> 本文由 简悦 SimpRead 转码， 原文地址 www.imooc.com

本篇文章介绍 Docker 的操作参数，包括针对容器（container）、镜像（image）和镜像仓库中心（registry）的操作命令。

1. 容器操作

Docker 和容器相关的常用的操作命令如下：

run

docker run 用来通过镜像启动一个容器。这个可以算是操作 docker 容器的核心命令了，参数及其丰富，多达 91 个参数，使用手册如下：

```
[root@emr-header-1 ~]

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

Run a command in a new container
```

比如我们启动一个 centos 的镜像

```
[root@xxx ~]
[root@b16716790a3f /]
```

其中 centos:latest 就是 Usage 中的 IMAGE，COMMAND 就是 /bin/bash，然后 -ti 分配一个终端用于交互式输入。

我们下面主要介绍几个重要的参数。

--interactive 等同于 ***-i***，接受 stdin 的输入；

--tty 等同于 ***-t***，分配一个 tty，一般和 i 一起使用；

--name 给容器设置一个名字；

--add-host 给容器设置 hosts 文件，格式 host:ip；

--env 环境变量设置；

--expose 暴露端口；

--hostname 设置容器的主机名；

--link 容器网络相关，和其他的 container 连接；

--cpu-quota 设置 CPU 限制；

--memory 设置容器可以使用的内存限制。

attach

docker attach 让我们可以进入到一个运行着的容器的内部，这个命令的原理是给一个正在运行的容器分配一个 stdin、stdout 和 stderr。

```
[root@xxx ~]

Usage:  docker attach [OPTIONS] CONTAINER

Attach local standard input, output, and error streams to a running container

Options:
      --detach-keys string   Override the key sequence for detaching a container
      --no-stdin             Do not attach STDIN
      --sig-proxy            Proxy all received signals to the process (default
true)
```

需要注意的是，如果 docker attach 之后要退出的话，不能使用 exit，使用 exit 原容器也会退出。我们可以使用 Ctrl + C 的方式退出。

exec

docker exec 命令也可以达到 attach 的目的。exec 命令的用处是在一个运行着的容器里面执行一个命令。关于这个命令的原理其实很简单，在 Linux 内核层面，相当于 fork 了一个进程，然后这个进程设置和容器相同的 NameSpace。如果我们 OPTIONS 指定 -ti ，那么我们就可以进入到一个运行着的容器里面执行命令了。因为这个是一个 fork 出来的进程，所以可以 exit。

```
[root@emr-header-1 ~]

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

Run a command in a running container

Options:
  -d, --detach               Detached mode: run command in the background
      --detach-keys string   Override the key sequence for detaching a container
  -e, --env list             Set environment variables
  -i, --interactive          Keep STDIN open even if not attached
      --privileged           Give extended privileges to the command
  -t, --tty                  Allocate a pseudo-TTY
  -u, --user string          Username or UID (format: <name|uid>[:<group|gid>])
  -w, --workdir string       Working directory inside the container
```

ps

docker ps 可以用来列出所有在运行的容器的信息，同时支持一些类似 filter 的参数。

```
[root@xxx ~]

Usage:  docker ps [OPTIONS]
```

```
List containers

Options:
  -a, --all             Show all containers (default shows just running)
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print containers using a Go template
  -n, --last int        Show n last created containers (includes all states)
(default -1)
  -l, --latest          Show the latest created container (includes all states)
      --no-trunc        Don't truncate output
  -q, --quiet           Only display numeric IDs
  -s, --size            Display total file sizes
```

kill

docker kill 用来 kill 一个或者一组 container。

```
[root@xxx ~]

Usage:  docker kill [OPTIONS] CONTAINER [CONTAINER...]

Kill one or more running containers

Options:
  -s, --signal string   Signal to send to the container (default "KILL")
```

logs

docker logs 用来获取 docker 的 log。

```
[root@xxx ~]

Usage:  docker logs [OPTIONS] CONTAINER

Fetch the logs of a container

Options:
      --details        Show extra details provided to logs
  -f, --follow         Follow log output
      --since string   Show logs since timestamp (e.g. 2013-01-02T13:23:37) or
relative (e.g. 42m for 42 minutes)
      --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. 2013-01-02T13:23:37) or
relative (e.g. 42m for 42 minutes)
```

top

docker top 这个命令有时候比较有用，我们想看一下运行这个容器在宿主机上面是那个进程就可以使用这个命令，毕竟 container 的本质就是一个进程，这个我们后面会细说。

```
[root@xxx ~]

Usage:  docker top CONTAINER [ps OPTIONS]

Display the running processes of a container
```

