIONS):

-c, --change list 为创建的镜像加入Dockerfile命令

-m, --message string 导入时,添加提交信息

```
root@ubuntu:~# docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
centos       latest    9f38484d220f   9 days ago      202 MB
root@ubuntu:~# docker import -m '(import)install net-tools' net-tools-image.tar centos-ne
sha256:6b175a5000ac1d9233248e7718b373e428c2c4acb39b6332542414a6167a51b0
```

```
root@ubuntu:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
centos-net2   v1.0      6b175a5000ac   9 seconds ago    263 MB
centos        latest    9f38484d220f   9 days ago       202 MB
```

注意:

1.使用docker export的容器tar文件只能使用docker import导入,使用docker save的镜像tar文件只能通过docker load导入, 他们之间不能混用.

2.docker save + docker load 是生成未改变(也就是一开始导入的镜像,无论容器是否改变都没影响这个镜像)的镜像

3.dcoker export + docker import 和 docker commit 是生成的是已改变(也就是说在容器中进行了一系列操作后)的镜像

docker import和docker commit的区别:

1.通过docker history命令发现,docker commit是在原来镜像的基础上加了几层,类似于继承的,可以看到父镜像是什么,而docker import是把原来的几层和现在的几层合并到一起,形成一层新的镜像,即docker commit保留历史镜像信息,而docker import不保留历史镜像信息

```
# 原始镜像
root@ubuntu:~# docker history centos
IMAGE               CREATED             CREATED BY                                      S
9f38484d220f        9 days ago          /bin/sh -c #(nop)  CMD ["/bin/bash"]            0
<missing>           9 days ago          /bin/sh -c #(nop)  LABEL org.label-schema....   0
<missing>           9 days ago          /bin/sh -c #(nop) ADD file:074f2c974463ab3...   2
# commit的镜像
root@ubuntu:~# docker history centos-net:v1.0
IMAGE               CREATED             CREATED BY                                      S
3361ee547702        33 minutes ago      bash                                            8
9f38484d220f        9 days ago          /bin/sh -c #(nop)  CMD ["/bin/bash"]            0
<missing>           9 days ago          /bin/sh -c #(nop)  LABEL org.label-schema....   0
<missing>           9 days ago          /bin/sh -c #(nop) ADD file:074f2c974463ab3...   2
# import的镜像
root@ubuntu:~# docker history centos-net2:v1.0
IMAGE               CREATED             CREATED BY                                      SI
6b175a5000ac        15 minutes ago                                                      26
```

2.通过docker inspect命令发现,docker commit会保留原来父镜像的所有数据,并重写有更改的地方,而docker import会丢父镜像的所有元数据

```
# 原始镜像
root@ubuntu:~# docker inspect centos
...
        "Config": {
            "Hostname": "",
            "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/bash"
            ],
            "ArgsEscaped": true,
            "Image": "sha256:294e8d8145287e70f07328cc09d840fad8980b801223321b983442f097a
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {
                "org.label-schema.build-date": "20190305",
                "org.label-schema.license": "GPLv2",
                "org.label-schema.name": "CentOS Base Image",
                "org.label-schema.schema-version": "1.0",
                "org.label-schema.vendor": "CentOS"
            }
        },
...
# commit的镜像
root@ubuntu:~# docker inspect centos-net:v1.0
...
        "Config": {
            "Hostname": "",
            "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": [
                "bash"
            ],
            "Image": "",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {
                "org.label-schema.build-date": "20190305",
                "org.label-schema.license": "GPLv2",
                "org.label-schema.name": "CentOS Base Image",
                "org.label-schema.schema-version": "1.0",
                "org.label-schema.vendor": "CentOS"
            }
        },
...
# import的镜像
root@ubuntu:~# docker inspect centos-net2:v1.0
...
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
```

```
            "Env": null,
            "Cmd": null,
            "Image": "",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": null
        },
...
```

网络的操作

查看网络(查看已经建立的网络对象)

docker network ls [OPTIONS]

命令参数(OPTIONS):

-f, --filter filter 过滤条件(如'driver=bridge')

--format string 格式化打印结果

--no-trunc 不缩略显示

-q, --quiet 只显示网络对象的ID

```
root@ubuntu:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
110ce6b0275f        bridge              bridge              local
013223ba24cf        host                host                local
21a308252f8f        none                null                local
```

注意:

1.默认情况下,docker安装完成后,会自动创建bridge、host、none三种网络驱动

2.SCOPE表示网络对象的使用范围,默认下为local,表示在当前宿主机下使用

如果指定驱动是overlay,默认SCOPE是swarm,可以实现跨主机的集群服务

创建网络(创建新的网络对象)

docker network create [OPTIONS] NETWORK

命令参数(OPTIONS):

