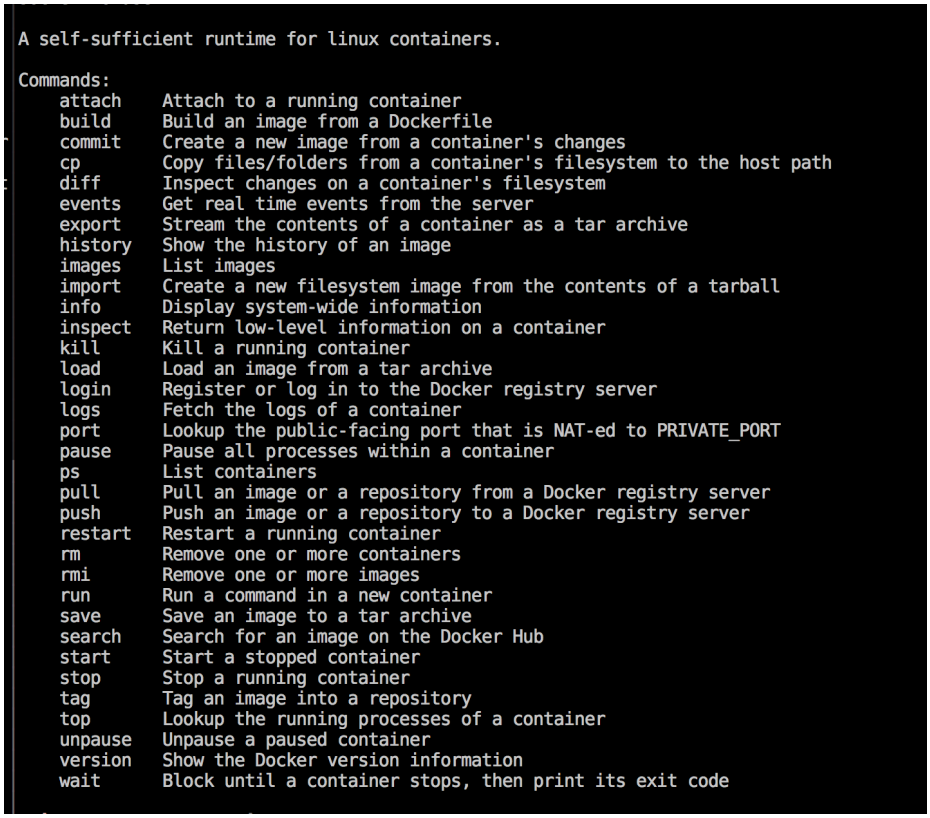
深入浅出Docker（二）：Docker命令行探秘

1. Docker命令行

Docker官方为了让用户快速了解Docker，提供了一个交互式教程，旨在帮助用户掌握Docker命令行的使用方法。但是由于Docker技术的快速发展，此交互式教程已经无法满足Docker用户的实际使用需求，所以让我们一起开始一次真正的命令行学习之旅。首先，Docker的命令清单可以通过运行 *docker* ，或者 *docker help* 命令得到:

$ sudo docker



在Docker容器技术不断演化的过程中，Docker的子命令已经达到34个之多，其中核心子命令(例如：run)还会有复杂的参数配置。笔者通过结合功能和应用场景方面的考虑，把命令行划分为4个部分，方便我们快速概览Docker命令行的组成结构：

功能划分	命令
环境信息相关	1. info 2. version

系统运维相关	1. attach 2. build 3. commit 4. cp 5. diff 6. export 7. images 8. import / save / load 9. inspect 10. kill 11. port 12. pause / unpause 13. ps 14. rm 15. rmi 16. run 17. start / stop / restart 18. tag 19. top 20. wait
日志信息相关	1. events 2. history 3. logs
Docker Hub服务相关	1. login 2. pull / push 3. search

1.1 参数约定

单个字符的参数可以放在一起组合配置，例如

```
docker run -t -i --name test busybox sh
```

可以用这样的方式等同：

```
docker run -ti --name test busybox sh
```

1.2 Boolean

Boolean参数形式如： -d=false。注意，当你声明这个Boolean参数时，比如 docker run -d=true，它将直接把启动的Container挂起放在后台运行。

