Docker容器基础与命令汇总

文档说明：

- OS与Docker版本：CentOS 7.2、***Docker 1.13.1***，未做特殊说明均为以上版本。
- 本文档为了简化完整容器镜像名称，直接使用了*image_name*。
- 本文档以Docker 1.13.1版本为基准，Docker 17.x及更高版本架构与配置文件发生改变，

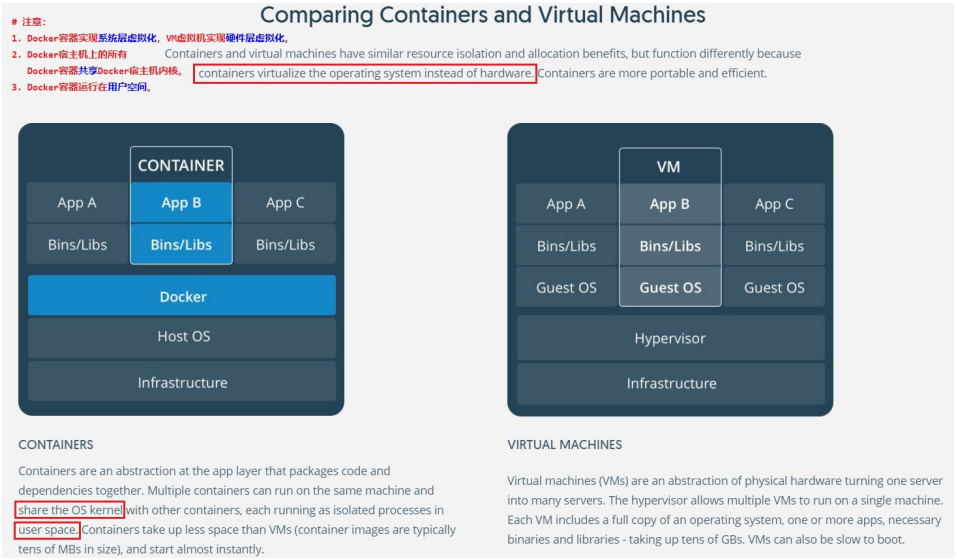
  使用与配置存在差异，但docker命令行使用方式基本一致！

- Docker公司将Docker项目贡献给社区后，成立 ***Moby*** 项目（Docker CE）。

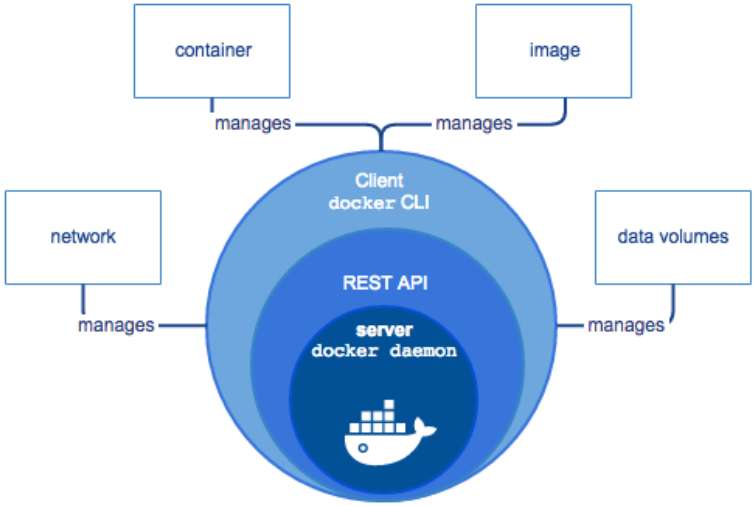
Docker容器基础概念：

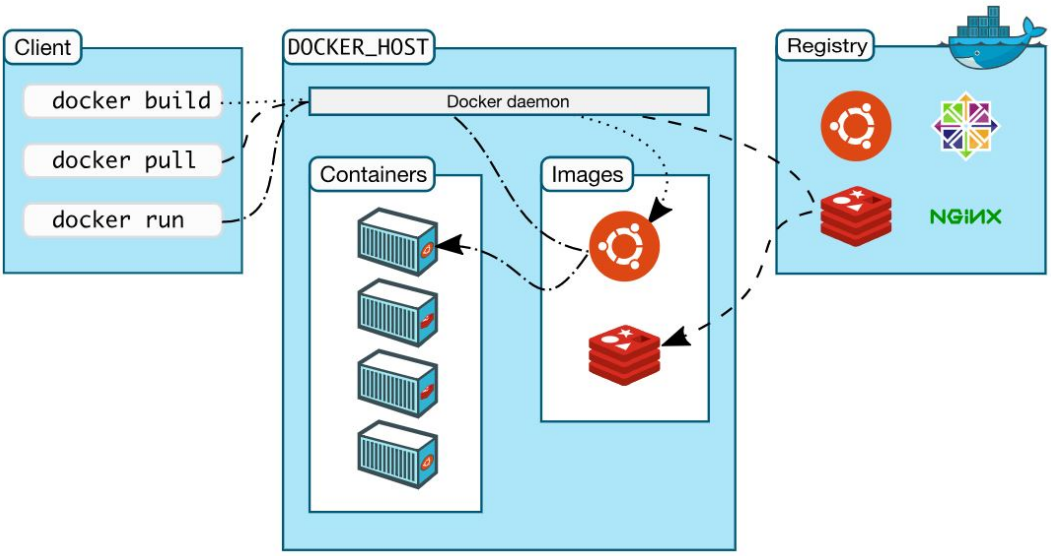
- 大型应用不适合实现容器化，如e-mail、CRM（客户关系管理系统）等。
- Docker容器（container）与镜像（image）特点：