-d, --driver string 指定网络的驱动(默认"bridge")

--subnet strings 指定子网网段(如192.168.0.0/16、172.88.0.0/24)

192.168.0.0/16等于192.168.0.0～192.168.255.255

172.88.0.0/24等于172.88.0.0～172.88.0.255

/后面的数字代表子网ip的范围

--ip-range strings 执行容器的IP范围,格式同subnet参数

--gateway strings 子网的IPv4 or IPv6网关,如(192.168.0.1)

```
# 创建bridge网络
root@ubuntu:~# docker network create -d bridge my-bridge
5bc2ed61b305a4859549004c394a17a2dd52a05182670616aa5e689d4e278cda

# 创建macvlan网络
root@ubuntu:~# docker network create -d macvlan my-macvlan
08a0cf6a73bad0e9f1e65ca541eb94f8c5aa7d4ff8ae114dd77ad1dd4e2408c9
```

注意:

1.两个网络的名字不能相同,否则会报错

2.host模式网络只能存在一个

3.none模式网络只能存在一个

4.docker自带的overlay网络创建依赖于docker swarm(集群负载均衡)服务

5.容器的IP范围必须在子网网段范围内

网络删除(删除一个或多个网络)

docker network rm NETWORK [NETWORK...]

```
root@ubuntu:~# docker network rm my-macvlan
my-macvlan
```

查看网络详细信息(查看一个或多个网络的详细信息)

docker network inspect [OPTIONS] NETWORK [NETWORK...]

或者 docker inspect [OPTIONS] NETWORK [NETWORK...]

命令参数(OPTIONS):

-f, --format string 根据format输出结果

```
root@ubuntu:~# docker network inspect -f "{{json .IPAM.Config}}" my-bridge
[{"Subnet":"172.18.0.0/16","Gateway":"172.18.0.1"}]
```

使用网络(为启动的容器指定网络模式)

docker run/create --network NETWORK

```
# 使用bridge网络
root@ubuntu:~# docker run -dti --network bridge centos bash
68259b70c23aa367565afa8a10eb2e5237f96f8af4604c0698b9afed3ff00e39
root@ubuntu:~# docker exec 6825 ping baidu.com
PING baidu.com (123.125.114.144) 56(84) bytes of data.
64 bytes from 123.125.114.144 (123.125.114.144): icmp_seq=1 ttl=127 time=90.6 ms
...

# 使用host网络
root@ubuntu:~# docker run -dti --network host centos bash
f472103471ce43712ddc90e76f412a4eee267fa79d578c26d92afb8240205939
root@ubuntu:~# docker exec f472 ping baidu.com
PING baidu.com (220.181.57.216) 56(84) bytes of data.
64 bytes from 220.181.57.216 (220.181.57.216): icmp_seq=1 ttl=128 time=64.7 ms
...

# 使用Container网络模式
root@ubuntu:~# docker run -dti -p ::6379 centos
275c52223054095bf6a4b7e7fa890472ebeb2e4d0e5dd83aa3c54b67084f1c09
root@ubuntu:~# docker run -dti --network container:275c redis
0e04446af2777439660528d9f354ff480dae49c1308ae47be971ff6f681acf12
root@ubuntu:~# docker ps -a
CONTAINER ID   IMAGE    COMMAND                  CREATED          STATUS          PORTS
0e04446af277   redis    "docker-entrypoint..."   4 seconds ago    Up 3 seconds
275c52223054   centos   "/bin/bash"              32 seconds ago   Up 31 seconds   0.0.0
root@ubuntu:~# redis-cli -h 0.0.0.0 -p 32773
0.0.0.0:32773>

# 使用none网络
root@ubuntu:~# docker run -dti --network none centos bash
e706bfd5b4c5596b5d3e523c86e00d287e199f4d19625e43fcc5435ffa818661
root@ubuntu:~# docker exec e706 ping baidu.com
ping: baidu.com: Name or service not known
```

注意:

1.默认情况下,docker创建或启动容器时,会默认使用名为bridge的网络

网络连接与断开(将指定容器与指定网络进行连接或者断开连接)

docker network connect [OPTIONS] NETWORK CONTAINER

docker network disconnect [OPTIONS] NETWORK CONTAINER

命令参数(OPTIONS):

