docker入门与应用

0x01 简介

docker 是什么？docker就是一个比虚拟机轻量级很多的的一个虚拟环境。Docker 是一种容器化技术，它将应用程序及其依赖项打包到一个可移植的容器中，以便在任何地方运行。本质上来说可以理解为一个超级轻量级的虚拟机。

docker 常用概念

docker file 是一个文本文件，它包含了一系列指令，用于构建 Docker 镜像。

docker image 镜像是一个只读的模板，它包含了运行应用程序所需的所有文件、依赖项和配置信息

docker continer 容器是从镜像创建的运行实例，它包含了应用程序及其依赖项，并且可以在任何地方运行。

docker Repository 仓库是用于存储 Docker 镜像的地方，可以是公共的或私有的。

```
+----------+     +----------+      +----------+
|  docker  |---->|  docker  |----> |docker    |
|   file   |     |  image   |      |container |
+----------+     +----------+      +----------+
```

image 可以自己用dockerfile创建也可以从dockerhub上直接拉取别人的image

docker 常用命令

1. `docker run`：启动一个新的容器
2. `docker ps`：列出当前正在运行的容器
3. `docker ps -a`：列出所有的容器包括未运行的
4. `docker images`：列出本地镜像
5. `docker stop`：停止一个或多个容器
6. `docker rm`：删除一个或多个容器
7. `docker rmi`：删除一个或多个镜像
8. `docker pull`：从 Docker 镜像仓库中拉取镜像
9. `docker push`：将本地镜像推送到 Docker 镜像仓库
10. `docker exec`：在运行中的容器中执行命令

11. `docker logs`：查看容器的日志输出

这些命令只是 Docker 命令的一小部分，还有很多其他命令可以使用。可使用`docker --help`命令来查看完整的命令列表。操作镜像或者容器的时候基本都使用id，因为id具有唯一性

0x02 常见问题处理

pull 拉取慢

无法登录dockerhub

减小自己做的imge的体积

0x03 基本操作

列出镜像

```
C:\Users\L14>docker images
REPOSITORY         TAG       IMAGE ID       CREATED         SIZE
kernelubuntu       2204      480bd2a1a10b   5 hours ago     499MB
kernelubuntu       1804      318e5e80dbcf   5 hours ago     433MB
kernelubuntu       1604      2d3c656084a9   5 hours ago     466MB
myimage            16.04     5b9f3a8a4ebd   5 hours ago     117MB
ubuntu             latest    e4c58958181a   2 weeks ago     77.8MB
ubuntu             18.04     f9a80a55f492   4 months ago    63.2MB
ubuntu             16.04     b6f507652425   2 years ago     135MB
training/webapp    latest    6fae60ef3446   8 years ago     349MB
```

各个选项说明:

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

同一仓库源可以有多个 TAG，代表这个仓库源的不同个版本，如 ubuntu 仓库源里，有 15.10、14.04 等多个不同的版本，我们使用 REPOSITORY:TAG 来定义不同的镜像。

pull拉取image

`docker pull` 命令用于从 Docker 镜像仓库中拉取（下载）镜像到本地。如果本地没有指定的镜像，`docker pull` 命令会自动从 Docker 镜像仓库中下载。例如，要拉取官方的 Ubuntu 镜像，可以使用以下命令：

```
docker pull ubuntu
```

这将从 Docker 镜像仓库中下载最新版本的 Ubuntu 镜像。如果要下载特定版本的镜像，可以在镜像名称后面加上版本号，例如：

```
docker pull ubuntu:18.04
```

这将下载 Ubuntu 18.04 版本的镜像。

构建镜像

我们使用命令 **docker build**， 从零开始来创建一个新的镜像。为此，我们需要创建一个 Dockerfile 文件，其中包含一组指令来告诉 Docker 如何构建我们的镜像。`dockerfile的名字不许改也不许加尾缀`