2. 镜像操作

和镜像相关的常用的操作命令如下：

- images：列出本地所有的镜像；
- build：通过 Dockerfile build 出镜像；
- commit：将容器中的所有改动生成新的镜像；
- history：查看镜像的历史；
- save：将镜像保存成 tar 包；
- import：通过 tar 包导入新的镜像；
- load：通过 tar 包或者标志输入导入镜像；
- rmi：删除本地镜像；
- tag：给镜像打 tag。

images

docker images 会显示本地所有的非隐藏镜像，默认会将中间依赖镜像进行隐藏。

```
[root@xxx ~]
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
busybox             1-musl              ff04c2bddacb        3 days ago
1.44MB
busybox             1-glibc             ad06ec8ab37b        3 days ago
5.2MB
busybox             1-uclibc            b534869c81f0        3 days ago
1.22MB
busybox             latest              b534869c81f0        3 days ago
1.22MB
busybox             1.24-glibc          54df49495ae4        3 years ago
4.18MB
busybox             1-ubuntu            d34ea343a882        3 years ago
4.35MB
busybox             1.21-ubuntu         d34ea343a882        3 years ago
4.35MB
busybox             1.21.0-ubuntu       d34ea343a882        3 years ago
4.35MB
busybox             1.23                a84c36ecc374        4 years ago
1.1MB
busybox             1.23.2              a84c36ecc374        4 years ago
1.1MB
```

这是 docker images 的默认显示，但是有时候我们需要显示更多的信息比如 Digest，或者过滤掉一些镜像，那么我们可以通过添加参数来实现。我们看一下 docker images 的完整的功能。

```
[root@xxx ~]

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

List images

Options:
  -a, --all             Show all images (default hides intermediate images)
      --digests         Show digests
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print images using a Go template
      --no-trunc        Don't truncate output
  -q, --quiet           Only show numeric IDs
```

- ***all*** 参数显示所有镜像，包括中间依赖镜像；
- ***digest*** 显示 digest；
- ***filter*** 对镜像进行过滤；
- ***no-trunc*** 默认会对某些输出列进行截断展示，该参数将全量展示。
- ***quiet*** 只展示镜像的数字 ID。
- ***format*** 参数使用一种 Go 模板的形式输出，简单来说就是指定输出的列。这么描述不太直观，我们可以看一个简单的例子如下，其中的 ID 和 Repository 就是指定的输出的列。

```
[root@xxx ~]
ff04c2bddacb: busybox
ad06ec8ab37b: busybox
b534869c81f0: busybox
b534869c81f0: busybox
```

除了 ID 和 Repository，Docker 支持的全部的列如下：

Placeholder	Description
`.ID`	镜像 ID
`.Repository`	镜像 repo，其实就是镜像名称
`.Tag`	镜像 tag
`.Digest`	镜像 digest
`.CreatedSince`	镜像创建之后多长时间
`.CreatedAt`	镜像的创建时间
`.Size`	镜像的磁盘大小

build

在 Docker 中我们可以通过一个 Dockerfile，使用 docker build 构建出镜像。这一块我们后面将会有专门的小节来展开，这里暂时先略过。

commit

上面说到我们可以通过一个 Dockerfile 构建出镜像，但是有时候我们在使用过程中对容器（container）做了一些改动，比如安装了一些依赖包。我们想把这些改动保存下来形成新的镜像，同时不想再去编写 Dockerfile，或者之前的 Dockerfile 我们没有。那么这时候我们就可以通过 docker commit 将一个 container 的环境持久成镜像。我们首先看一下 docker commit 的使用规范说明。

```
[root@xxx ~]

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

Create a new image from a container's changes

Options:
  -a, --author string    Author (e.g., "John Hannibal Smith <hannibal@a-
team.com>")
  -c, --change list      Apply Dockerfile instruction to the created image
  -m, --message string   Commit message
  -p, --pause            Pause container during commit (default true)
```