-f, --force 强制断开连接(用于disconnect)

```
root@ubuntu:~# docker run -dti centos bash
eac8d6504b2d101a4e695ca3612a490b3969d0b9536a902068cbc6345c870e11
root@ubuntu:~# docker inspect eac8
...
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "110ce6b0275ff6b17a07454d1c82feab29393219e4c751630e0dc7
                    "EndpointID": "3c0da28f9237312685e115ee5e882bfa221d6ce7e2312da11b804
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.4",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:04"
                }
            }
...
root@ubuntu:~# docker network connect my-bridge eac8
root@ubuntu:~# docker inspect eac8
...
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "110ce6b0275ff6b17a07454d1c82feab29393219e4c751630e0dc7
                    "EndpointID": "3c0da28f9237312685e115ee5e882bfa221d6ce7e2312da11b804
                    "Gateway": "172.17.0.1",
```

```
                    "IPAddress": "172.17.0.4",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:04"
                },
                "my-bridge": {
                    "IPAMConfig": {},
                    "Links": null,
                    "Aliases": [
                        "eac8d6504b2d"
                    ],
                    "NetworkID": "5bc2ed61b305a4859549004c394a17a2dd52a05182670616aa5e68
                    "EndpointID": "8d43aae3ead94bf8b18cb9237c07758ed0f1b5d2c52d7d6ebb8f4
                    "Gateway": "172.18.0.1",
                    "IPAddress": "172.18.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:12:00:02"
                }
            }
...
root@ubuntu:~# docker network disconnect my-bridge eac8
root@ubuntu:~# docker network disconnect bridge eac8
root@ubuntu:~# docker inspect eac8
...
            "Networks": {}
...
```

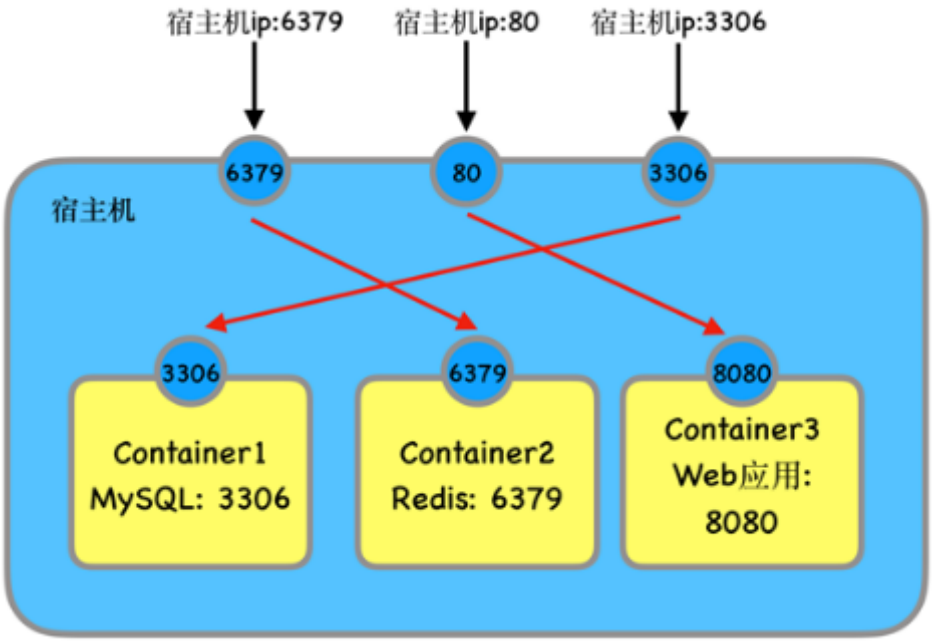
注意:

1.一个容器的bridge和macvlan可以共存,并且而可以有多个

2.none网络驱动不能和其他网络驱动模式并存

3.host、overlay不能和其他网络驱动模式并存,并且不能connect和disconnect

端口映射(启动容器时,为容器进行端口映射)



docker run -P/docker create -P

或者 docker run -p/docker create -p

命令参数(OPTIONS):