1.3 字符串和数字

参数如 --name= “” 定义一个字符串，它仅能被定义一次。同类型的如-c=0 定义一个数字，它也只能被定义一次。

1.4 后台进程

Docker后台进程是一个常驻后台的系统进程，值得注意的是Docker使用同一个文件来支持客户端和后台进程，其中角色切换通过-d来实现。这个后台进程是用来管理容器的，使用Docker --help可以得到更详细的功能参数配置, 如下图：

Docker后台进程参数清单如下表：

参数	解释
--api-enable-cors=false	开放远程API调用的 CORS 头信息。这个接口开关对想进行二次开发的上层应用提供了支持。
-b, --bridge=""	挂载已经存在的网桥设备到 Docker 容器里。注意，使用 **none** 可以停用容器里的网络。
--bip=""	使用 CIDR 地址来设定网络桥的 IP。注意，此参数和 -b 不能一起使用。
-D, --debug=false	开启Debug模式。例如：docker -d -D
-d, --daemon=false	开启Daemon模式。
--dns=[]	强制容器使用DNS服务器。例如： docker -d --dns 8.8.8.8
--dns-search=[]	强制容器使用指定的DNS搜索域名。例如： docker -d --dns-search example.com
-e, --exec-driver="native"	强制容器使用指定的运行时驱动。例如：docker -d -e lxc
-G, --group="docker"	在后台运行模式下，赋予指定的Group到相应的unix socket上。注意，当此参数 --group 赋予空字符串时，将去除组信息。
-g, --graph="/var/lib/docker"	配置Docker运行时根目录

-H, --host=[]	在后台模式下指定socket绑定，可以绑定一个或多个 tcp://host:port, unix:///path/to/socket, fd://* 或 fd://socketfd。例如：
	$ docker -H tcp://0.0.0.0:2375 ps 或者
	$ export DOCKER_HOST="tcp://0.0.0.0:2375"
	$ docker ps

--icc=true	启用内联容器的通信。
--ip="0.0.0.0"	容器绑定IP时使用的默认IP地址
--ip-forward=true	启动容器的 net.ipv4.ip_forward
--iptables=true	启动Docker容器自定义的iptable规则
--mtu=0	设置容器网络的MTU值，如果没有这个参数，选用默认 route MTU，如果没有默认 route，就设置成常量值 1500。
-p, --pidfile="/var/run/docker.pid"	后台进程PID文件路径。
-r, --restart=true	重启之前运行中的容器

-s, --storage-driver=""	强制容器运行时使用指定的存储驱动，例如,指定使用devicemapper, 可以这样： docker -d -s devicemapper
--selinux-enabled=false	启用selinux支持
--storage-opt=[]	配置存储驱动的参数
--tls=false	启动TLS认证开关
--tlscacert="/Users/dxiao/.docker/ca.pem"	通过CA认证过的的certificate文件路径
--tlscert="/Users/dxiao/.docker/cert.pem"	TLS的certificate文件路径
--tlskey="/Users/dxiao/.docker/key.pem"	TLS的key文件路径
--tlsverify=false	使用TLS并做后台进程与客户端通讯的验证
-v, --version=false	显示版本信息

注意，其中带有[] 的启动参数可以指定多次，例如

```
$ docker run -a stdin -a stdout -a stderr -i -t ubuntu /bin/bash
```