  1．**Docker容器与系统之间共享Linux内核，实现轻量级虚拟化。**
  2．Docker容器占用资源较少，可实现大规模高密度部署。
  3．Docker使用C/S架构来实现，docker命令与docker守护进程相互通讯，管理Docker容器与
  镜像的生命周期。
  4．Docker使用Linux namespace机制与Docker宿主机间实现资源隔离。
  5．Docker使用cgroup资源管理机制实现Docker容器间的资源配额分配（quota）。
  6．Docker使用联合文件系统（UnionFS）实现各镜像层（layer）间的联合挂载，区别默认的
  Ubuntu的AUFS与RHEL的DeviceMapper存储驱动机制。
  7．**Docker镜像层（layer）越多，Docker容器的I/O性能将下降，不适合I/O负载密集的任务。**
  8．Docker镜像最多支持128个层。
  9．Docker容器ID（container_id）与Docker镜像ID（image_id）使用docker命令时只显示
  前12位（短UUID），而完整的ID共64位（长UUID）。



- **Docker**组件的逻辑关系：





- **Docker容器的本质：**

1. Linux namespace命名空间分类：
   PID、user、mount、network、UTS、IPC
2. 被**Linux namespace**命名空间以及**cgroup**控制组进行资源隔离与限制的进程，且该进程
   实际运行于宿主机系统中。
3. 实现容器的3个关键技术：**namespace、cgroup、rootfs**

```
root@myubuntu:/sys/fs/cgroup/pids/docker/b48c5eaa1da7eaae81ed12fdd7c38831bcb30212b43dc9a4754dfaebb4501673# cat tasks
3061
3283 # 容器进程的本质：运行在docker容器中的进程号，该进程使用cgroup控制组限制，且该PID为宿主机中的真实PID，docker容器使用pid namespace进行隔离。
root@myubuntu:/sys/fs/cgroup/pids/docker/b48c5eaa1da7eaae81ed12fdd7c38831bcb30212b43dc9a4754dfaebb4501673# ps -ef | egrep '3061|3283'
root        3061    3032  0 10:26 ?        00:00:00 nginx: master process nginx -g daemon off;   # docker容器中的进程实际运行于宿主机中
systemd+    3283    3061  0 10:27 ?        00:00:00 nginx: worker process
root        4572    4499  0 11:03 pts/0    00:00:00 grep -E --color=auto 3061|3283
root@myubuntu:/sys/fs/cgroup/pids/docker/b48c5eaa1da7eaae81ed12fdd7c38831bcb30212b43dc9a4754dfaebb4501673# docker exec -it docker-doc /bin/sh
/ # top
Mem: 2045872K used, 1969812K free, 39260K shrd, 21736K buff, 967584K cached
CPU:   0% usr   0% sys   0% nic 100% idle   0% io   0% irq   0% sirq
Load average: 0.00 0.01 0.08 1/488 18
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
    6     1 nginx    S    14300   0%   1   0% nginx: worker process
    1     0 root     S    13812   0%   1   0% nginx: master process nginx -g daemon off;   # docker容器中查看进程时，已由pid namespace
   13     0 root     S     1596   0%   1   0% /bin/sh                                      进行隔离，docker容器中的进程号进行重新分配。
   18    13 root     R     1524   0%   1   0% top
/ # exit
root@myubuntu:/sys/fs/cgroup/pids/docker/b48c5eaa1da7eaae81ed12fdd7c38831bcb30212b43dc9a4754dfaebb4501673#
```

Docker权限配置汇总：

- 本地运行权限：

1. root用户可直接运行本地docker命令与docker守护进程。
2. 普通用户需添加入docker用户组中才能运行docker命令与docker守护进程通信，
   但添加入docker用户组中的所有普通用户通能运行docker命令，存在较大安全风险！
3. 建议不添加普通用户至docker用户组，普通用户使用**sudo**命令来运行**docker**命令。

   * 注意：
     1. 若攻击者获取docker用户组权限，即可获取宿主机root权限。
     2. 若攻击者获取docker用户组中的普通用户权限，也可获取主机root权限。

4. 添加**docker**用户组及其组用户

```
[root@cloud-ctl ~]# groupadd -g 8000 docker
[root@cloud-ctl ~]# mkdir /home/dockeruser
[root@cloud-ctl ~]# useradd -m -d /home/dockeruser -g docker -u 8000 dockeruser    # 创建docker用户组及其组用户dockeruser
useradd: warning: the home directory already exists.
Not copying any file from skel directory into it.
[root@cloud-ctl ~]# cp -avr /etc/skel/.[[:alpha:]]* /home/dockeruser
‘/etc/skel/.bash_logout’ -> ‘/home/dockeruser/.bash_logout’
‘/etc/skel/.bash_profile’ -> ‘/home/dockeruser/.bash_profile’
‘/etc/skel/.bashrc’ -> ‘/home/dockeruser/.bashrc’
‘/etc/skel/.mozilla’ -> ‘/home/dockeruser/.mozilla’
‘/etc/skel/.mozilla/extensions’ -> ‘/home/dockeruser/.mozilla/extensions’
‘/etc/skel/.mozilla/plugins’ -> ‘/home/dockeruser/.mozilla/plugins’
[root@cloud-ctl ~]# chown -R dockeruser:docker /home/dockeruser     # 更改并查看dockeruser用户家目录属组
[root@cloud-ctl ~]# ls -ld /home/dockeruser
drwxr-xr-x 3 dockeruser docker 74 Aug 18 16:11 /home/dockeruser
[root@cloud-ctl ~]# gpasswd -a hualf docker     # 添加hualf用户至docker用户组中
Adding user hualf to group docker
[root@cloud-ctl ~]# grep dockeruser /etc/passwd
dockeruser:x:8000:8000::/home/dockeruser:/bin/bash
[root@cloud-ctl ~]# grep docker /etc/group    # 查看docker用户组中的用户信息
dockerroot:x:983:
docker:x:8000:hualf
[root@cloud-ctl ~]# ls -l /var/run/docker.sock
srw-rw---- 1 root root 0 Aug 18 14:55 /var/run/docker.sock     # docker守护进程的本地套接字文件，用于监听本地docker客户端请求。
[root@cloud-ctl ~]#
```

```
[root@cloud-ctl ~]# docker images
REPOSITORY                                      TAG                 IMAGE ID            CREATED             SIZE
ocp-registry.domain12.example.com:5000/centos   7.2.1511            ddc0fb7d7a72        11 days ago         195 MB
ocp-registry.domain12.example.com:5000/centos   7.4.1708            d3949e34634c        11 days ago         197 MB
ocp-registry.domain12.example.com:5000/registry 2.6.2               b2b03e9146e1        6 weeks ago         33.3 MB
[root@cloud-ctl ~]# su - dockeruser
[dockeruser@cloud-ctl ~]$ docker images       # 切换至docker用户组中的普通用户，该用户可直接使用docker命令。
REPOSITORY                                      TAG                 IMAGE ID            CREATED             SIZE
ocp-registry.domain12.example.com:5000/centos   7.2.1511            ddc0fb7d7a72        11 days ago         195 MB
ocp-registry.domain12.example.com:5000/centos   7.4.1708            d3949e34634c        11 days ago         197 MB
ocp-registry.domain12.example.com:5000/registry 2.6.2               b2b03e9146e1        6 weeks ago         33.3 MB
[dockeruser@cloud-ctl ~]$ exit
logout
[root@cloud-ctl ~]# su - hualf              # 切换至docker用户组中的另一普通用户，该用户也可直接使用docker命令。
Last login: Fri May 18 13:33:35 CST 2018 on pts/2
[hualf@cloud-ctl ~]$ docker images
REPOSITORY                                      TAG                 IMAGE ID            CREATED             SIZE
ocp-registry.domain12.example.com:5000/centos   7.2.1511            ddc0fb7d7a72        11 days ago         195 MB
ocp-registry.domain12.example.com:5000/centos   7.4.1708            d3949e34634c        11 days ago         197 MB
ocp-registry.domain12.example.com:5000/registry 2.6.2               b2b03e9146e1        6 weeks ago         33.3 MB
```