-P(大写p), --publish-all 将容器内所有暴露端口在宿主机进行随机(端口)映射

-p(小写p), --publish list 手动指定端口映射

```
# 使用大P
root@ubuntu:~# docker run -dti -P redis
dc800c870d3d5279395b747e9ef30b76ab16be2b3f05e20dff1bac332886965b
root@ubuntu:~# docker ps -a
CONTAINER ID   IMAGE   COMMAND                  CREATED         STATUS          PORTS
dc800c870d3d   redis   "docker-entrypoint..."   3 seconds ago   Up 2 seconds    0.0.0.0:

# 使用小p,不指定ip和端口
root@ubuntu:~# docker run -dti -p ::6379 redis
60072aa8818f7a04d89efa04cf9afd31af9b5e70c5e02f624dfffa6f2a970050
root@ubuntu:~# docker ps -a
CONTAINER ID   IMAGE   COMMAND                  CREATED         STATUS         PORTS
60072aa8818f   redis   "docker-entrypoint..."   4 seconds ago   Up 3 seconds   0.0.0.0:32

# 使用小p,指定ip和端口
root@ubuntu:~# docker run -dti -p 172.17.0.1:6379:6379 redis
5a4a27a64562085abbd2d4b04236fc565a6e587249195366346237db9ca63536
root@ubuntu:~# docker ps -a
CONTAINER ID  IMAGE   COMMAND                  CREATED         STATUS         PORTS
5a4a27a64562  redis   "docker-entrypoint..."   4 seconds ago   Up 3 seconds   172.17.0.1
```

注意:

1.像centos这种系统的容器是没有暴露的端口,而redis提供某个服务的容器就有暴露的端口

2.-p [HOST_IP]:[HOST_PORT]:CONTAINER_PORT

如:

-p ::80 将容器的80端口在宿主机任意IP进行随机(端口)映射(相当于指定-P(大写p))

-p :8000:6379 将容器的6379端口映射到宿主机任意IP的8000端口

-p 192.168.5.1::3306 将容器的3306端口在宿主机的192.168.5.1IP进行随机(端口)映射

不写主机IP,也就是说能够通过任何IP去访问到容器的指定服务

数据卷的操作

bind mounts方式挂载数据卷

docker run -v HOST_FILE/HOST_DIR:CONTAINER_FILE/CONTAINER_DIR IMAGE [COMMAND] [ARG...]

或者 docker run --mount type=bind,src=HOST_FILE/HOST_DIR,dst=CONTAINER_FILE/CONTAINER_DIR IMAGE [COMMAND] [ARG...]

```
# 方式一:使用-v时
root@ubuntu:~# docker run -dti -v /root/host1_dir:/root/container1_dir centos bash
f8f47ff7bf98e637d70e9627c9d6424c3edeb09e68014f030cbacd02c21b30d3
root@ubuntu:~# docker exec f8f4 touch /root/container1_dir/1.txt
root@ubuntu:~# ls host1_dir/
1.txt

# 方式二:使用--mount时
root@ubuntu:~# mkdir /root/host2_dir
root@ubuntu:~# docker run -dti --mount type=bind,src=/root/host2_dir,dst=/root/container2
d9e9ad32d8799c31e30926b2ef6a7ae099d96780277d5baf69dc1d49ee510bce
root@ubuntu:~# docker exec d9e9 touch /root/container2_dir/2.txt
root@ubuntu:~# ls host2_dir/
2.txt
```

注意:

1.使用--mount时,src指定的目录必须提前手动创建或存在,而使用-v时,src的目录不存在,则会自动创建

2.直接映射文件的时候,宿主机须有映射的文件

volumes方式挂载数据卷

docker run -v VOLUME-NAME:CONTAINER_FILE/CONTAINER_DIR IMAGE [COMMAND] [ARG...]

或者 docker run --mount type=volume,src=VOLUME-NAME,dst=CONTAINER_FILE/CONTAINER_DIR IMAGE [COMMAND] [ARG...]

```
# 方式一:使用-v
root@ubuntu:~# docker run -dti -v volume_test1:/root/container_dir centos bash
8a5167e3751422823a4e9db093ede1b9e32c57fc4578de114de870ac346952d4
root@ubuntu:~# docker inspect volume_test1
[
    {
        "CreatedAt": "2019-03-29T23:42:31+08:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/volume_test1/_data",
        "Name": "volume_test1",
        "Options": null,
        "Scope": "local"
    }
]
root@ubuntu:~# docker exec 8a51 touch /root/container_dir/1.txt
root@ubuntu:~# cd /var/lib/docker/volumes/volume_test1/_data
root@ubuntu:/var/lib/docker/volumes/volume_test1/_data# ls
1.txt

# 方式二:使用--mount
root@ubuntu:~# docker run -dti --mount type=volume,src=volume_test2,dst=/root/container_
3a5cd7bce977eeb5cd4e8a6fefbb75beb0f709d1665ba7f3aca65e276e457580
root@ubuntu:~# docker inspect volume_test2
[
    {
        "CreatedAt": "2019-03-30T00:04:30+08:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/volume_test2/_data",
        "Name": "volume_test2",
        "Options": null,
```

```
        "Scope": "local"
    }
]
root@ubuntu:~# docker exec 3a5c touch /root/container_dir/1.txt
root@ubuntu:~# cd /var/lib/docker/volumes/volume_test2/_data
root@ubuntu:/var/lib/docker/volumes/volume_test2/_data# ls
1.txt
```

注意:

1.不管使用-v还是--mount,若数据卷不存在,则会自动创建,并使用这个创建的数据卷,若存在,则使用这个存在的数据卷,而不会再次创建

2.如果命令不提供VOLUME-NAME,则会生成一个随机名字的数据卷

创建数据卷对象

docker volume create [OPTIONS] [VOLUME]

```
root@ubuntu:~# docker volume create volume_test3
volume_test3
```

查看数据卷详细信息

docker volume inspect [OPTIONS] VOLUME [VOLUME...]

