

`docker -d -p 80:80 nginx` -d表示不阻塞shell指令的端口 -p表示指定内外端口映射

`docker ps` 查看正在运行的容器有哪些

`docker exec -it id bash` 进入到dockers的bash容器

`docker commit`指定一个容器

```
$ docker commit 92 m1
sha256:e0f7072b5bfb97f5a420872e988d4e4096a6c85e9819c64fe6e71c1566762260
[nodem1] (local) root@192.168.0.23 ~
```

`docker image` 查看

```
[nodem1] (local) root@192.168.0.23 ~
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
m1                   latest             e0f7072b5bfb       5 seconds ago      109MB
nginx                latest             f68d6e55e065       6 days ago         109MB
[nodem1] (local) root@192.168.0.23 ~
```

运行创建的镜像

```
[nodem1] (local) root@192.168.0.23 ~
$ docker run -d -p 90:80 m1
c5511ced5db42f2c6f0586b387dad759c4c7d026eab595e116be7dd96e665c78
[nodem1] (local) root@192.168.0.23 ~
$
```

`docker file`

```
1 FROM nginx
2 ADD ./ /usr/share/nginx/html/
```

## 2. docker build 建立一个镜像容器

```
[root@localhost ~]# docker build -t m2 .
Sending build context to Docker daemon 1.748MB
Step 1/2 : FROM nginx
--> f68d6e55e065
Step 2/2 : ADD ./ /usr/share/nginx/html/
--> bed7a0925e0e
Successfully built bed7a0925e0e
Successfully tagged m2:latest
```

### 运行容器

```
[root@localhost ~]# docker run -d -p100:80 m2
5f9acf1bfd023a1d44a2c4f7285655ea04cd692b6685007b39775cd91d723d40
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             Status              Ports
```

docker save m2 (镜像) > 1.tar 镜像打包

docker load < 1.tar

docker images

### 配置docker加速器 阿里云加速

```
Person p1 = new Person();
Person p2 = new Person();
Person p3 = new Person();

https://aa25jngu.mirror.aliyuncs.com

centos7.0++

1 vim /etc/docker/daemon.json
2
{
  "registry-mirrors": ["https://aa25jngu.mirror.aliyuncs.com"]
}
3 systemctl daemon-reload
4 systemctl restart docker
```

### docker 原理

(1)docker有着比虚拟机更少的抽象层。由于docker不需要Hypervisor实现硬件资源虚拟化,运行在docker容器上的程序直接使用的都是实际物理机的硬件资源。因此在CPU、内存利用率上docker将会在效率上有明显优势。

(2)docker利用的是宿主机的内核,而不需要Guest OS。因此,当新建一个容器时,docker不需要和虚拟机一样重新加载一个操作系统内核。仍而避免引导、加载操作系统内核这个比较费时费资源的过程,当新建一个虚拟机时,虚拟机软件需要加载Guest OS,这个新建过程是分钟级别的。而docker由于直接利用宿主机的操作系统,则省略了这个过程,因此新建一个docker容器只需要几秒钟。

### 【docker命令】

docker images --digests 显示镜像摘要信息

docker search 镜像

docker -s 30 tomcat

docker rmi -f 强制删除镜像

docker run -it imag\_id 运行容器

退出容器: exit ctrl+p+Q

docker ps -l 上一个容器

docker ps -n 3 上3次运行的容器记录

docker start id 或者name

docker kill id 强制停止容器

docker exec id ls -l 显示容器内的文件夹 对容器内操作

docker run -it cnetos /bin/bash 以交互式运行

docker run -d centos /bin/sh -c "while true ;do echo centos is runing;sleep 2;done"  
容器名