2. Docker命令行探秘

2.1 环境信息相关

info

使用方法： docker info

例子：

```
[fedora@docker-devel-cli docker]$ sudo docker -D info
Containers: 0
Images: 32
Storage Driver: devicemapper
 Pool Name: docker-252:1-130159-pool
 Data file: /var/lib/docker/devicemapper/devicemapper/data
 Metadata file: /var/lib/docker/devicemapper/devicemapper/metadata
 Data Space Used: 1616.9 Mb
 Data Space Total: 102400.0 Mb
 Metadata Space Used: 2.4 Mb
 Metadata Space Total: 2048.0 Mb
Execution Driver: native-0.2
Kernel Version: 3.11.10-301.fc20.x86_64
Debug mode (server): false
Debug mode (client): true
Fds: 11
Goroutines: 14
EventsListeners: 0
Init SHA1: 2c5adb59737b8a01fa3fb968519a43fe140bc9c9
Init Path: /usr/libexec/docker/dockerinit
Sockets: [fd://]
```

使用说明：

这个命令在开发者报告Bug时会非常有用，结合docker vesion一起，可以随时使用这个命令把本地的配置信息提供出来，方便Docker的开发者快速定位问题。

version

使用方法： docker version

使用说明：

显示Docker的版本号，API版本号，Git commit， Docker客户端和后台进程的Go版本号。

2.2 系统运维相关

attach

使用方法： docker attach [OPTIONS] CONTAINER

例子：

```
$ ID=$(sudo docker run -d ubuntu /usr/bin/top -b)
$ sudo docker attach $ID
top - 17:21:49 up  5:53,  0 users,  load average: 0.63, 1.15, 0.78
Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.0 us,  0.7 sy,  0.0 ni, 97.7 id,  0.7 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   2051644 total,   723700 used,  1327944 free,    33032 buffers
KiB Swap:   0 total,   0 used,  0 free.   565836 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
    1 root      20   0   19748   1280   1008 R  0.0  0.1   0:00.04 top
$ sudo docker stop $ID
```

使用说明：

使用这个命令可以挂载正在后台运行的容器，在开发应用的过程中运用这个命令可以随时观察容器内进程的运行状况。开发者在开发应用的场景中，这个命令是一个非常有用的命令。

build

使用方法：docker build [OPTIONS] PATH | URL | -

例子：

```
$ docker build .
Uploading context 18.829 MB
Uploading context
Step 0 : FROM busybox
 ---> 769b9341d937
Step 1 : CMD echo Hello world
 ---> Using cache
 ---> 99cc1ad10469
Successfully built 99cc1ad10469
```

使用说明：

这个命令是从源码构建新Image的命令。因为Image是分层的，最关键的Base Image是如何构建的是用户比较关心的，Docker官方文档给出了构建方法，请参考这里。

commit

使用方法：docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

例子：

```
$ sudo docker ps
ID                  IMAGE               COMMAND             CREATED             STATUS              PORTS
c3f279d17e0a        ubuntu:12.04        /bin/bash           7 days ago          Up 25 hours
197387f1b436        ubuntu:12.04        /bin/bash           7 days ago          Up 25 hours
$ docker commit c3f279d17e0a  SvenDowideit/testimage:version3
f5283438590d
$ docker images | head
REPOSITORY                      TAG                 ID                  CREATED             VIRTUAL SIZE
SvenDowideit/testimage          version3            f5283438590d        16 seconds ago      335.7 MB
```

使用说明：

这个命令的用处在于把有修改的container提交成新的Image，然后导出此Imange分发给其他场景中调试使用。Docker官方的建议 是，当你在调试完Image的问题后，应该写一个新的Dockerfile文件来维护此Image。commit命令仅是一个临时创建Imange的辅助 命令。

cp

使用方法： cp CONTAINER:PATH HOSTPATH

使用说明：

使用cp可以把容器内的文件复制到Host主机上。这个命令在开发者开发应用的场景下，会需要把运行程序产生的结果复制出来的需求，在这个情况下就可以使用这个cp命令。

diff

使用方法：docker diff CONTAINER

例子：

```
$ sudo docker diff 7bb0e258aefe

C /dev
A /dev/kmsg
C /etc
A /etc/mtab
A /go
A /go/src
A /go/src/github.com
A /go/src/github.com/dotcloud
....
```