或者 docker inspect [OPTIONS] VOLUME [VOLUME...]

```
# 使用volume inpect查看数据卷详细信息
root@ubuntu:~# docker volume inspect volume_test1
[
    {
        "CreatedAt": "2019-03-29T23:43:49+08:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/volume_test1/_data",
        "Name": "volume_test1",
        "Options": null,
        "Scope": "local"
    }
]

# 使用inspect查看数据卷详细信息
```

```
root@ubuntu:~# docker inspect volume_test1
[
    {
        "CreatedAt": "2019-03-29T23:43:49+08:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/volume_test1/_data",
        "Name": "volume_test1",
        "Options": null,
        "Scope": "local"
    }
]
```

查看已创建的数据卷对象

docker volume ls [OPTIONS]

```
root@ubuntu:~# docker volume ls
DRIVER              VOLUME NAME
local               volume_test1
local               volume_test2
local               volume_test3
```

删除未被使用的数据卷对象

docker volume prune [OPTIONS]

```
root@ubuntu:~# docker volume prune
WARNING! This will remove all local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
volume_test1
volume_test2
volume_test3
```

删除一个或多个数据卷对象

docker volume rm [OPTIONS] VOLUME [VOLUME...]

```
root@ubuntu:~# docker volume rm volume_test1
```

```
volume_test1
```

tmpfs mount方式挂载数据卷

docker run --mount type=tmpfs,dst=CONTAINER_FILE/CONTAINER_DIR IMAGE [COMMAND] [ARG...]

```
root@ubuntu:~# docker run -dti --mount type=tmpfs,dst=/root/container_dir centos bash
ef729deafc289b649bb814fa0c849a68b99b22601e02d58a761c92478b00eda6
```

注意:

1.tmpfs方式挂载数据卷没有src参数

共享其他容器的数据卷-数据卷容器

docker run --volumes-from CONTAINER IMAGE [COMMAND] [ARG...]

```
root@ubuntu:~# docker run -dti --mount type=volume,src=volume_test,dst=/root/container_d
c0688d647bf522e055ae6ca478e05916dc2c4b9fb1345a7ef564604a581ba14f
root@ubuntu:~# docker run -dti --volumes-from c068 centos bash
f8134f370022ae1984ec555345778c1f79ecad6039deb659543322f24d603a7b
root@ubuntu:~# docker inspect volume_test
[
    {
        "CreatedAt": "2019-03-30T11:00:09+08:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/volume_test/_data",
        "Name": "volume_test",
        "Options": null,
        "Scope": "local"
    }
]
# 在volume指定的目录下创建一个文件
root@ubuntu:/var/lib/docker/volumes/volume_test/_data# docker exec c068 touch /root/conta
root@ubuntu:~# cd /var/lib/docker/volumes/volume_test/_data
root@ubuntu:/var/lib/docker/volumes/volume_test/_data# ls
1.txt
root@ubuntu:/var/lib/docker/volumes/volume_test/_data# docker exec f813 ls /root/containe
1.txt
# 在共享数据卷下创建一个文件
```

```
root@ubuntu:/var/lib/docker/volumes/volume_test/_data# docker exec f813 touch /root/conta
root@ubuntu:/var/lib/docker/volumes/volume_test/_data# ls
1.txt  2.txt
root@ubuntu:/var/lib/docker/volumes/volume_test/_data# docker exec c068 ls /root/containe
1.txt
2.txt
```