69313e8e7a9d 查看容器运行日志 -t 时间 -f

docker top 69313e8e7a9d 查看docker内运行的进程

docker inspect 查看容器内部的细节

docker -it id /bin/bash 进入正在运行的容器, 并以命令交互 ctrl +p+q 退出 不停止

docker attach id 再次进入容器

docker exec -t id ls -l /tmp 交互式执行docker容器内的命令

docker cp id:path /path 从容器拷贝到宿主机

docker commit -m="提交的描述信息"-a="作者" 容器id 提交副本

docker serach

docker build -f /docker/Dockerfile -t

docker commit -a="zzyy" -m="del tomcat docs" d52498cea537 atguigu/tomcat02:1.2

docker rm -rf \$(docker ps -q)

docker程序再启动 docker start id

add 具有copy和解压缩的功能

docker run -it -d -v /msm:/msm 72.16.1.188/ihi/msm:7.8

docker 镜像保存 docker save spring-boot-docker -o /home/wzh/docker/spring-boot-docker.tar

docker load -i spring-boot-docker.tar 镜像加载

制作镜像：

`docker build -t dlr_img .` 制作镜像 -f 文件路径

`docker run -it -p 9060:9060 --name ef_nfcs ef_nfcs_img`

【容器数据卷】

【安装】

```
docker run -p 12345:3306 --name mysql
-v /zzyyuse/mysql/conf:/etc/mysql/conf.d
-v /zzyyuse/mysql/logs:/logs
-v /zzyyuse/mysql/data:/var/lib/mysql
-e MYSQL_ROOT_PASSWORD=123456
-d mysql:5.6
```

特点：

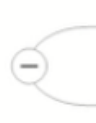
- 1：数据卷可在容器之间共享或重用数据
- 2：卷中的更改可以直接生效
- 3：数据卷中的更改不会包含在镜像的更新中
- 4：数据卷的生命周期一直持续到没有容器使用它为止

<code>attach</code>	Attach to a running container	# 当前 shell 下 attach 连接指定运行镜像
<code>build</code>	Build an image from a Dockerfile	# 通过 Dockerfile 定制镜像
<code>commit</code>	Create a new image from a container changes	# 提交当前容器为新的镜像
<code>cp</code>	Copy files/folders from the containers filesystem to the host path	# 从容器中拷贝指定文件或者目录到宿主机中
<code>create</code>	Create a new container	# 创建一个新的容器，同 run，但不启动容器
<code>diff</code>	Inspect changes on a container's filesystem	# 查看 docker 容器变化
<code>events</code>	Get real time events from the server	# 从 docker 服务获取容器实时事件
<code>exec</code>	Run a command in an existing container	# 在已存在的容器上运行命令
<code>export</code>	Stream the contents of a container as a tar archive	# 导出容器的内容流作为一个 tar 归档文件[对应 import]
<code>history</code>	Show the history of an image	# 展示一个镜像形成历史
<code>images</code>	List images	# 列出系统当前镜像
<code>import</code>	Create a new filesystem image from the contents of a tarball	# 从 tar 包中的内容创建一个新的文件系统映像[对应 export]
<code>info</code>	Display system-wide information	# 显示系统相关信息

push	Push an image or a repository to the docker registry server	# 推送指定镜像或者库镜像至docker源服务器
restart	Restart a running container	# 重启运行的容器
rm	Remove one or more containers	# 移除一个或者多个容器
rmi	Remove one or more images	# 移除一个或多个镜像[无容器使用该镜像才可删除, 否则需删除相关容器才可继续或 -f 强制删除]
run	Run a command in a new container	# 创建一个新的容器并运行一个命令
save	Save an image to a tar archive	# 保存一个镜像为一个 tar 包[对应 load]
search	Search for an image on the Docker Hub	# 在 docker hub 中搜索镜像
start	Start a stopped containers	# 启动容器
stop	Stop a running containers	# 停止容器
tag	Tag an image into a repository	# 给源中镜像打标签
top	Lookup the running processes of a container	# 查看容器中运行的进程信息
unpause	Unpause a paused container	# 取消暂停容器
version	Show the docker version information	# 查看 docker 版本号
wait	Block until a container stops, then print its exit code	# 截取容器停止时的退出状态值