示例dockerfile内容

```
FROM    mytestimage:1604
RUN     /bin/echo 'root:123456' |chpasswd
RUN     useradd test
RUN     /bin/echo 'test:123456' |chpasswd
RUN     /bin/echo -e "LANG=\"en_US.UTF-8\"" >/etc/default/local
EXPOSE  22
EXPOSE  80
CMD     /usr/sbin/sshd -D
```

```
C:\Users\L14>docker build -t testimage:16 .\Desktop\
[+] Building 43.6s (9/9) FINISHED
docker:default
 => [internal] load build definition from Dockerfile
4.4s
 => => transferring dockerfile: 287B
0.0s
 => [internal] load .dockerignore
3.4s
 => => transferring context: 2B
0.0s
 => [internal] load metadata for docker.io/library/mytestimage:1604
0.0s
 => [1/5] FROM docker.io/library/mytestimage:1604
1.9s
 => [2/5] RUN     /bin/echo 'root:123456' |chpasswd
4.3s
 => [3/5] RUN     useradd test
8.7s
 => [4/5] RUN     /bin/echo 'test:123456' |chpasswd
5.4s
 => [5/5] RUN     /bin/echo -e "LANG="en_US.UTF-8"" >/etc/default/local
5.1s
 => exporting to image
7.2s
 => => exporting layers
6.6s
 => => writing image
sha256:640927733e899ec8ff66159e044250295037ef02024845091eacb78aff71228a
0.2s
 => => naming to docker.io/library/testimage:16
0.4s

What's Next?
  1. Sign in to your Docker account → docker login
  2. View a summary of image vulnerabilities and recommendations → docker scout
quickview

C:\Users\L14>docker images
REPOSITORY        TAG       IMAGE ID       CREATED          SIZE
testimage         16        640927733e89   18 seconds ago   118MB
kernelubuntu      2204      480bd2a1a10b   5 hours ago      499MB
kernelubuntu      1804      318e5e80dbcf   5 hours ago      433MB
kernelubuntu      1604      2d3c656084a9   5 hours ago      466MB
myimage           16.04     5b9f3a8a4ebd   5 hours ago      117MB
mytestimage       1604      5b9f3a8a4ebd   5 hours ago      117MB
ubuntu            latest    e4c58958181a   2 weeks ago      77.8MB
ubuntu            18.04     f9a80a55f492   4 months ago     63.2MB
ubuntu            16.04     b6f507652425   2 years ago      135MB
training/webapp   latest    6fae60ef3446   8 years ago      349MB
```

- **-t**： 指定要创建的目标镜像名
- .\Desktop\： Dockerfile 文件所在目录，可以指定Dockerfile 的路径

设置镜像标签

我们可以使用 docker tag 命令，为镜像添加一个新的标签。其实就是重命名

```
C:\Users\L14>docker images
REPOSITORY        TAG       IMAGE ID       CREATED        SIZE
kernelubuntu      2204      480bd2a1a10b   5 hours ago    499MB
kernelubuntu      1804      318e5e80dbcf   5 hours ago    433MB
kernelubuntu      1604      2d3c656084a9   5 hours ago    466MB
myimage           16.04     5b9f3a8a4ebd   5 hours ago    117MB
ubuntu            latest    e4c58958181a   2 weeks ago    77.8MB
ubuntu            18.04     f9a80a55f492   4 months ago   63.2MB
ubuntu            16.04     b6f507652425   2 years ago    135MB
training/webapp   latest    6fae60ef3446   8 years ago    349MB

C:\Users\L14>docker tag  5b9f3a8a4ebd mytestimage:1604

C:\Users\L14>docker images
REPOSITORY        TAG       IMAGE ID       CREATED        SIZE
kernelubuntu      2204      480bd2a1a10b   5 hours ago    499MB
kernelubuntu      1804      318e5e80dbcf   5 hours ago    433MB
kernelubuntu      1604      2d3c656084a9   5 hours ago    466MB
myimage           16.04     5b9f3a8a4ebd   5 hours ago    117MB
mytestimage       1604      5b9f3a8a4ebd   5 hours ago    117MB
ubuntu            latest    e4c58958181a   2 weeks ago    77.8MB
ubuntu            18.04     f9a80a55f492   4 months ago   63.2MB
ubuntu            16.04     b6f507652425   2 years ago    135MB
training/webapp   latest    6fae60ef3446   8 years ago    349MB
```

可以看出还是去操作的id

测试容器

完成在容器中的工作就把它删掉。如果是这样，通过使用 `--rm` 标签在关闭后自动删掉容器

```
sudo docker run -it --rm debian:latest
```

交互式运行容器

```
C:\Users\L14>docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       latest    e4c58958181a   13 days ago    77.8MB
ubuntu       18.04     f9a80a55f492   4 months ago   63.2MB
ubuntu       16.04     b6f507652425   2 years ago    135MB
```

查看现在的机器里面有哪些image，想运行容器必须要有 image才可以，注意也可以使用image id来运行容器

```
C:\Users\L14>docker run -it ubuntu:16.04 /bin/bash
root@527ca4fe1a0e:/#
```

参数说明：

- **-i**: 交互式操作。
- **-t**: 终端。
- **ubuntu**: ubuntu 镜像。
- **/bin/bash**：放在镜像名后的是命令，这里我们希望有个交互式 Shell，因此用的是 /bin/bash

从容器种脱离

使用 `CTRL+P` 然后 `CTRL+Q` 就可以从运行中的容器脱离（不需要关闭）。

现在，你就回到了你原来的主机的终端窗口。请注意，容器还在后台运行中，我们并没有关掉它。

后台运行

在大部分的场景下，我们希望 docker 的服务是在后台运行的，我们可以过 -d 指定容器的运行模式。

```
docker run -itd --name ubuntu-test ubuntu /bin/bash
```