仓库的操作

搭建无认证私有仓库

```
# 我的操作是在阿里云的服务器上设置私有仓库
root@ubuntu:~# ssh root@119.23.181.12
The authenticity of host '119.23.181.12 (119.23.181.12)' can't be established.
ECDSA key fingerprint is SHA256:751BYVKt4uYXTp8gDZBJlleCqD2IFeEWbTadPCtLi7M.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '119.23.181.12' (ECDSA) to the list of known hosts.
root@119.23.181.12's password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-117-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

Welcome to Alibaba Cloud Elastic Compute Service !

Last login: Tue Mar 26 20:38:12 2019 from 113.68.179.99
# 此时控制的是ECS服务器
# 第一步:在需要搭建仓库的服务器上安装docker,此步自行完成
# 第二步:在服务器上,从docker hub下载registry仓库
root@iZwz9f1r2pkhonvb1tc0bwZ:~# docker pull registry
Using default tag: latest
...
Status: Downloaded newer image for registry:latest
# 第三步:在服务器上,启动仓库
root@iZwz9f1r2pkhonvb1tc0bwZ:~# docker run -dti --restart always --name my-registry -p 8
0e5f52e3317da7f608daa2d89dfbf0d943aed8f9f1e90f00e2135cf8b4ee1fa8
root@iZwz9f1r2pkhonvb1tc0bwZ:~# docker ps -a
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STAT
0e5f52e3317d        registry            "/entrypoint.sh /etc…"   5 seconds ago       Up 4
# 测试私有仓库搭建是否成功,在浏览器测试(公网测试)时需要注意端口是否打开
root@iZwz9f1r2pkhonvb1tc0bwZ:~# curl 127.0.0.1:8000/v2/_catalog
{"repositories":[]}
```

注意:

1.正常情况下,当docker服务器重启的时候,所有的容器都会被关闭,但如果指定了--restart,当docker服务器重启后,容器也自动重启,不用手动去开启原来的容器

2.-p 后面的宿主机端口号可以任意指定,只要这个端口没被占用,5000端口不能变

3.-v 后面的src目录可以任意指定,但dst目录不能变

4.registry内部对外开放端口是5000.默认情况下,会将镜像存放于容器内的/var/lib/registry(官网Dockerfile中查看)目录下,这样容器被删除,则存放于容器中的镜像也会丢失,使用-v将容器内的镜像保存到宿主机中,会更安全

无认证的私有仓库--上传、下载镜像

```
# 上传
root@ubuntu:~# docker images
REPOSITORY  TAG   IMAGE ID       CREATED      SIZE
centos-net  v1.0  3361ee547702   6 days ago   285MB
# 第一步:利用docker tag重命名需要上传的镜像
root@ubuntu:~# docker tag centos-net:v1.0 119.23.181.12:8000/centos-net:v2.0
root@ubuntu:~# docker images
REPOSITORY                      TAG    IMAGE ID      CREATED      SIZE
119.23.181.12:8000/centos-net   v2.0   3361ee547702  6 days ago   285MB
# 第二步:利用docker push上传刚刚重命名的镜像
root@ubuntu:~# docker push 119.23.181.12:8000/centos-net
The push refers to repository [119.23.181.12:8000/centos-net]
Get https://119.23.181.12:8000/v2/: http: server gave HTTP response to HTTPS client
# 遇到这个错误,往配置文件/etc/docker/daemon.json里添加: (若前面有语句,需加上逗号)  "insecure-
root@ubuntu:~# docker push 119.23.181.12:8000/centos-net
The push refers to repository [119.23.181.12:8000/centos-net]
c8af0df06ff8: Pushed
d69483a6face: Pushed
v2.0: digest: sha256:32d80fd4721b1013ea6fc363ae0614bd88d7e86fde7bb01a05b4a94abb82f8bb siz

# 下载
# 若存在和当前ID相同的镜像,则不会出现下载进度条
```

```
root@ubuntu:~# docker pull 119.23.181.12:8000/centos-net:v2.0
v2.0: Pulling from centos-net
8ba884070f61: Already exists
6987f1922cf4: Pull complete
Digest: sha256:32d80fd4721b1013ea6fc363ae0614bd88d7e86fde7bb01a05b4a94abb82f8bb
Status: Downloaded newer image for 119.23.181.12:8000/centos-net:v2.0
root@ubuntu:~# docker images
REPOSITORY                       TAG    IMAGE ID       CREATED      SIZE
119.23.181.12:8000/centos-net    v2.0   3361ee547702   6 days ago   285MB
```

注意:

1.push的镜像必须重命名为服务器IP:端口/IMAGE_NAME,否则会报错

2.如果push出现了类似https的错误

原因是默认情况下,docker push是使用https协议,如果搭建仓库的服务器不是使用https协议的话,而是使用http协议,push就会失败

解决方法:

第一种方式,将https请求改成http请求:往配置文件/etc/docker/daemon.json里添加这句"insecure-registries":["服务器IP:端口"],(若前面有语句,需加上逗号) ,然后重启docker服务器