docker -d 容器名 启动交互是容器

一次性删除多个容器

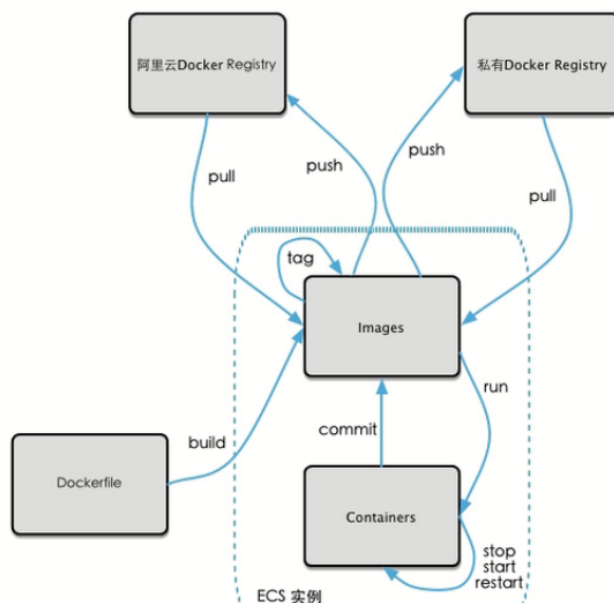

 docker rm -f \$(docker ps -a -q)  
 docker ps -a -q | xargs docker rm

```

7
8
9 docker run -d -p 9080:8080 --name myt9
10 -v /zzyyuse/mydockerfile/tomcat9/test:/usr/local/apache-tomcat-9.0.8/webapps/test
11 -v /zzyyuse/mydockerfile/tomcat9/tomcat9logs:/usr/local/apache-tomcat-9.0.8/logs
12 --privileged=true
13 zzyytomcat9

```

本地镜像上传到阿里云



利用本地运行的docker镜像生成新的镜像 推送到云端

先生成

```

[ root@atguigu data]# docker commit -a zzyy -m "new mycentos1.4 with vim and ifconfig" d2f590e000b2 mycentos:1.4
bd53651fa3463ee48fdcd6ff020fd0ba3b9adccc9b15a82639d06d4389c727ef
[ root@atguigu data]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
mycentos             1.4                bd53651fa346       5 seconds ago      359.5 MB
zzyytomcat9         latest             2ee11d082dfc       About an hour ago   738.3 MB
myip_son            latest             01bb8d8dfce1       2 hours ago        263.5 MB
myip_father         latest             98eca5466940       2 hours ago        261.7 MB
myip2               latest             319dc44d3cf2       2 hours ago        261.7 MB
myip                latest             ea174000d4a8       2 hours ago        261.7 MB
mycentos            1.3                431eb4092d06       3 hours ago        359.5 MB
zzyy/centos         latest             ee2a35672484       6 hours ago        199.7 MB
atguigu/mytomcat    1.2                4db6a44256b2       2 days ago         462.7 MB
mongo               latest             ca96c146aa68       2 days ago         378.4 MB
hello-world         latest             3535063d9957       2 days ago         1.848 kB
tomcat              latest             d94da71c3a1f       9 days ago         462.6 MB
redis               3.2                2b715bf14369       2 weeks ago        75.99 MB
mysql               5.6                a1a19b6fa471       2 weeks ago        256.1 MB
centos              latest             88ec626ba223       5 weeks ago        199.7 MB
[ root@atguigu data]#
  
```

登录阿里云开发者页面



待镜像推送到registry.

```
$ sudo docker login --username= registry.cn-hangzhou.aliyuncs.com
$ sudo docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/zzyybuy/mycentos:[镜像版本号]
$ sudo docker push registry.cn-hangzhou.aliyuncs.com/zzyybuy/mycentos:[镜像版本号]
```

```
[root@atguigu data]# docker images mycentos
REPOSITORY          TAG          IMAGE ID          CREATED          VIRTUAL SIZE
mycentos             1.4          bd53651fa346     4 minutes ago   359.5 MB
mycentos             1.3          431eb4092d06     3 hours ago     359.5 MB
[root@atguigu data]# docker tag bd53651fa346 registry.cn-hangzhou.aliyuncs.com/zzyybuy/mycentos:
```

## 【docker原理】

### docker cs结构 减少硬件资源虚拟化 守护进程

Docker是一个Client-Server结构的系统，Docker守护进程运行在主机上，然后通过Socket连接从客户端访问，守护进程从客户端接受命令并管理运行在主机上的容器。容器，是一个运行时环境，就是我们前面说到的集装箱。