- 远程运行权限：

1. docker守护进程监听远程docker客户端的RESTful API请求。
2. 远程docker客户端上的所有用户都能执行docker命令，存在较大安全风险，生产环境中禁用。
3. **docker客户端连接远程docker守护进程：**

```
[root@ocp-registry ~]# docker images                                                    # 本地docker缓存中未下载容器镜像
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
[root@ocp-registry ~]# docker -H tcp://192.168.0.152:4200 images      # docker客户端查看远程docker宿主机的容器镜像，需指定IP与端口号。
REPOSITORY                                      TAG                 IMAGE ID            CREATED             SIZE
ocp-registry.domain12.example.com:5000/centos   7.2.1511            ddc0fb7d7a72        11 days ago         195 MB
ocp-registry.domain12.example.com:5000/centos   7.4.1708            d3949e34634c        11 days ago         197 MB
ocp-registry.domain12.example.com:5000/registry 2.6.2               b2b03e9146e1        6 weeks ago         33.3 MB
[root@ocp-registry ~]# export DOCKER_HOST='tcp://192.168.0.152:4200'  # 导出远程docker守护进程宿主机的环境变量：DOCKER_HOST
[root@ocp-registry ~]# docker images     # 导出DOCKER_HOST环境变量后可直接运行docker命令
REPOSITORY                                      TAG                 IMAGE ID            CREATED             SIZE
ocp-registry.domain12.example.com:5000/centos   7.2.1511            ddc0fb7d7a72        11 days ago         195 MB
ocp-registry.domain12.example.com:5000/centos   7.4.1708            d3949e34634c        11 days ago         197 MB
ocp-registry.domain12.example.com:5000/registry 2.6.2               b2b03e9146e1        6 weeks ago         33.3 MB
[root@ocp-registry ~]# echo $DOCKER_HOST
tcp://192.168.0.152:4200
[root@ocp-registry ~]# unset DOCKER_HOST      # 清除DOCKER_HOST环境变量
[root@ocp-registry ~]# echo $DOCKER_HOST

[root@ocp-registry ~]# docker images    # 清除DOCKER_HOST环境变量后，只能查看本地docker缓存中的容器镜像。
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
[root@ocp-registry ~]#
```

Docker常用配置文件汇总：

- docker守护进程的驱动程序：RHEL/CentOS/Fedora

1. 运行时驱动（runtime driver）：
   docker-runc、*runc*
2. 存储驱动（storage driver）：
   overlay、**overlay2**、**devicemapper**、aufs *or* btrfs *or* zfs in ubuntu
3. 日志驱动（logging driver）：
   none、json-file（ubuntu默认）、journald（RHEL默认）、syslog、*fluentd*、*splunk*
4. 控制组驱动（cgroup driver）：
   systemd、*cgroupfs*（该驱动从Docker 17.x开始支持）

- docker守护进程配置文件：

1. **/usr/lib/systemd/system/docker.service：**
   docker守护进程的unit文件，其余配置文件的参数集成于该文件中。

```
[Unit]
Description=Docker Application Container Engine
Documentation=http://docs.docker.com
After=network.target rhel-push-plugin.socket registries.service
Wants=docker-storage-setup.service
Requires=docker-cleanup.timer

[Service]
Type=notify
NotifyAccess=all
EnvironmentFile=-/run/containers/registries.conf
EnvironmentFile=-/etc/sysconfig/docker
EnvironmentFile=-/etc/sysconfig/docker-storage
EnvironmentFile=-/etc/sysconfig/docker-network
Environment=GOTRACEBACK=crash
Environment=DOCKER_HTTP_HOST_COMPAT=1
Environment=PATH=/usr/libexec/docker:/usr/bin:/usr/sbin
ExecStart=/usr/bin/dockerd-current \
          --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current \
          --default-runtime=docker-runc \
          --exec-opt native.cgroupdriver=systemd \
          --userland-proxy-path=/usr/libexec/docker/docker-proxy-current \
          --init-path=/usr/libexec/docker/docker-init-current \
          --seccomp-profile=/etc/docker/seccomp.json \
          -H unix:///var/run/docker.sock \
          -H tcp://0.0.0.0:4200 \
          --graph=/var/lib/docker.bak \
          $OPTIONS \
          $DOCKER_STORAGE_OPTIONS \
          $DOCKER_NETWORK_OPTIONS \
          $ADD_REGISTRY \
          $BLOCK_REGISTRY \
          $INSECURE_REGISTRY \
        $REGISTRIES
"/usr/lib/systemd/system/docker.service" 44L, 1449C
```

docker守护进程的主配置文件，存储驱动与网络配置文件，
其中的各项参数可集成引用至该配置文件中。

docker守护进程通过socket文件监听本地docket客户端请求，该socket文件可任意指定。
docker守护进程通过本地TCP 4200端口（未占用）监听远程docker客户端的restful api请求。
指定自定义的docker守护进程的根目录路径，默认为/var/lib/docker，建议创建新的文件系统挂载docker守护进程的根目录。