第二种方式,搭建仓库的服务器使用https协议,需要申请配置证书,自行查询网上教程

搭建带认证的私有仓库

```
# 第一步:删除先前创建的无认证的仓库容器
root@iZwz9f1r2pkhonvb1tc0bwZ:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STAT
0e5f52e3317d        registry            "/entrypoint.sh /etc…"   4 hours ago         Up 4
root@iZwz9f1r2pkhonvb1tc0bwZ:~# docker rm -f 0e5f
0e5f
# 若要保留之前上传的镜像,则无需删除此文件夹
root@iZwz9f1r2pkhonvb1tc0bwZ:~# rm -rf /my-registry/
# 第二步:创建存放认证用户名和密码的文件
root@iZwz9f1r2pkhonvb1tc0bwZ:~# mkdir /my-registry/auth -p
# 第三步:创建密码验证文件.注意将USERNAME和PASSWORD替换为设置的用户名和密码
root@iZwz9f1r2pkhonvb1tc0bwZ:~# docker run --entrypoint htpasswd registry -Bbn docker-reg
# 第四步:启动仓库镜像
root@iZwz9f1r2pkhonvb1tc0bwZ:~# docker run -d -p 8000:5000 --restart=always --name docker
add81e6efc10ddb29a55fb4959886d7c0558eee3178882cb43191b22779dc453
# 通过浏览器访问验证仓库是否搭建成功http://119.23.181.12:8000/v2/_catalog
```

注意:

1.--entrypoint 意思是使用后面的命令(htpasswd)覆盖镜像(registry)默认的命令

2.htpasswd命令 是Apache的Web服务器内置工具,用于创建和更新储存用户名、域和用户基本认证的密码文件

3.-B 强制使用bcrypt算法加密密码(非常安全)

4.-b:在命令行中一并输入用户名和密码而不是根据提示输入密码

5.-n:不更新加密文件,只将加密后的用户名密码显示在屏幕上

6.REGISTRY_AUTH 认证的文件名

7.REGISTRY_AUTH_HTPASSWD_PATH 认证文件在容器内的位置

带认证的私有仓库--上传、下载镜像

```
# 上传
root@ubuntu:~# docker images
REPOSITORY  TAG   IMAGE ID       CREATED      SIZE
centos-net  v1.0  3361ee547702   6 days ago   285MB
# 利用docker tag重命名需要上传的镜像
root@ubuntu:~# docker tag centos-net:v1.0 119.23.181.12:8000/centos-net:v2.0
root@ubuntu:~# docker images
REPOSITORY                     TAG    IMAGE ID      CREATED      SIZE
119.23.181.12:8000/centos-net  v2.0   3361ee547702  6 days ago   285MB
# 第一步:首先登录到服务器
root@ubuntu:~# docker login -u docker-registry -p xiaoma 119.23.181.12:8000
...
Login Succeeded
# 第二步:然后执行pull或者push命令
root@ubuntu:~# docker push 119.23.181.12:8000/centos-net
The push refers to repository [119.23.181.12:8000/centos-net]
c8af0df06ff8: Pushed
d69483a6face: Pushed
v2.0: digest: sha256:32d80fd4721b1013ea6fc363ae0614bd88d7e86fde7bb01a05b4a94abb82f8bb siz
# 第三步:操作完毕后,可以退出登录
root@ubuntu:~# docker logout 119.23.181.12:8000
Removing login credentials for 119.23.181.12:8000

# 下载
# 若存在和当前ID相同的镜像,则不会出现下载进度条
# 登录
root@ubuntu:~# docker login -u docker-registry -p xiaoma 119.23.181.12:8000
```

```
...
Login Succeeded
# 拉取镜像
root@ubuntu:~# docker pull 119.23.181.12:8000/centos-net:v2.0
v2.0: Pulling from centos-net
8ba884070f61: Already exists
6987f1922cf4: Pull complete
Digest: sha256:32d80fd4721b1013ea6fc363ae0614bd88d7e86fde7bb01a05b4a94abb82f8bb
Status: Downloaded newer image for 119.23.181.12:8000/centos-net:v2.0
root@ubuntu:~# docker images
REPOSITORY                        TAG    IMAGE ID      CREATED       SIZE
119.23.181.12:8000/centos-net     v2.0   3361ee547702  7 days ago    285MB
# 退出登录
root@ubuntu:~# docker logout 119.23.181.12:8000
Removing login credentials for 119.23.181.12:8000
```

注意:

1.无论是push还是pull,首先登录到服务器,然后执行pull或者push命令

Dockerfile的操作

根据dockerfile创建镜像

docker build [OPTIONS] PATH | URL | -

PATH Dockerfile所在路径(文件夹路径),文件名必须是Dockerfile

URL Dockerfile所在URL地址

命令参数(OPTIONS):