(1)docker有着比虚拟机更少的抽象层。由于docker不需要Hypervisor实现硬件资源虚拟化,运行在docker容器上的程序直接使用的都是实际物理机的硬件资源。因此在CPU、内存利用率上docker将会在效率上有明显优势。

(2)docker利用的是宿主机的内核,而不需要Guest OS。因此,当新建一个容器时,docker不需要和虚拟机一样重新加载一个操作系统内核。仍而避免引导、加载操作系统内核这个比较费时费资源的过程,当新建一个虚拟机时,虚拟机软件需要加载Guest OS,这个新建过程是分钟级别的。而docker由于直接利用宿主机的操作系统,则省略了这个过程,因此新建一个docker容器只需要几秒钟。

docker容器后台运行，前台就必须要有有一个进程

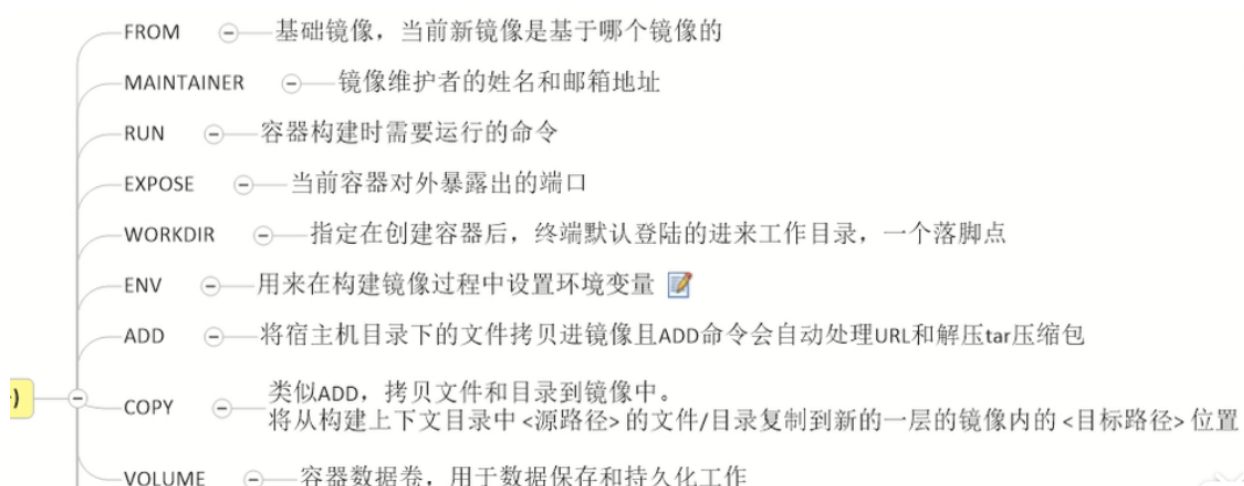
## 【容器数据卷】数据的持久化和数据共享

docker run -it -v /宿主机绝对路径目录: /容器内目录 镜像名 数据同步，修改后数据仍能同步

docker run -it -v /宿主机绝对路径目录: /容器内目录 :ro 镜像名 ro代表只读  
volume 容器数据卷，用于数据保存和持久化工作

编写 dockerfile

## 【容器间传递共享】



<https://www.jianshu.com/p/da1cf9c01c3d> docker harbor

docker 数据库启动

docker

docker run -it -e MYSQL\_ROOT\_PASSWORD='123456' -p 3306:3306

172.16.1.188/database/mysql:8.0

开机启动

vim /usr/lib/systemd/system/redis.service

docker pull 172.16.1.188/database/mysql:5.5

<http://172.16.1.187/SOTP/index.php/DataInit/datainit>

数据库备份: mysqldump -h 172.16.1.187 -u root -p123456 nc4

**【docker数据持久化】**

172.16.1.188/database/mysql:5.5

指定本地的一个文件夹



### 【docker mysql启动】

```
docker run -it -d -p 3306:3306 -v /root/data-service/mysql:/var/lib/mysql  
172.16.1.188/database/mysql:5.5
```