停止容器

因为有事需要停止容器有几种方法

再交互式的界面里面输入 `exit` 即可

停止容器的命令如下：

```
docker stop <容器 ID>
```

```
C:\Users\L14>docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED             STATUS          PORTS
NAMES
527ca4fe1a0e   ubuntu:16.04   "/bin/bash"   About an hour ago   Up 24 minutes
kind_liskov

C:\Users\L14>docker stop 527ca4fe1a0e
527ca4fe1a0e
```

再次启动容器

停止的容器可以通过 docker restart 重启：

```
$ docker restart <容器 ID>
```

```
C:\Users\L14>docker restart  527ca4fe1a0e
527ca4fe1a0e

C:\Users\L14>docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED       STATUS         PORTS
NAMES
527ca4fe1a0e   ubuntu:16.04   "/bin/bash"   2 hours ago   Up 6 seconds
kind_liskov
```

之后想再次运行以前的容器怎么办？输入 `docker ps -a` 命令，它可以查询有哪些关闭了的容器。

```
C:\Users\L14>docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS
PORTS     NAMES
527ca4fe1a0e   ubuntu:16.04   "/bin/bash"   51 minutes ago   Exited (0) 50 minutes ago
kind_liskov
a2249fed3267   ubuntu:18.04   "/bin/bash"   6 hours ago      Exited (0) 6 hours ago
strange_rhodes
cf4944e1614d   ubuntu:16.04   "/bin/bash"   6 hours ago      Exited (1) 6 hours ago
fervent_pascal
217380518db6   ubuntu         "/bin/bash"   7 hours ago      Exited (137) 6 hours ago
elastic_bohr
```

要启动第一个容器，可以使用以下命令：

```
docker start 527ca4fe1a0e
```

注意这里可以同时启动多个容器 就是在id部分用空格隔开即可。

其中，`527ca4fe1a0e` 是容器的 ID。如果你想要进入该容器的 shell，可以使用以下命令：

```
docker exec -it 527ca4fe1a0e /bin/bash
```

这将会在该容器中打开一个交互式的 shell。进入交互式的前提是先执行上面的`start`的命令来启动容器，然后用第二个命令来进入容器里面

```
C:\Users\L14>docker start 527ca4fe1a0e
527ca4fe1a0e

C:\Users\L14>docker exec -it 527ca4fe1a0e /bin/bash
root@527ca4fe1a0e:/#
```

进入容器

在使用 -d 参数时，容器启动后会进入后台。此时想要进入容器，可以通过以下指令进入：

- **docker attach**
- **docker exec**：推荐 docker exec 命令，因为此命令会退出容器终端，但不会导致容器的停止。

attach 命令

下面演示了使用 docker attach 命令。

```
C:\Users\L14>docker attach 527ca4fe1a0e
root@527ca4fe1a0e:/#
root@527ca4fe1a0e:/# exit
exit

C:\Users\L14>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

注意： 如果从这个容器退出，会导致容器的停止。

exec 命令

下面演示了使用 docker exec 命令。

```
C:\Users\L14>docker restart  527ca4fe1a0e
527ca4fe1a0e

C:\Users\L14>docker exec -it 527ca4fe1a0e /bin/bash
root@527ca4fe1a0e:/# exit
exit

C:\Users\L14>docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED       STATUS          PORTS
NAMES
527ca4fe1a0e   ubuntu:16.04   "/bin/bash"   2 hours ago   Up 14 seconds
kind_liskov
```

导出容器

如果要导出本地某个容器，可以使用 **docker export** 命令。

```
C:\Users\L14>docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED       STATUS          PORTS
NAMES
527ca4fe1a0e   ubuntu:16.04   "/bin/bash"   2 hours ago   Up 14 seconds
kind_liskov

C:\Users\L14>docker export 527ca4fe1a0e >myimage.tar
```

可以不用关闭容器的导出。导出到了当前目录的`myimage.tar`

导入容器

可以使用 docker import 从容器快照文件中再导入为镜像

```
C:\Users\L14>docker import myimage.tar test:123
sha256:1fc6f6e37aa10d1787ac185c2015e15dfa73169285793921b830e740247f7d90

C:\Users\L14>docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
test         123       1fc6f6e37aa1   6 seconds ago   117MB
myimage      16.04     5b9f3a8a4ebd   4 minutes ago   117MB
ubuntu       latest    e4c58958181a   2 weeks ago     77.8MB
ubuntu       18.04     f9a80a55f492   4 months ago    63.2MB
ubuntu       16.04     b6f507652425   2 years ago     135MB
```

容器删除

用docker rm 命令来实现下面看示例

```
C:\Users\L14>docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
test         123       1fc6f6e37aa1   6 seconds ago   117MB
myimage      16.04     5b9f3a8a4ebd   4 minutes ago   117MB
ubuntu       latest    e4c58958181a   2 weeks ago     77.8MB
ubuntu       18.04     f9a80a55f492   4 months ago    63.2MB
ubuntu       16.04     b6f507652425   2 years ago     135MB