该参数为各配置文件中定义的参数

2．**/etc/sysconfig/docker**：docker守护进程的主配置文件，可配置常用选项、registry等。

```
# /etc/sysconfig/docker
# 关闭Docker使用的SELinux；配置docker守护进程的日志驱动程序
# Modify these options if you want to change the way the docker daemon runs
OPTIONS='--selinux-enabled=false --log-driver=journald --signature-verification=false'
if [ -z "${DOCKER_CERT_PATH}" ]; then
    DOCKER_CERT_PATH=/etc/docker
fi

# Do not add registries in this file anymore. Use /etc/containers/registries.conf
# from the atomic-registries package.
#                 # 配置未经过CA证书认证的非安全的私有容器镜像仓库（v2 restful api）
INSECURE_REGISTRY='--insecure-registry kube-registry.domain12.example.com:5000'
BLOCK_REGISTRY='--block-registry docker.io --block-registry registry.access.redhat.com'   # 屏蔽两个公共容器镜像仓库
# ADD_REGISTRY='--add-registry ' # 配置经过CA证书认证的安全的私有容器镜像仓库（v2 restful api）

# On an SELinux system, if you remove the --selinux-enabled option, you
# also need to turn on the docker_transition_unconfined boolean.
# setsebool -P docker_transition_unconfined 1

# Location used for temporary files, such as those created by
# docker load and build operations. Default is /var/lib/docker/tmp
# Can be overriden by setting the following environment variable.
# DOCKER_TMPDIR=/var/tmp

# Controls the /etc/cron.daily/docker-logrotate cron job status.
# To disable, uncomment the line below.
# LOGROTATE=false

# docker-latest daemon can be used by starting the docker-latest unitfile.
# To use docker-latest client, uncomment below lines
#DOCKERBINARY=/usr/bin/docker-latest
#DOCKERDBINARY=/usr/bin/dockerd-latest
#DOCKER_CONTAINERD_BINARY=/usr/bin/docker-containerd-latest
#DOCKER_CONTAINERD_SHIM_BINARY=/usr/bin/docker-containerd-shim-latest
"/etc/sysconfig/docker" 34L, 1450C written
```

3．**/etc/sysconfig/docker-storage**：docker守护进程的存储选项配置文件

```
DOCKER_STORAGE_OPTIONS="--storage-driver devicemapper --storage-opt dm.fs=xfs --storage-opt dm.thinpooldev=/dev/mapper/dockervg-docker--pool --storage-opt dm.use_deferred_removal=true --storage-opt dm.use_deferred_deletion=true "
~
~ # docker守护进程的存储驱动配置参数：devicemapper
```

4．**/etc/sysconfig/docker-storage-setup**：docker守护进程的存储配置文件

```
# STORAGE_DRIVER=overlay2          # 默认的docker守护进程存储驱动为overlay2
STORAGE_DRIVER=devicemapper
DEVS=/dev/sdd                      # 单独指定磁盘
VG=dockervg
~   # 指定devicemapper存储驱动创建的资源池（LV thin-pool）的VG名称。
~
```

5. **/etc/sysconfig/docker-network**：docker守护进程的网络选项配置文件

```
# /etc/sysconfig/docker-network
DOCKER_NETWORK_OPTIONS="--bip=172.42.0.1/16"
~
~ # 指定Docker默认网桥docker0的IP地址，默认为172.17.0.1/16。
~
```

6. **/etc/docker/daemon.json**：
   a. JSON格式的docker守护进程的选项配置文件
   b. 也可用于配置常用选项（options）、registry等
      * 注意：各配置文件实现永久生效，而docker命令运行容器只临时生效！

docker命令常用选项汇总：

- docker run命令常用选项：*man docker-run*

-i，--interactive　　启动Docker容器时进入交互模式，退出即关闭容器。
-t，--tty　　生成Docker容器的伪终端（pseudo）tty，常与-i选项并用。
-d，--detach　　启动Docker容器时以守护进程方式运行
-v，--volume　　**目录映射**：映射Docker容器目录至宿主机目录，使数据持久化。

-p，--publish　　**端口映射**：映射Docker容器端口至宿主机端口，使外部能够通过
宿主机访问本地容器。
Dockerfile中 *EXPOSE* 指令只定义容器对外暴露的端口号，
实际对外暴露需使用docker run命令公开（publish）。
Dockerfile中未指定容器对外暴露的端口号，docker run命令
能指定并公开对外暴露的端口号。

-p <*container_port*>　　将宿主机 **49153~65535** 中的随机端口号映射到容器端口上
-p <*host_port*>:<*container_port*>　　指定未被占用的宿主机端口号映射到容器端口上

-p <*host_ip*>:<*host_port*>:<*container_port*>
由于宿主机可具有多个IP地址，未限制IP地址的情况下，均可
通过任意IP地址进行容器的端口映射。
限制只通过指定的宿主机IP地址，将指定的宿主机端口号映射到
容器端口上。