实现mysql数据持久化

### 【dockers-compose】

Docker Compose 是通过python编写的，Docker的服务编排工具，主要用来构建基于Docker的复杂应用，Compose 通过一个配置文件来管理多个Docker容器，非常适合组合使用多个容器进行开发的场景。

<https://blog.51cto.com/9291927/2310444>

docker-compose.yml文件编写

## docker容器之间通信的三种方式

### 1、build

服务除了可以基于指定的镜像，还可以基于一份Dockerfile，在使用up启动时执行构建任务，构建标签是build，可以指定Dockerfile所在文件夹的路径。Compose将会利用Dockerfile自动构建镜像，然后使用镜像启动服务容器。

```
build: /path/to/build/dir
```

也可以是相对路径，只要上下文确定就可以读取到Dockerfile。

```
build: ./dir
```

设定上下文根目录，然后以该目录为准指定Dockerfile。

```
build:
```

```
  context: ../
```

```
  dockerfile: path/of/Dockerfile
```

build都是一个目录，如果要指定Dockerfile文件需要在build标签的子级标签中使用dockerfile标签指定。

如果同时指定image和build两个标签，那么Compose会构建镜像并且把镜像命名为image值指定的名字。

### 2、context

context选项可以是Dockerfile的文件路径，也可以是到链接到git仓库的url，当提供的值是相对路径时，被解析为相对于撰写文件的路径，此目录也是发送到Docker守护进程的

```
context
```

```
build:
```

```
  context: ../dir
```

### 3、dockerfile

使用dockerfile文件来构建，必须指定构建路径

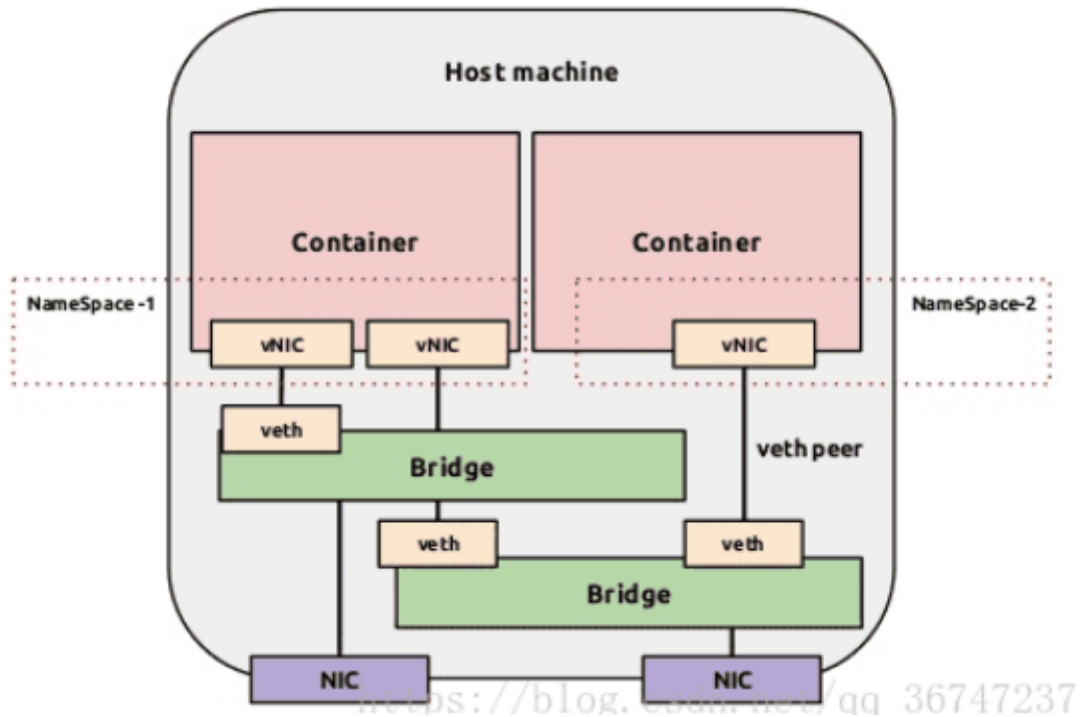
build:

```
context: .
```

```
dockerfile: Dockerfile-alternate
```

dockerfile指令不能跟image同时使用，否则Compose将不确定根据哪个指令来生成最终的服务镜像。

### 4.docker的四种网络方式:



#### 【docker服务重启】

```
systemctl daemon-reload
```

```
systemctl restart docker.service
```

```
--insecure-registry=http://114.67.235.205:8080
```

```
systemctl status firewalld.service
```