C:\Users\L14>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\L14>docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED       STATUS
PORTS     NAMES
a2249fed3267   ubuntu:18.04   "/bin/bash"   9 hours ago   Exited (0) 8 hours ago
strange_rhodes
cf4944e1614d   ubuntu:16.04   "/bin/bash"   9 hours ago   Exited (1) 8 hours ago
fervent_pascal
217380518db6   ubuntu         "/bin/bash"   9 hours ago   Exited (137) 8 hours ago
elastic_bohr

C:\Users\L14>docker run -it test:123 /bin/bash
root@a1d3a147af09:/# exit
exit

C:\Users\L14>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\L14>docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS
PORTS     NAMES
a1d3a147af09   test:123       "/bin/bash"   23 seconds ago   Exited (0) 10 seconds ago
vigilant_bartik
a2249fed3267   ubuntu:18.04   "/bin/bash"   9 hours ago      Exited (0) 8 hours ago
strange_rhodes
cf4944e1614d   ubuntu:16.04   "/bin/bash"   9 hours ago      Exited (1) 8 hours ago
fervent_pascal
217380518db6   ubuntu         "/bin/bash"   9 hours ago      Exited (137) 8 hours ago
elastic_bohr

C:\Users\L14>docker rm -f a1d3a147af09
a1d3a147af09

C:\Users\L14>docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED       STATUS
PORTS     NAMES
a2249fed3267   ubuntu:18.04   "/bin/bash"   9 hours ago   Exited (0) 8 hours ago
strange_rhodes
cf4944e1614d   ubuntu:16.04   "/bin/bash"   9 hours ago   Exited (1) 8 hours ago
fervent_pascal
```

```
217380518db6   ubuntu         "/bin/bash"   9 hours ago   Exited (137) 8 hours ago
elastic_bohr
```

暂停一个运行中的容器

`sudo docker pause 10615254bb45`

把暂停的容器恢复过来

`sudo docker unpause 10615254bb45`

网络应用端口示例操作

```
C:\Users\L14>docker pull training/webapp
Using default tag: latest
latest: Pulling from training/webapp
[DEPRECATION NOTICE] Docker Image Format v1, and Docker Image manifest version 2,
schema 1 support will be removed in an upcoming release. Suggest the author of
docker.io/training/webapp:latest to upgrade the image to the OCI Format, or Docker
Image manifest v2, schema 2. More information at
https://docs.docker.com/go/deprecated-image-specs/
e190868d63f8: Pull complete
909cd34c6fd7: Pull complete
0b9bfabab7c1: Pull complete
a3ed95caeb02: Pull complete
10bbbc0fc0ff: Pull complete
fca59b508e9f: Pull complete
e7ae2541b15b: Pull complete
9dd97ef58ce9: Pull complete
a4c1b0cb7af7: Pull complete
Digest: sha256:06e9c1983bd6d5db5fba376ccd63bfa529e8d02f23d5079b8f74a616308fb11d
Status: Downloaded newer image for training/webapp:latest
docker.io/training/webapp:latest

What's Next?
  1. Sign in to your Docker account → docker login
  2. View a summary of image vulnerabilities and recommendations → docker scout
quickvi
ew training/webapp

C:\Users\L14>docker run -d -P training/webapp python app.py
9ee67b858b78ed153df2a193164e8a3bf36f7b57435b1fb1a6e6d5ea63c38219
```

参数说明:

- -d 让容器在后台运行。

- -P 将容器内部使用的网络端口随机映射到我们使用的主机上。

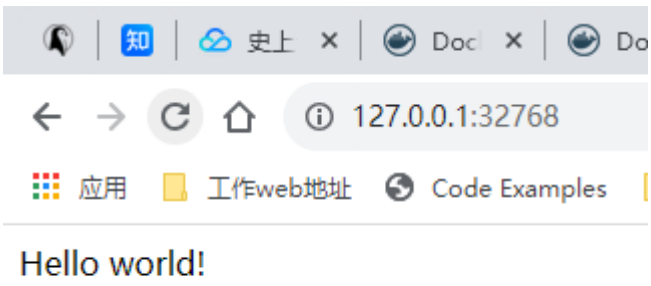
查看运行的容器

```
C:\Users\L14>docker ps
CONTAINER ID   IMAGE             COMMAND           CREATED              STATUS
PORTS                     NAMES
4a602b00715e   training/webapp   "python app.py"   About a minute ago   Up About a
minute   0.0.0.0:32768->5000/tcp   brave_brahmagupta## 0x04 常规应用
```

可以看到和以往不同的是有了端口

`0.0.0.0:32768->5000/tcp`

本机访问效果



实际上是把docker内部的5000端口转发到了宿主的32768上面。这个32768是随机的端口

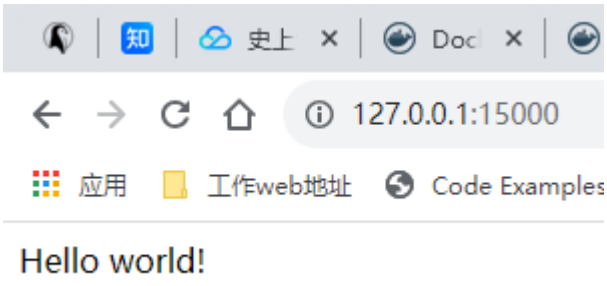
还可以通过`-p`参数来实现指定内外端口的映射关系