-t,--tag list 为镜像设置名称和tag

-f, --file string 指定Dockerfile的路径(这时可以使用其他名称命名Dockerfile)

```
root@ubuntu:~# mkdir dockerfile-dir
root@ubuntu:~# cd dockerfile-dir/
root@ubuntu:~/dockerfile-dir# vim Dockerfile
--------------- Dockerfile文件内部内容------------------
# 这是一个Dockerfile文件
FROM centos
RUN echo 'hello Dockerfile'
-------------------------------------------------
root@ubuntu:~/dockerfile-dir# docker build .
```

```
Sending build context to Docker daemon  2.048kB
```

```
Step 1/2 : FROM centos
 ---> 9f38484d220f
Step 2/2 : RUN echo 'hello Dockerfile'
 ---> Running in 73b4103eed5b
hello Dockerfile
Removing intermediate container 73b4103eed5b
 ---> 1945d14be3d3
Successfully built 1945d14be3d3
root@ubuntu:~/dockerfile-dir# docker images
REPOSITORY  TAG     IMAGE ID       CREATED         SIZE
<none>      <none>  1945d14be3d3   4 seconds ago   202MB
# 加上-t参数重新构建镜像
root@ubuntu:~/dockerfile-dir# docker build -t centos:v1.0 .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM centos
 ---> 9f38484d220f
Step 2/2 : RUN echo 'hello Dockerfile'
 ---> Using cache
 ---> 1945d14be3d3
Successfully built 1945d14be3d3
Successfully tagged centos:v1.0
root@ubuntu:~/dockerfile-dir# docker images
REPOSITORY  TAG    IMAGE ID      CREATED       SIZE
centos      v1.0   1945d14be3d3  7 hours ago   202MB
# 使用-f参数构建镜像
root@ubuntu:~/dockerfile-dir# mv Dockerfile dockerfile-test
root@ubuntu:~/dockerfile-dir# docker build -t centos:v2.0 -f ./dockerfile-test .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM centos
 ---> 9f38484d220f
Step 2/2 : RUN echo 'hello Dockerfile'
 ---> Using cache
 ---> 1945d14be3d3
Successfully built 1945d14be3d3
Successfully tagged centos:v2.0
root@ubuntu:~/dockerfile-dir# docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED
centos                          v1.0                1945d14be3d3        7 hours ago
centos                          v2.0                1945d14be3d3        7 hours ago
```

注意:

1.通常撰写Dockerfile,会先创建一个文件夹,在文件夹里面撰写Dockerfile文件

2.Dockerfile文件通常命名为Dockerfile,使用别的名称在docker build的时候需要使用-f参数指定,该参数后面加路径,而且不能只提供文件夹,而要提供到具体的Dockerfile文件

3.不使用-t参数,会使得创建的镜像的名字和版本号未none

4.镜像CREATED的时间就是docker build的时间

5.如果Dockerfile未做更改并且存在之前使用该Dockerfile构建的镜像,使用该命令加上-t参数重新构建镜像,可以发现并未新创建镜像,而是将原来镜像重命名

6.可以在任何路径下构建镜像,只要路径不写错

7.构建过程中的规则:第一次构建Dockerfile中的每一步都会被执行,第二次构建,如果中间的某一步发生变化,从这步开始,这一步和之后的每一步操作都会被重新执行

8.Dockerfile中的基础镜像如果在本地存在,则直接使用本地的镜像,如果不存在,则会先pull下来再拿该镜像来使用

Dockerfile命令

具体使用参考官网https://docs.docker.com/engine/reference/builder/

FROM:指定基础镜像

RUN:构建镜像过程中需要执行的命令.可以有多条.

CMD:添加启动容器时需要执行的命令.多条只有最后一条生效.可以在启动容器时被覆盖和修改

ENTRYPOINT:同CMD,但这个一定会被执行,不会被覆盖修改

LABEL:为镜像添加信息

MAINTAINER:表明镜像的作者.将被遗弃,被LABEL代替

EXPOSE:设置对外暴露的端口

ENV:设置执行命令时的环境变量,并且在构建完成后,仍然生效

ARG:设置只在构建过程中使用的环境变量,构建完成后,将消失

ADD:将本地文件或目录拷贝到镜像的文件系统中.能解压特定格式文件,能将URL作为要拷贝的文件

COPY:将本地文件或目录拷贝到镜像的文件系统中

VOLUME:添加数据卷

USER:指定以哪个用户的名义执行RUN,CMD和ENTRYPOINT等命令

WORKDIR:设置工作目录