使用说明：

diff会列出3种容器内文件状态变化（A - Add, D - Delete, C - Change ）的列表清单。构建Image的过程中需要的调试指令。

export

使用方法：docker export CONTAINER

例子：

```
$ sudo docker export red_panda > latest.tar
```

使用说明：

把容器系统文件打包并导出来，方便分发给其他场景使用。

images

使用方法：docker images [OPTIONS] [NAME]

例子：

```
$ sudo docker images | head
REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE
<none> <none> 77af4d6b9913 19 hours ago 1.089 GB
committest latest b6fa739cedf5 19 hours ago 1.089 GB
<none> <none> 78a85c484f71 19 hours ago 1.089 GB
$ docker latest 30557a29d5ab 20 hours ago 1.089 GB
<none> <none> 0124422dd9f9 20 hours ago 1.089 GB
<none> <none> 18ad6fad3402 22 hours ago 1.082 GB
<none> <none> f9f1e26352f0 23 hours ago 1.089 GB
tryout latest 2629d1fa0b81 23 hours ago 131.5 MB
<none> <none> 5ed6274db6ce 24 hours ago 1.089 GB
```

使用说明：

Docker Image是多层结构的，默认只显示最顶层的Image。不显示的中间层默认是为了增加可复用性、减少磁盘使用空间，加快build构建的速度的功能，一般用户不需要关心这个细节。

import / save / load

使用方法：

```
docker import URL|- [REPOSITORY[:TAG]]
docker save IMAGE
docker load
```

使用说明：

这一组命令是系统运维里非常关键的命令。加载(两种方法: import, load)，导出(一种方法: save)容器系统文件。

inspect

使用方法：

```
docker inspect CONTAINER|IMAGE [CONTAINER|IMAGE...]
```

例子：

```
$ sudo docker inspect --format='{{.NetworkSettings.IPAddress}}' $INSTANCE_ID
```

使用说明：

查看容器运行时详细信息的命令。了解一个Image或者Container的完整构建信息就可以通过这个命令实现。

kill

使用方法：

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

使用说明：

杀掉容器的进程。

port

使用方法：

```
docker port CONTAINER PRIVATE_PORT
```

使用说明：

打印出Host主机端口与容器暴露出的端口的NAT映射关系

pause / unpause

使用方法：

```
docker pause CONTAINER
```

使用说明：

使用cgroup的freezer顺序暂停、恢复容器里的所有进程。详细freezer的特性，请参考官方文档。

ps

使用方法：

```
docker ps [OPTIONS]
```

例子：

```
$ docker ps
CONTAINER ID        IMAGE                          COMMAND               CREATED             STATUS              PORTS
            NAMES
4c01db0b339c        ubuntu:12.04                   bash                  17 seconds ago      Up 16 seconds
            webapp
d7886598dbe2        crosbymichael/redis:latest     /redis-server --dir   33 minutes ago      Up 33 minutes       6379/tc
p           redis,webapp/db
```

使用说明：

docker ps打印出正在运行的容器，docker ps -a打印出所有运行过的容器。

rm

使用方法：

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

例子：

```
$ sudo docker rm /redis
/redis
```

使用说明：

删除指定的容器。

rmi

使用方法：

```
docker rmi IMAGE [IMAGE...]
```

例子：

```
$ sudo docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
test1 latest fd484f19954f 23 seconds ago 7 B (virtual 4.964 MB)
test latest fd484f19954f 23 seconds ago 7 B (virtual 4.964 MB)
test2 latest fd484f19954f 23 seconds ago 7 B (virtual 4.964 MB)

$ sudo docker rmi fd484f19954f
Error: Conflict, cannot delete image fd484f19954f because it is tagged in multiple repositories
2013/12/11 05:47:16 Error: failed to remove one or more images

$ sudo docker rmi test1
Untagged: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8

$ sudo docker rmi test2
Untagged: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8

$ sudo docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
test latest fd484f19954f 23 seconds ago 7 B (virtual 4.964 MB)

$ sudo docker rmi test
Untagged: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8
Deleted: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8
```