```
C:\Users\L14>docker run -d -p 15000:5000 training/webapp python app.py
6baf849a3f2dc2be530ba6b96a1361ac14e5beeda1fffded9a62dab1d6f4ae38

C:\Users\L14>docker ps
CONTAINER ID   IMAGE             COMMAND           CREATED         STATUS
PORTS                     NAMES
6baf849a3f2d   training/webapp   "python app.py"   8 seconds ago   Up 3 seconds
0.0.0.0:15000->5000/tcp   awesome_fermi
4a602b00715e   training/webapp   "python app.py"   6 minutes ago   Up 6 minutes
0.0.0.0:32768->5000/tcp   brave_brahmagupta
```

参数说明

- -p 15000：5000 是把docker的5000端口映射到宿主的 15000上面
- -p 宿主：docker



还可以映射 udp的端口出来

```
C:\Users\L14>docker run -d -p 127.0.0.1:15000:5000/udp training/webapp python app.py
4b0f28d9baced5c153f39469d4ffe3a9e48bf11fff23c776f0420e17b4066eb4

C:\Users\L14>docker ps
CONTAINER ID   IMAGE             COMMAND           CREATED          STATUS
PORTS                                NAMES
4b0f28d9bace   training/webapp   "python app.py"   14 seconds ago   Up 9 seconds
5000/tcp, 127.0.0.1:15000->5000/udp   fervent_wright
6baf849a3f2d   training/webapp   "python app.py"   47 minutes ago   Up 47 minutes
0.0.0.0:15000->5000/tcp               awesome_fermi
4a602b00715e   training/webapp   "python app.py"   54 minutes ago   Up 54 minutes
0.0.0.0:32768->5000/tcp               brave_brahmagupta

C:\Users\L14>docker port 4b0f28d9bace
5000/udp -> 127.0.0.1:15000
```

docker 容器端口查询

上面提到了会有随机的映射的方式那么查询端口当然可是用docker ps来查询。还可以使用下面的port命令

```
C:\Users\L14>docker ps
CONTAINER ID   IMAGE             COMMAND           CREATED          STATUS
PORTS                     NAMES
6baf849a3f2d   training/webapp   "python app.py"   7 minutes ago    Up 7 minutes
0.0.0.0:15000->5000/tcp   awesome_fermi
4a602b00715e   training/webapp   "python app.py"   14 minutes ago   Up 14 minutes
0.0.0.0:32768->5000/tcp   brave_brahmagupta

C:\Users\L14>docker port 4a602b00715e
5000/tcp -> 0.0.0.0:32768

C:\Users\L14>docker port  brave_brahmagupta
5000/tcp -> 0.0.0.0:32768
```

docker 大多数的情况下各种命令都可以对name和id进行操作，id是唯一的，name不重名有也能直接拿来用。

docker 查看容器内日志

使用logs 命令来查询

```
C:\Users\L14>docker logs 4a602b00715e
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
172.17.0.1 - - [19/Oct/2023 13:38:39] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [19/Oct/2023 13:38:40] "GET /favicon.ico HTTP/1.1" 404 -

C:\Users\L14>docker logs -f 4a602b00715e
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
172.17.0.1 - - [19/Oct/2023 13:38:39] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [19/Oct/2023 13:38:40] "GET /favicon.ico HTTP/1.1" 404 -
^C
C:\Users\L14>
C:\Users\L14>
C:\Users\L14>
```

```
C:\Users\L14>docker logs --help

Usage:  docker logs [OPTIONS] CONTAINER

Fetch the logs of a container

Aliases:
  docker container logs, docker logs

Options:
      --details        Show extra details provided to logs
  -f, --follow         Follow log output
      --since string   Show logs since timestamp (e.g.
                       "2013-01-02T13:23:37Z") or relative (e.g. "42m"
                       for 42 minutes)
  -n, --tail string    Number of lines to show from the end of the logs
                       (default "all")
  -t, --timestamps     Show timestamps
      --until string   Show logs before a timestamp (e.g.
                       "2013-01-02T13:23:37Z") or relative (e.g. "42m"
                       for 42 minutes)
```

参数解释

- docker logs -f 其中 `-f` 参数表示实时跟踪日志输出，即在容器中有新的日志输出时，会自动将其显示在终端中。这个命令非常有用，可以帮助开发人员快速定位容器中的问题。

查看容器内的top进程

`docker top` 命令用于查看指定容器内部运行的进程信息，类似于 Linux 系统中的 `top` 命令。该命令的语法如下：