-p <*host_ip*>::<*container_port*>
限制只通过指定的宿主机IP地址，将随机的宿主机端口号映射到
容器端口上。

-P，--publish-all　　公开暴露Dockerfile中EXPOSE指令定义的所有端口号，将随机的
宿主机端口号映射到容器端口上。

-e，--env 指定Docker容器运行时的环境变量
如MySQL容器运行时必须指定环境变量，否则启动报错！

--name='' 指定Docker容器名称，否则Docker引擎将自动随机分配容器名称。
建议使用**Docker容器名称，易于辨别其中应用的功能。**

--hostname='' 指定Docker容器的主机名称，否则Docker引擎将自动随机分配
容器的主机名称。

--log-driver=[none|json-file|journald|syslog|fluentd|splunk]
指定docker守护进程的日志驱动程序，包括none、json-file、
journald、**syslog**、**fluentd**、splunk等。
默认使用 json-file 与 journald 日志驱动程序。

--restart=[no|always|on-failure:<*count*>]，如：--restart=on-failure:5
指定docker守护进程重启容器的策略
--restart选项将检测容器的退出状态码
no：默认参数，不重启容器。
always：忽略容器的退出状态码，始终重启容器。
on-failure:<*count*>：容器的退出状态码为非0时，即容器非
正常退出时重启容器，且可指定重启次数。

--expose=[<*port*>|<*range_of_ports*>]，如：--expose=3300-3310
指定容器镜像对外暴露的端口号，若在Dockerfile中未指定，可
使用该选项指定对外暴露的端口号。

--network=[bridge|<*network_name_or_id*>|host|container:<*name_or_id*>|none]
指定容器运行的网络模式

- docker ps命令常用选项：*man docker-ps*

-a，--all 查看所有的Docker容器信息，包括运行中的和已停止的容器。
-q，--quiet 查看Docker容器的container_id
-n，--last <*number*> 查看最后n个被创建的容器状态，包括运行中的和已停止的容器。
-l，--latest 查看最新创建的容器状态，包括运行中的和已停止的容器。
-s，--size 查看容器的文件大小

```
[godev@cloud-ctl backup]$ docker ps -s
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS
               NAMES               SIZE
3ea752c1cce6        docs/docker.github.io:latest   "/bin/sh -c 'echo -e…"   20 months ago       Up 8 hours          80/tcp, 0.0.0.0:40
00->4000/tcp   docker-ce-18.09-docs   33B (virtual 2.48GB)
[godev@cloud-ctl backup]$
```

- docker logs命令常用选项：

--detail 查看Docker容器日志的详细信息

`--tail <number>` 查看Docker容器日志的最新信息
`-f, --follow` 动态跟踪刷新Docker容器日志，相当于使用tailf命令。
`-t, --timestamps` 为容器日志加上时间戳

- docker inspect命令常用选项：

`-f, --format` 查看运行中或已停止的Docker容器元数据并格式化输出

- docker commit命令常用选项：

`-m, --message` 指定提交新容器镜像时的提交信息
`-a, --author` 指定新容器镜像的作者信息

- docker build命令常用选项：

`-t, --tag` 指定新容器镜像的标签
`--no-cache` 指定构建自定义容器镜像时不使用构建缓存，默认为使用。

- Docker选项与参数优先级：

1. docker命令行 > docker守护进程选项与参数 && Dockerfile中定义的选项与参数
2. **docker守护进程配置文件或Dockerfile中未定义的选项均可在docker命令行中指定**

- 常见的Docker轻量级基础镜像：

scratch、busybox、alpine、ubuntu、debian

Docker自动化安装脚本：

```
-> $ sudo curl -fsSL https://get.docker.com | sh
   # 该自动化安装脚本将安装最新版本的Docker CE
```

Docker container管理命令：

```
-> $ docker <subcommand> --help or man docker-<subcommand>
   # 查看docker命令的使用方法

-> $ docker info
   # 查看docker守护进程与系统的详细信息

-> $ docker run <image_name>:[tag]|<image_id>
   # 以容器内默认命令（default command）在终端上运行容器
   # 终端上的输出为容器内进程的输出，终止该进程即可终止容器运行。

-> $ docker run -it <image_name>:[tag]|<image_id> /bin/bash
   # 运行容器并进入容器的交互模式，运行命令行指定的命令而非容器中的默认命令。
   # 退出容器终端将终止容器运行。

-> $ docker run <image_name>:[tag]|<image_id> <command>
```