如果要对现在的一个 container 进行 commit 操作，我们首先可以通过 docker ps 找到我们要执行 commit 操作的 container 的 id。

```
[root@xxx ~]
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
d7bb841d9811        legendtkl:v2.1   "/bin/bash"         About a minute ago   Up
About a minute   22/tcp              vigorous_chaplygin
```

然后执行 commit 。

然后再执行 docker images 就可以看到我们新 commit 出来的镜像了。

history

使用 docker history 可以查看镜像的历史，举个例子。后面的命令我们在 Dockerfile 那一节在详细说。

```
[root@emr-header-1 ~]
IMAGE               CREATED             CREATED BY
       SIZE                COMMENT
a84c36ecc374        4 years ago         /bin/sh -c
<missing>           4 years ago         /bin/sh -c
[root@emr-header-1 ~]
IMAGE               CREATED             CREATED BY
       SIZE                COMMENT
243b193cdf70        3 hours ago
      0B
```

```
e104690ec93c        2 weeks ago         /bin/sh -c
<missing>           2 weeks ago         /bin/sh -c
<missing>           2 weeks ago         /bin/sh -c
<missing>           2 weeks ago         /bin/sh -c
<missing>           2 weeks ago         /bin/sh -c
<missing>           5 weeks ago         /bin/sh -c mkdir -p /app
       0B
<missing>           4 years ago         /bin/sh -c
<missing>           4 years ago         /bin/sh -c
<missing>           4 years ago         /bin/sh -c sh /tmp/install/run.sh
      260MB
<missing>           4 years ago         /bin/sh -c
<missing>           4 years ago
      2.31GB              Imported from
http://10.137.67.190/download/os/AliOS5U7-x86-64.tgz
```

save

有这么一种场景，有时候我们要**将一台机器的本地镜像导入到另外一台机器**，当然你可以将镜像先 push 到镜像仓库中心，然后另外一个机器再进行 pull。但是有的时候由于镜像的安全性或者镜像比较大，不是很适合这种先 push 再 pull 的场景，那么我们就**可以将镜像先导出成压缩文件，然后再将压缩文件导入到另外一个机器**。其中镜像导出成压缩文件，就是 docker save 做的事情。

```
[root@xxx ~]

Usage:  docker save [OPTIONS] IMAGE [IMAGE...]

Save one or more images to a tar archive (streamed to STDOUT by default)

Options:
  -o, --output string   Write to a file, instead of STDOUT
```

默认是输出到 stdout，当然也可以通过 -o 参数指定输出的文件。那么我们可以通过如下两个方式进行导出。

```
docker save busybox:latest > busybox-latest.tar
```

或者

```
docker save busybox:latest -o busybox-latest.tar
```

import

docker import 以及下面的 load 都是用来从压缩中导入镜像。使用方式也比较简单。

```
[root@emr-header-1 ~]

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

Import the contents from a tarball to create a filesystem image

Options:
  -c, --change list      Apply Dockerfile instruction to the created image
  -m, --message string   Set commit message for imported image
```

比如我们导入上面导出的镜像 tar 包，可以使用下面的命令。

```
[root@emr-header-1 ~]
sha256:1278080eee0524c6ca5c1de63ea439deb0e6d62035549cca9dbd4d9129e38655
```

load

通过 tar 包导入镜像，使用方式。

```
[root@xxx ~]

Usage:  docker load [OPTIONS]

Load an image from a tar archive or STDIN

Options:
  -i, --input string   Read from tar archive file, instead of STDIN
  -q, --quiet          Suppress the load output
```

比如我要将上面导出的镜像压缩包导入，可以通过下面的命令完成。

```
$ [root@xxx ~]
Loaded image: busybox:latest
```

rmi

docker rmi 可以用来删除镜像。使用方式如下：

```
[root@xxx ~]

Usage:  docker rmi [OPTIONS] IMAGE [IMAGE...]

Remove one or more images

Options:
  -f, --force      Force removal of the image
      --no-prune   Do not delete untagged parents
```