```
docker top CONTAINER [ps OPTIONS]
```

其中，`CONTAINER` 参数指定要查看进程信息的容器名称或 ID，`ps OPTIONS` 参数用于指定 `ps` 命令的选项，用于过滤和格式化进程信息。

例如，要查看名为 `mycontainer` 的容器内部运行的所有进程信息，可以使用以下命令：

```
C:\Users\L14>docker ps
CONTAINER ID   IMAGE             COMMAND           CREATED          STATUS
PORTS                      NAMES
6baf849a3f2d   training/webapp   "python app.py"   16 minutes ago   Up 16 minutes
0.0.0.0:15000->5000/tcp   awesome_fermi
4a602b00715e   training/webapp   "python app.py"   23 minutes ago   Up 23 minutes
0.0.0.0:32768->5000/tcp   brave_brahmagupta

C:\Users\L14>docker top 4a602b00715e
```

该命令会输出类似以下的进程信息：

```
UID                 PID                 PPID                C                   STIME
TTY                 TIME                CMD
root                1750                1729                0                   13:35
?                   00:00:00            python app.py
```

其中，各列的含义如下：

- `UID`：进程的用户 ID。
- `PID`：进程的进程 ID。
- `PPID`：进程的父进程 ID。
- `C`：进程的 CPU 占用率。
- `STIME`：进程的启动时间。
- `TTY`：进程所在的终端。
- `TIME`：进程的 CPU 时间。

- `CMD`： 进程的命令行。

查看 docker 的底层信息

`docker inspect` 是一个用于检查 Docker 对象（如容器、镜像、网络等）详细信息的命令。它可以返回一个 JSON 格式的对象，其中包含了有关 Docker 对象的各种元数据，如配置、网络设置、挂载点、环境变量等等。

使用 `docker inspect` 命令可以获取 Docker 对象的详细信息，例如容器的 IP 地址、端口映射、挂载的卷、环境变量等等。可以通过 `docker inspect` 命令来查看容器的详细信息

```
C:\Users\L14>docker inspect  4a602b00715e
[
    {
        "Id": "4a602b00715e72868613da35d4e6285dbd86bf50cf6d9dd8af722ef9dad6860f",
        "Created": "2023-10-19T13:35:25.895436759Z",
        "Path": "python",
        "Args": [
            "app.py"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 1750,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2023-10-19T13:35:30.04231991Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image":
"sha256:6fae60ef344644649a39240b94d73b8ba9c67f898ede85cf8e947a887b3e6557",
        "ResolvConfPath":
"/var/lib/docker/containers/4a602b00715e72868613da35d4e6285dbd86bf50cf6d9dd8af722ef9da
d6860f/resolv.conf",
        "HostnamePath":
"/var/lib/docker/containers/4a602b00715e72868613da35d4e6285dbd86bf50cf6d9dd8af722ef9da
d6860f/hostname",
        "HostsPath":
"/var/lib/docker/containers/4a602b00715e72868613da35d4e6285dbd86bf50cf6d9dd8af722ef9da
d6860f/hosts",
        "LogPath":
"/var/lib/docker/containers/4a602b00715e72868613da35d4e6285dbd86bf50cf6d9dd8af722ef9da
d6860f/4a602b00715e72868613da35d4e6285dbd86bf50cf6d9dd8af722ef9dad6860f-json.log",
        "Name": "/brave_brahmagupta",
        "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,
        "VolumeDriver": "",
        "VolumesFrom": null,
        "ConsoleSize": [
            62,
            235
        ],
        "CapAdd": null,
        "CapDrop": null,
        "CgroupnsMode": "host",
        "Dns": [],
        "DnsOptions": [],
        "DnsSearch": [],
        "ExtraHosts": null,
        "GroupAdd": null,
        "IpcMode": "private",
        "Cgroup": "",
        "Links": null,
        "OomScoreAdj": 0,
        "PidMode": "",
        "Privileged": false,
        "PublishAllPorts": true,
        "ReadonlyRootfs": false,
        "SecurityOpt": null,
        "UTSMode": "",
        "UsernsMode": "",
        "ShmSize": 67108864,
        "Runtime": "runc",
        "Isolation": "",
        "CpuShares": 0,
        "Memory": 0,
        "NanoCpus": 0,
        "CgroupParent": "",
        "BlkioWeight": 0,
        "BlkioWeightDevice": [],
        "BlkioDeviceReadBps": [],
        "BlkioDeviceWriteBps": [],
        "BlkioDeviceReadIOps": [],
        "BlkioDeviceWriteIOps": [],
        "CpuPeriod": 0,
        "CpuQuota": 0,
        "CpuRealtimePeriod": 0,
        "CpuRealtimeRuntime": 0,
        "CpusetCpus": "",
```