**docker tag**

# Docker tag 命令



**docker tag** : 标记本地镜像，将其归入某一仓库。

## 语法

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

## 实例

将镜像ubuntu:15.10标记为 runoob/ubuntu:v3 镜像。

```
root@runoob:~# docker tag ubuntu:15.10 runoob/ubuntu:v3
root@runoob:~# docker images runoob/ubuntu:v3
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
runoob/ubuntu	v3	4e3b13c8a266	3 months ago	136.3 MB

## docker的四种网络模式

host

none

contianiner

查看docker所占空间: docker system df

查看docker中docker ps -aq

删除不再使用的数据卷docker volume rm \$(docker volume ls -q)

host模式，使用--net=host指定。

- container模式，使用--net=container:NAME\_or\_ID指定。
- none模式，使用--net=none指定。
- bridge模式，使用--net=bridge指定，默认设置。

apt-get install build-essential

gcc

install "gcc-c++.x86\_64" -y

[https://blog.csdn.net/zhaojianting/article/details/81095120?utm\\_medium=distribute.pc\\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth\\_1-utm\\_source=distribute.pc\\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase](https://blog.csdn.net/zhaojianting/article/details/81095120?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase)

docker启动镜像远程访问: <https://www.jianshu.com/p/b5f8e79a7c4d>

docker 镜像导入导出有两种方法:

一种是使用 save 和 load 命令

使用例子如下:

<

```
docker load<ubuntu.tar
```

一种是使用 export 和 import 命令

使用例子如下:

```
docker export 98ca36> ubuntu.tar
```

```
cat ubuntu.tar | sudo docker import - ubuntu:import
```

## (7) expose

暴露端口, 只将端口暴露给连接的服务, 而不暴露给宿主机。示例

## (10) links

连接到其他服务的容器。可以指定服务名称和服务别名 ( SERVICE:ALIAS ), 也可只指定服务名称。例如:

【docker-compose开机启动】

需要设置docker开机自启动: `systemctl enable docker`

# docker-compose开机自启动两种方式

## 第一种方式

主要步骤如下:

- (1) 创建docker-compose软连接: `cd /usr/local/bin && ln -s /usr/bin/docker-compose docker-compose`
- (2) 编辑docker-compose自启动脚本: `cd /etc/rc.d/init.d/ && vim start-docker-compose.sh`添加如下内容:
  1. `#!/bin/bash`
  2. `# chkconfig: 2345 85 15`
  3. `# description: docker-compose init start`
  - 4.

```
5. /usr/local/bin/docker-compose -f /data/product/deploy_nginx/docker-  
compose.yml up -d
```

保存退出!

说明△: -f参数后面是docker-compose.yml文件存放的路径; # chkconfig: 2345 85 15  
也可以是# chkconfig: 2345 80 90

- (3) 赋予执行权限: `chmod +x ./start-docker-compose.sh`

## 第二种方式

在vim /etc/rc.d/rc.local添加:

```
/usr/local/bin/docker-compose -f /data/product/deploy_nginx/docker-compose.yml  
up -d
```

转载于:<https://blog.51cto.com/wutengfei/2357985>

# docker如何将运行中的容器保存为 docker镜像?

答: 使用docker commit和docker save保存镜像

```
$ sudo docker commit <当前运行的container id> <仓库名称>:<tag>
```

例子: `docker commit -a "runoob.com" -m "my apache" a404c6c174a2 mymysql:v1`

```
$ sudo docker save -o <仓库名称>-<tag>.img <仓库名称>:<tag>
```

【删除已经退出的容器】

```
docker rm $(sudo docker ps -qf status=exited)
```

#删除所有未运行的容器 (已经运行的删除不了, 未运行的就一起被删除了)

```
sudo docker rm $(sudo docker ps -a -q)
```