其中 IMAGE 我们可以直接使用镜像名称 + tag 或者镜像 ID 的方式来知道，上面的说明文档我们看到还支持两个参数：

- force：强制删除；
- no-prune：不要删除未带标签的父镜像。

tag

docker tag 可以用来给镜像打 tag，目标镜像可以使用镜像名称 + tag 或者镜像 ID 的方式，如下：

```
docker tag busybox:v1 busybox:v2
```

或者

```
docker tag <image_id> busybox:v2
```

3. 镜像仓库操作

和镜像仓库（registry）相关的操作重要有 4 个操作命令：

- login: 登录镜像仓库；
- logout: 登出镜像仓库；
- pull: 从镜像仓库拉取镜像；
- push: 向镜像仓库 push 镜像，需要先 login。

login

login 的使用比较简单，我们可以直接通过 help 看一下使用说明，如下：

```
root@xxx

Usage:  docker login [OPTIONS] [SERVER]

Log in to a Docker registry

Options:
  -p, --password string   Password
      --password-stdin    Take the password from stdin
  -u, --username string   Username
```

如上所示 `docker login [OPTION] [SERVER]`，其中 SERVER 就是仓库地址，OPTION 是用户名密码，比如我要登入阿里云杭州的镜像仓库，可以像如下操作：

```
$ docker login -u legendtkl -p <password> registry.cn-hangzhou.aliyuncs.com
```

但是一般我们不太建议将密码直接展示在命令行中，这样别的用户可以通过 history 直接看到，所以我们一般都是省略 -p 参数，由键盘输入密码，如下：

```
$ docker login -u legendtkl registry.cn-hangzhou.aliyuncs.com
Password:
```

logout

logout 就比较简单了，不需要任何参数，直接接仓库地址即可，如下。

```
root@xxx

Usage:  docker logout [SERVER]

Log out from a Docker registry
```

pull

拉取镜像也比较简单，如下

```
root@xxx

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

Pull an image or a repository from a registry

Options:
  -a, --all-tags                Download all tagged images in the repository
      --disable-content-trust   Skip image verification (default true)
```

我们**可以通过镜像名称后面加 Tag 或者 Digest 来拉取指定版本的镜像**，如果不指定则拉取 latest 这个标签的。那么什么是 Tag 和 Digest 呢?

我们可以这样理解，Tag 和 Digest 都是镜像版本的一种唯一性标识。Tag 可以理解成 Git 里面的 Tag，Digest 是镜像文件的摘要，一般是 sha256 散列计算出来的值。当然我们正常 pull 的时候都是指定 Tag，很少会指定 Digest，举个例子:

```
docker pull busybox:1.23
docker pull
busybox@sha256:2824fe048727a69da66cf1be00cebd3bb9cfe1f238473693aa9358b411208527
```

有一种情况需要注意的是，**尽量不要使用 latest 这个 Tag。**顾名思义，latest 这个标签表示最新的镜像，换言之，也就是**会进行变化的**，这种情况是万万不能用于生产环境的。

我们看到拉取的镜像的时候还有两个参数： all-tags 和 disable-content-trust。第一个参数会下载镜像的全部标签，一般不会这么用。第二个参数是用来跳过镜像的校验。

push

向仓库 push 镜像，参数和 pull 类似。有一点需要注意的是，如果 push 的镜像在仓库中已经存在，则会覆盖已经存在的镜像。

```
root@xxx

Usage:  docker push [OPTIONS] NAME[:TAG]

Push an image or a repository to a registry

Options:
      --disable-content-trust   Skip image signing (default true)
```

4. 总结

本篇文章介绍了 docker 的常用命令，涉及到容器、镜像和镜像仓库中心。掌握了上面的命令，基本使用 docker 不会有什么问题。当然由于 docker 的命令及其参数之多，这里并不能完全展示出来，关于更多信息，大家可以参考下面的参考链接。

5. 参考

```
}
```