```
            "CpusetMems": "",
            "Devices": [],
            "DeviceCgroupRules": null,
            "DeviceRequests": null,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": null,
            "OomKillDisable": false,
            "PidsLimit": null,
            "Ulimits": null,
            "CpuCount": 0,
            "CpuPercent": 0,
            "IOMaximumIOps": 0,
            "IOMaximumBandwidth": 0,
            "MaskedPaths": [
                "/proc/asound",
                "/proc/acpi",
                "/proc/kcore",
                "/proc/keys",
                "/proc/latency_stats",
                "/proc/timer_list",
                "/proc/timer_stats",
                "/proc/sched_debug",
                "/proc/scsi",
                "/sys/firmware"
            ],
            "ReadonlyPaths": [
                "/proc/bus",
                "/proc/fs",
                "/proc/irq",
                "/proc/sys",
                "/proc/sysrq-trigger"
            ]
        },
        "GraphDriver": {
            "Data": {
                "LowerDir":
"/var/lib/docker/overlay2/e0f28dbcf8d7f3a8b90cd13eb5274fc664a4bd677809bf038ee087ce8a66
e362-
init/diff:/var/lib/docker/overlay2/dc592009da7b75b1e2960c33b8b397f61f88b4283539dc95d34
b02eca3f06f46/diff:/var/lib/docker/overlay2/471dbfca4786ffa3474112c5046339ee6d6b0daa5c
03f31f08192866c64bd9ba/diff:/var/lib/docker/overlay2/2a63530d219e516d89cec752d99a35b7e
a6571e90dca3c100f9487e579d68e83/diff:/var/lib/docker/overlay2/60366501d2545585cf6709f0
dff9b4313ba8ab4a9aa6d2d6697644b8b05e6168/diff:/var/lib/docker/overlay2/e3627acc41a8e06
550806a74485689e2a5b1f77f1441da8177c71f0ad92b057f/diff:/var/lib/docker/overlay2/819b04
d4c147eb9cb89bca4a889e910a5ca19f7aaaeb982432eb81dc2bbca9d5/diff:/var/lib/docker/overla
y2/3cf5460a13f5c6f236004c0d05673fef30ed61048304ed513f76c3ef23677815/diff:/var/lib/dock
er/overlay2/cffeb35a1581b4de66de9400b231f29c0deedd8d67555c7e9c78c151d732190c/diff:/var
/lib/docker/overlay2/42a92222fea1318e0787d5c878f33a49ba4b18d92fcd9c959346248435c5a843/
diff:/var/lib/docker/overlay2/504d4c3faebd9eee1ed2b440a5a51f21002e1141a234b4d58d82605f
b727eb0f/diff:/var/lib/docker/overlay2/95942432c3ecdea1490d67c9c39908ace44469b829a497b
```

```
1e9f8291b8d32ec16/diff:/var/lib/docker/overlay2/2b508319da6b7b0c32fa802abb1a0471025b69
5960331a5d08bc7498cd01d5c5/diff:/var/lib/docker/overlay2/455be259f8eaaf8135c72ac872419
d9d8d991d0d2713dfaa0487ca8d61c066c2/diff",
                "MergedDir":
"/var/lib/docker/overlay2/e0f28dbcf8d7f3a8b90cd13eb5274fc664a4bd677809bf038ee087ce8a66
e362/merged",
                "UpperDir":
"/var/lib/docker/overlay2/e0f28dbcf8d7f3a8b90cd13eb5274fc664a4bd677809bf038ee087ce8a66
e362/diff",
                "WorkDir":
"/var/lib/docker/overlay2/e0f28dbcf8d7f3a8b90cd13eb5274fc664a4bd677809bf038ee087ce8a66
e362/work"
            },
            "Name": "overlay2"
        },
        "Mounts": [],
        "Config": {
            "Hostname": "4a602b00715e",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "5000/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "python",
                "app.py"
            ],
            "Image": "training/webapp",
            "Volumes": null,
            "WorkingDir": "/opt/webapp",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"668231ae1cb4fdbfce9a885422d8697cd92932f347b2463429061a4ca5aa5c7e",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
```

```
                "5000/tcp": [
                    {
                        "HostIp": "0.0.0.0",
                        "HostPort": "32768"
                    }
                ]
            },
            "SandboxKey": "/var/run/docker/netns/668231ae1cb4",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID":
"8825fcaf072c9d353dd478b11da96322a70f21f471ab894295d6679926b58d8c",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID":
"9d70d5463c788831eef3610a6ff8ea09574d081028dc48d9e86860b910bd6f91",
                    "EndpointID":
"8825fcaf072c9d353dd478b11da96322a70f21f471ab894295d6679926b58d8c",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
        }
    }
]
```


挂载外面的文件夹到docker内使用

使用 Docker 的 `-v` 参数来挂载一个本地文件夹到容器中。