使用说明：

指定删除Image文件。

run

使用方法：

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

例子：

```
$ sudo docker run --cidfile /tmp/docker_test.cid ubuntu echo "test"
```

使用说明：

这个命令是核心命令，可以配置的参数多达28个参数。详细的解释可以通过docker run --help列出。官方文档中提到的Issue 2702："lxc-start: Permission denied - failed to mount" could indicate a permissions problem with AppArmor. 在最新版本的Dcoker中已经解决。

start / stop / restart

使用方法：

```
docker start CONTAINER [CONTAINER...]
```

使用说明：

这组命令可以开启(两个：start, restart)，停止(一个：stop)一个容器。

tag

使用方法：

```
docker tag [OPTIONS] IMAGE[:TAG] [REGISTRYHOST/][USERNAME/]NAME[:TAG]
```

使用说明：

组合使用用户名，Image名字，标签名来组织管理Image。

top

使用方法：

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

使用说明：

显示容器内运行的进程。

wait

使用方法：

```
docker wait CONTAINER [CONTAINER...]
```

使用说明：

阻塞对指定容器的其他调用方法，直到容器停止后退出阻塞。

2.3 日志信息相关

events

使用方法：

```
docker events [OPTIONS]
```

使用说明：

打印容器实时的系统事件。

history

使用方法：

```
docker history [OPTIONS] IMAGE
```

例子：

```
$ docker history docker
IMAGE CREATED CREATED BY SIZE
3e23a5875458790b7a806f95f7ec0d0b2a5c1659bfc899c89f939f6d5b8f7094 8 days ago
/bin/sh -c #(nop) ENV LC_ALL=C.UTF-8 0 B
8578938dd17054dce7993d21de79e96a037400e8d28e15e7290fea4f65128a36 8 days ago
/bin/sh -c dpkg-reconfigure locales && locale-gen C.UTF-8 &&
/usr/sbin/update-locale LANG=C.UTF-8 1.245 MB
be51b77efb42f67a5e96437b3e102f81e0a1399038f77bf28cea0ed23a65cf60 8 days ago /bin/sh
-c apt-get update && apt-get install -y git libxml2-dev python build-essential
make gcc python-dev locales python-pip 338.3 MB
4b137612be55ca69776c7f30c2d2dd0aa2e7d72059820abf3e25b629f887a084 6 weeks ago
/bin/sh -c #(nop) ADD jessie.tar.xz in / 121 MB
750d58736b4b6cc0f9a9abe8f258cef269e3e9dceced1146503522be9f985ada 6 weeks ago
/bin/sh -c #(nop) MAINTAINER Tianon Gravi <admwiggin@gmail.com>
 - mkimage-debootstrap.sh -t jessie.tar.xz jessie http://http.debian.net/debian 0 B
511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158 9 months ago 0 B
```

使用说明：

打印指定Image中每一层Image命令行的历史记录。

logs

使用方法：

```
docker logs CONTAINER
```

使用说明：

批量打印出容器中进程的运行日志。

2.4 Dcoker Hub服务相关

login

使用方法：

```
docker login [OPTIONS] [SERVER]
```

使用说明：

登录Hub服务。

pull / push

使用方法：

```
docker push NAME[:TAG]
```

使用说明：

通过此命令分享Image到Hub服务或者自服务的Registry服务。

search

使用方法：

```
docker search TERM
```

使用说明：

通过关键字搜索分享的Image。

3. 总结

通过以上Docker命令行的详细解释，可以强化对Docker命令的全面理解。考虑到Docker命令行的发展变化非常快，读者可以参考官方的命令行解释文档更新相应的命令行解释。另外，通过以上Docker命令行的分析，可以知道Docker命令行架构设计的特点在于客户端和服务端的运行文件是同一个文件，内部实现代码应该是重用的设计。笔者希望开发者在开发类似的命令行应用时参考这样的设计，减少前后台容错的复杂度。

参考文献

[1] https://docs.docker.com/reference/commandline/cli/

[2] https://en.wikipedia.org/wiki/Cross-Origin_Resource_Sharing

[3] https://en.wikipedia.org/wiki/CIDR_notation#CIDR_notation