容器以指定命令运行，不以容器内默认命令运行，该命令覆盖镜像Dockerfile中的CMD指令。

```
-> $ docker run \
     --log-driver=[json-file|journald|syslog] \
     --name=<container_name> \
     --hostname=<container_hostname> \
     -v <host_directory>:<container_directory> \
     -p <host_port>:<container_port> \
     -e MYSQL_USER=<username> \
     -e MYSQL_PASSWORD=<password> \
     -e MYSQL_DATABASE=<database> \
     -e MYSQL_ROOT_PASSWORD=<root_password> \
     -d <image_name>:[tag]|<image_id>
   # 以docker守护进程方式运行MySQL容器，并指定日志驱动程序。
   # 将MySQL容器目录映射到本地主机目录以永久存储应用数据，并映射容器端口到本地端口，
     使外部可访问。
   # 运行MySQL容器时，必须指定MySQL容器的环境变量，否则无法正确运行并异常退出！
     * 注意：
       1. 未指定容器的名称时，容器的container_id与CONTAINER_HOSTNAME相同。
       2. 若具有相同名称的运行中或已停止的容器，创建容器时，将发生冲突（conflict），
          无法创建该容器，容器名称具有唯一性。
       3. 指定容器的3种方式：短UUID（12位）、长UUID（64位）、容器名称
```

```
[student@workstation ~]$ sudo mkdir -p /var/local/mysql    # 创建用于容器目录映射的Docker宿主机目录，实现容器数据持久化存储。
[student@workstation ~]$ sudo chcon -t svirt_sandbox_file_t /var/local/mysql/     # 更改Docker宿主机目录的SELinux规则，关闭SELinux后无需配置。
[student@workstation ~]$ sudo chown -R 27:27 /var/local/mysql/  # 更改Docker宿主机目录的属组：容器中运行mysql的用户ID，即mysql用户。
[student@workstation ~]$ ls -ldZ /var/local/mysql/
drwxr-xr-x. 27 27 unconfined_u:object_r:svirt_sandbox_file_t:s0 /var/local/mysql/     # 查看Docker宿主机目录的属性
[student@workstation ~]$ ls /var/local/mysql/
[student@workstation ~]$ docker run --name mysql-1 -v /var/local/mysql:/var/lib/mysql/data \   # 运行守护式mysql容器，映射容器目录至Docker宿
> -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 \                                              主机目录，并且必须定义mysql环境变量，否则运行容
> -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 \                                   器报错。
> -d rhscl/mysql-56-rhel7
6e6bee46affd7f01f4ced4d75b16cb27f86c49baa5ab593b7ecd786feb56bb6d
[student@workstation ~]$ docker ps
CONTAINER ID        IMAGE                    COMMAND                  CREATED             STATUS              PORTS               NAMES
6e6bee46affd        rhscl/mysql-56-rhel7     "container-entrypoint"   9 seconds ago       Up 8 seconds        3306/tcp            mysql-1
[student@workstation ~]$ docker inspect -f '{{ .NetworkSettings.IPAddress }}' mysql-1   # 使用Go语言模板，查看mysql容器分配的IP地址。
172.17.0.2
[student@workstation ~]$ mysql -uuser1 -h 172.17.0.2 -pmypa55 items < /home/student/DO180/labs/work-containers/db.sql
[student@workstation ~]$ mysql -uuser1 -h 172.17.0.2 -pmypa55 items -e 'show tables'          # 注入sql语句至本地mysql数据库容器中
+----------------+      # 查看本地mysql数据库容器中的数据表
| Tables_in_items |
+----------------+
| Item           |
+----------------+
[student@workstation ~]$ mysql -uuser1 -h 172.17.0.2 -pmypa55 items -e 'SELECT * FROM Item'
+----+--------------------+------+      # 查看本地mysql数据库容器中的数据表内容，验证sql语句是否成功注入数据库。
| id | description        | done |      # 本地mysql数据库容器的数据通过目录映射，保存于Docker宿主机目录中。
+----+--------------------+------+
|  1 | Pick up newspaper  |      |
|  2 | Buy groceries      |      |
+----+--------------------+------+
[student@workstation ~]$
```

```
-> $ docker attach [<container_name>|<container_id>]
   # 方法1：附着到运行的容器中，即使用交互式终端进入运行中的容器。

-> $ docker exec -it [<container_name>|<container_id>] /bin/bash
   # 方法2：使用交互式终端进入运行中的容器

-> $ docker exec [<container_name>|<container_id>] <command>
   # 在运行的容器中执行命令，该命令必须存在于运行的容器中。

-> $ docker ps
   # 查看运行中的容器信息
   /* EXAMPLE */
   $ docker ps \
     --filter status=running \
```

```
      --format="table {{.ID}}\t{{.Names}}\t{{.Image}}\t{{.Status}}"
    # 格式化输出运行中的容器信息，使用table参数自动识别制表符。

 -> $ docker ps -a
    # 查看运行中与已停止的容器信息

 -> $ docker ps -l
    # 查看最新一次运行的容器信息，该容器可在运行中或已停止。

 -> $ docker ps -n <number>
    # 查看最后n个被创建的容器状态，包括运行中的和已停止的容器。

 -> $ docker ps -aq
    # 查看所有的运行中与已停止的容器ID

 -> $ docker top [<container_name>|<container_id>]
    # 查看容器中运行的进程信息

 -> $ docker stop [<container_name>|<container_id>]
    # 停止运行中的容器，向容器进程发送SIGTERM信号。
    # 停止的容器将临时存储容器的状态与自身文件系统，可用于容器错误运行时排错，
      并在该容器重启时恢复。

 -> $ docker kill [<container_name>|<container_id>]
    # 强制杀死运行中的容器，向容器进程发送SIGKILL信号。

 -> $ docker rm [<container_name>|<container_id>] [-f|--force]
    # 删除已停止的容器，这将永久删除容器layer中的应用数据，即永久删除容器的状
态与
      自身文件系统。
    # --force选项可用于强制删除运行中的容器（非优雅删除容器的方法）

 -> $ docker rm $(docker ps -aq)
    # 批量删除运行中与已停止的全部容器
    # --force选项可用于强制删除运行中的容器

 -> $ docker start [<container_name>|<container_id>]
    # 方法1：重启已停止的容器，恢复容器的状态与自身文件系统。
    # 该命令将继承之前运行容器的选项

 -> $ docker restart [<container_name>|<container_id>]
    # 方法2：重启已停止的容器，恢复容器的状态与自身文件系统。

 -> $ docker logs [<container_name>|<container_id>]
    # 查看运行中或已停止的容器日志信息，常用于容器排错。

 -> $ docker logs --tail <number> -f -t [<container_name>|
<container_id>]
    # 动态跟踪守护式容器的日志信息
    /* EXAMPLE */
    $ docker logs --tail 5 -f -t mysql56

 -> $ docker inspect [<container_name>|<container_id>]
    # 查看运行中或已停止的容器的元数据

 -> $ docker inspect \
      -f '{{ .NetworkSettings.IPAddress }}' [<container_name>|
<container_id>]
```

    # 使用Go语言模板查看运行中或已停止的容器的IP地址

 -> $ docker inspect \
      -f '{{ .HostConfig.LogConfig.Type }}' [<*container_name*>|<*container_id*>]
    # 使用Go语言模板查看运行中或已停止的容器日志驱动类型

```
[godev@cloud-ctl backup]$ docker inspect docker-ce-18.09-docs --format='{{json .HostConfig}}' | jq .
{
  "Binds": null,
  "ContainerIDFile": "",
  "LogConfig": {
    "Type": "json-file",
    "Config": {}
  },
  "NetworkMode": "default",
  "PortBindings": {
    "4000/tcp": [
      {
        "HostIp": "",
        "HostPort": "4000"
      }
    ]
  },
  "RestartPolicy": {
    "Name": "always",
    "MaximumRetryCount": 0
  },
```