```
docker run -it -v /home/dock/Downloads:/usr/Downloads ubuntu64 /bin/bash
```

上面是linux下的操作

windows下docker desktop挂载本地路径

```
C:\Users\L14>docker run -it -v /d/temp:/tmp e4c58958181a /bin/bash
root@62643b8faeae:/# ip a
root@62643b8faeae:/# cd tmp/
root@62643b8faeae:/tmp# ls
root@62643b8faeae:/tmp# ls
''$'\346\226\260\345\273\272\346\226\207\344\273\266\345\244\271'
root@62643b8faeae:/tmp#
```

在windows下目录一般是这样的`D:\temp`在使用时，就把某个盘当做根目录下的子目录路径 分隔符使用`/`，所以`D:\temp`就变成了`/d/temp`这样来挂载使用

删掉所有未运行的容器、所有镜像、构建的缓存、所有网络

`sudo docker system prune -a`

0x04 docker 应用

构建自己的image

ubuntu 更换为ustc的源，在交互式的docker容器里面

```
sed -i 's@//.*archive.ubuntu.com@//mirrors.ustc.edu.cn@g' /etc/apt/sources.list
apt update
apt upgrade
```

使用`docker commit`命令将容器保存为一个新的镜像。具体步骤如下：

1. 首先使用`docker ps -a`命令查看你要保存的容器的 ID。
2. 然后使用`docker commit`命令将容器保存为一个新的镜像，命令格式如下：

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

   其中，`OPTIONS`为可选参数，`CONTAINER`为容器的 ID，`REPOSITORY`为新镜像的名称，`TAG`为新镜像的标签。

例如，假设你要将 ID 为 `527ca4fe1a0e` 的容器保存为一个名为 `myimage`，标签为 `v1.0` 的镜像，可以使用以下命令：

```
C:\Users\L14>docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED       STATUS
PORTS     NAMES
a2249fed3267   ubuntu:18.04   "/bin/bash"   9 hours ago   Exited (0) 8 hours ago
strange_rhodes
cf4944e1614d   ubuntu:16.04   "/bin/bash"   9 hours ago   Exited (1) 8 hours ago
fervent_pascal
217380518db6   ubuntu         "/bin/bash"   9 hours ago   Exited (137) 8 hours ago
elastic_bohr

C:\Users\L14>docker commit  cf4944e1614d kernelubuntu:1604
sha256:2d3c656084a96a3de1374140b0022f61732fd6352e03f057ff51223cc739c9b2

C:\Users\L14>
C:\Users\L14>
C:\Users\L14>docker images
REPOSITORY     TAG       IMAGE ID       CREATED              SIZE
kernelubuntu   1604      2d3c656084a9   About a minute ago   466MB
test           123       1fc6f6e37aa1   18 minutes ago       117MB
myimage        16.04     5b9f3a8a4ebd   22 minutes ago       117MB
ubuntu         latest    e4c58958181a   2 weeks ago          77.8MB
ubuntu         18.04     f9a80a55f492   4 months ago         63.2MB
ubuntu         16.04     b6f507652425   2 years ago          135MB

C:\Users\L14>docker commit  a2249fed3267 kernelubuntu:1804
sha256:318e5e80dbcfa6a6ec8a97c9e3c2caaabb51fda13cd985c9ee54a456203f5fb7

C:\Users\L14>docker commit  217380518db6 kernelubuntu:2204
sha256:480bd2a1a10b7a8665c243839e73aa543f2964723969117a4c5d6ead657834b6

C:\Users\L14>docker images
REPOSITORY     TAG       IMAGE ID       CREATED              SIZE
kernelubuntu   2204      480bd2a1a10b   18 seconds ago       499MB
kernelubuntu   1804      318e5e80dbcf   About a minute ago   433MB
kernelubuntu   1604      2d3c656084a9   4 minutes ago        466MB
test           123       1fc6f6e37aa1   22 minutes ago       117MB
myimage        16.04     5b9f3a8a4ebd   26 minutes ago       117MB
ubuntu         latest    e4c58958181a   2 weeks ago          77.8MB
ubuntu         18.04     f9a80a55f492   4 months ago         63.2MB
ubuntu         16.04     b6f507652425   2 years ago          135MB
```

3. 等待镜像保存完成后，可以使用 `docker images` 命令查看新的镜像是否已经保存成功。

内核编译

wordpress

vpn

0x05 精简版

```
docker pull ubuntu
```

docker ubuntu

```
sudo sed -i 's@//.*archive.ubuntu.com@//mirrors.ustc.edu.cn@g' /etc/apt/sources.list
sudo sed -i 's/http:/https:/g' /etc/apt/sources.list
sudo apt update



sudo apt-get install ca-certificates curl gnupg lsb-release

curl -fsSL https://get.docker.com -o get-docker.sh
sudo DOWNLOAD_URL=https://mirrors.ustc.edu.cn/docker-ce sh get-docker.sh
```

本文章的参考链接

Docker 教程 | 菜鸟教程