    /* EXAMPLE */
    $ docker inspect nova_api --format '{{json .HostConfig}}' | jq .
    $ docker inspect nova_api --format '{{json .HostConfig}}' | python -m json.tool
    # 使用Go语言模板查看运行中或已停止的容器的配置信息
    # 使用**json**参数将以正确的**JSON**格式输出，否则使用**jq**命令或**python**模块将报错。

 -> $ docker stats <*container_name1*> <*container_name2*> ... <*container_nameN*>
    # 查看一个或多个容器的系统资源使用统计信息，也可使用*container_id*。

 -> $ docker port [<*container_name*>|<*container_id*>]
    # 查看容器端口与宿主机的端口映射关系

Docker image管理命令：
 -> $ docker login
    # 使用DockerHub用户名与密码登录DockerHub公共镜像仓库
    # 登录DockerHub的认证信息保存路径：$HOME/.docker/config.json
    # 登录成功后将注入认证信息，退出登录后将清除认证信息。

```
{
    "auths": {    # Docker Hub公共镜像仓库使用v1 restful api
        "https://index.docker.io/v1/": {
            "auth": "YWxiZXJ0aHVhOnl1bHNpYzg5MTIyOCE="
        }
    }
}
```

 -> $ docker logout
    # 退出DockerHub登录，自动清除 $HOME/.docker/config.json 认证信息。

 -> $ docker search <*image_name*>:[*tag*]
    # 查找Docker公共镜像仓库中的相应Docker镜像信息
    # Docker使用RESTful API的方式与Docker镜像仓库进行交互，区分API的版本（v1或v2）。
    # Docker公共镜像仓库使用v1版RESTful API，只能使用docker search命令搜索。
    # **Docker私有镜像仓库使用*v2*版RESTful API，可使用docker-registry-cli工具搜索，**
      该命令为**python**脚本。

-> $ docker-registry-cli <*private_registry_url*>:5000 list all
列出非安全的Docker私有镜像仓库中的所有可用Docker镜像，无需验证本地CA证书。
* 注意：
访问非安全的Docker私有镜像仓库需配置本地Docker配置文件/etc/sysconfig/docker，
添加如下配置：
INSECURE_REGISTRY='... --insecure-registry <*private_registry_url*>:5000 ...'

-> $ docker-registry-cli <*private_registry_url*>:5000 search <*image_name*>:[*tag*]
在非安全的Docker私有镜像仓库中搜索相应的Docker镜像

-> **$ docker-registry-cli <*private_registry_url*>:5000 list all ssl**
/* EXAMPLE */
$ docker-registry-cli workstation.lab.example.com:5000 list all ssl
# 列出安全的Docker私有镜像仓库中所有可用的Docker镜像，需验证本地CA证书。
* 注意：
访问安全的Docker私有镜像仓库需配置本地Docker配置文件/etc/sysconfig/docker，
添加如下配置：
ADD_REGISTRY='... --add-registry <*private_registry_url*>:5000 ...'
/* EXAMPLE */
$ docker-registry-cli workstation.lab.example.com:5000 search php ssl

-> $ docker pull <*image_name*>:[*tag*]
拉取指定的Docker容器镜像至本地Docker缓存中
未指定Docker容器镜像的tag，默认情况下将下载latest最新版的镜像。
* 注意：
1. 私有镜像仓库容器镜像名称格式：
<*private_registry_url*>:5000/<*repository*>/<*image_name*>:[*tag*]
2. 公共镜像仓库容器镜像名称格式：
a. docker.io/<*repository*>/<*image_name*>:[*tag*]
b. registry.access.redhat.com/<*repository*>/<*image_name*>:[*tag*]
c. **quay.io/<*repository*>/<*image_name*>:[*tag*]**
3. 公共镜像仓库分类：
a. 用户仓库（user repository）：由用户管理与维护
b. 顶层仓库（top-level repository）：由Docker公司或各社区管理人员维护
容器镜像名称常为 docker.io/library/<*image_name*>:[*tag*] 或
docker.io/<*image_name*>:[*tag*]，可省略library。
4. 指定容器镜像名称时，尽量使用容器镜像名称的全称。
5. **registry包含repository，repository包含不同版本或标签的相似容器镜像。**
6. 若拉取容器镜像速度缓慢，可配置国内容器镜像加速：
/etc/docker/daemon.json

```
{
  "registry-mirrors": [
    "https://registry.docker-cn.com",
    "http://hub-mirror.c.163.com",
    "https://docker.mirrors.ustc.edu.cn"
  ]
}
```

-> $ docker push <*image_name*>:[*tag*]
   # 上传Docker容器镜像至Docker公共或私有镜像仓库中

-> **$ docker commit **
   **-m '<*commit_message*>' **
   **-a '<*author*>' **
   <*container_id*> <*new_image_name*>:[*tag*]
   # 提交容器中的更新内容以生成新的容器镜像
   * 注意:
   1. 构建自定义容器镜像时，必须先运行容器，进入容器构建配置完成并exit退出后，
      再使用docker commit命令进行构建。
   2. **构建提交的容器镜像只提交与原先容器镜像存在差异的镜像层（layer），加快容器**
      镜像部署，实现轻量级构建。
   3. 该方法构建的容器镜像将包含临时文件与日志文件，不推荐用于构建容器镜像!

-> $ docker build -t <*image_name*>:[*tag*] <*build_context_directory_with_dockerfile*>
   # 根据Dockerfile所在的构建上下文（目录），并指定自定义容器镜像的名称进行构建编译。
   /* EXAMPLE */
   $ docker build -t kube-registry.domain12.example.com:5000/centos74-nginx:1.0 .
   # 根据当前构建上下文（目录）的Dockerfile进行自定义容器镜像构建
   $ docker build -t kube-registry.domain12.example.com:5000/centos74-nginx:1.0 \
     /home/hualf/static_web
   # 根据指定的构建上下文（目录）的Dockerfile进行自定义容器镜像构建
   **$ docker build -t kube-registry.domain12.example.com:5000/centos74-nginx:1.0 **
     git@github.com:hualf/static_web
   # 根据Git仓库中的构建上下文（目录）中的Dockerfile进行自定义容器镜像构建
   * 注意:
   1. 使用Dockerfile构建自定义容器镜像，具有可重复性、透明性与**幂等**性。
   2. 使用Dockerfile构建自定义容器镜像时，必须先创建目录作为构建环境，
      即**构建上下文（build context）**，Dockerfile与用于构建自定义容器镜像的文件与目录都必须位于构建上下文中。
   3. **docker build命令根据Dockerfile的指令将所需的文件与目录向docker**
      守护进程提交，由docker守护进程进行构建。
   4. Dockerfile中的每条指令将提交生成一个镜像层（layer），并返回*image_id*，
      成功生成的镜像层位于构建缓存中。
   5. 若Dockerfile在构建过程中失败，**可运行失败之前的容器镜像**，进入容器

调试完成后，重新进行构建，由于之前构建的镜像层（layer）位于构建缓存中，
再次构建将从失败的这步开始构建。

```
-> $ docker history [<image_name>|<image_id>]
   # 查看容器镜像的构建历史与包含的镜像层（layer）

-> $ docker images
   # 查看本地Docker缓存中的所有Docker容器镜像信息

-> $ docker images -aq
   # 查看本地Docker缓存中的所有Docker容器镜像ID

-> $ docker rmi [<image_name>|<image_id>]
   # 删除本地Docker缓存中指定的Docker容器镜像。
     * 注意：
       1. 同一个Docker容器镜像只有唯一的image_id，但可具有多个不同的image_name，
          即多个不同的tagged镜像名称。
       2. 若直接指定image_id，则将具有多个image_name的Docker容器镜像全部删除。

-> $ docker rmi [<image_name>|<image_id>]
   # 删除本地Docker缓存中指定的Docker容器镜像，也可同时删除多个Docker容器镜像。
     * 注意：
       1. 若该Docker容器镜像具有多个名称，则只是对该容器镜像去除tag（untagged），
          即去除该容器镜像的名称。
       2. 若该Docker容器镜像只具有一个名称，则直接删除该容器镜像。

-> $ docker rmi $(docker images -aq)
   # 批量删除所有本地缓存中的Docker容器镜像

-> $ docker inspect [<image_name>|<image_id>]
   # 查看Docker容器镜像的元数据（详细信息）

-> $ docker load -i <docker_image_file>.[tar|tgz]
   # 加载本地打包的Docker容器镜像文件至本地Docker缓存

-> $ docker save -o <docker_image_file>.[tar|tgz] <image_name>:[tag]
   # 方法1：保存本地Docker镜像缓存至指定打包文件中

-> $ docker save <image_name>:[tag] > <image_name>.[tar|tgz]
   # 方法2：保存本地Docker镜像缓存至指定打包文件中

-> $ docker tag <image_tag>:[tag] <new_image_tag>:[tag]
   # 更改Docker容器镜像名称
   /* EXAMPLE */
   $ docker tag \
     docker.io/wordpress workstation.lab.example.com:5000/wordpress
```

容器报错与排查汇总：

- **CentOS 7.x安装Docker CE 18.09.0报错：**

  1. 问题描述：CentOS 7.2中使用Docker CE 18.09.0运行容器报错

解决方法：将CentOS 7.2升级至CentOS 7.4及更高版本可解决该问题

```
[godev@cloud-ctl ~]$ docker run --name docker-ce-18.09-docs -d --restart always -p 4000:4000 docs/docker.github.io:latest
16698456205b097ddff025fbf6289a38fbe91e6cb488b73c175e1ce482abb0f7
docker: Error response from daemon: OCI runtime create failed: unable to retrieve OCI runtime error (open /run/containerd/io.containerd.runtime.v1.linux/moby/16698
38fbe91e6cb488b73c175e1ce482abb0f7/log.json: no such file or directory): runc did not terminate sucessfully: unknown.
```

2. 问题描述：CentOS 7.x中无法使用Docker CE 18.09.0的overlay2存储驱动
   解决方法：
   a. 由于overlay2存储驱动使用后端xfs文件系统，该文件系统需要支持 *d_type* 特性。
   b. 在默认的 ***kernel 3.x*** 版本中不支持该特性，将默认的kernel版本升级至4.x版本即可
      支持d_type特性。
   c. **$ sudo mkfs.xfs -n ftype=1 <*device_name*>**
      # 使用d_type特性创建xfs文件系统

```
[root@cloud-ctl ~]# docker info | grep -E '^Server|^Kernel'
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
WARNING: overlay2: the backing xfs filesystem is formatted without d_type support, which leads to incorrect behavior.
         Reformat the filesystem with ftype=1 to enable d_type support.
         Running without d_type support will not be supported in future releases.
Server Version: 18.09.0
Kernel Version: 3.10.0-693.11.6.el7.x86_64   # 该kernel版本不支持overlay2特性
[root@cloud-ctl ~]#
# Docker CE 18.09.0使用overlay2存储驱动，该存储驱动的后端文件系统xfs必须支持d_type特性，该特性在kernel 4.x中实现，需升级内核！
```

```
[root@cloud-ctl scripts]# docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 18.09.0
Storage Driver: overlay2
 Backing Filesystem: xfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
```

- 容器镜像删除报错：

  问题描述：使用容器镜像ID删除容器镜像报错
  解决方法：直接使用容器镜像名称强制删除容器镜像

```
[ovnadmin@ovn-node1 ~]$ sudo docker rmi 6ce13de69622
Error response from daemon: conflict: unable to delete 6ce13de69622 (must be forced) - image is referenced in multiple repositories
```