**Docker入门与应用**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 日期 | 作者 | 复核者 | 版本 | 更新 |
| 2022/03/30 | 李怀明 | 定制交付部 | V 0.1 | 作成 |
| 2022/03/31 | 李怀明 | 定制交付部 | V 0.2 | 新增附录一章 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[一、 Docker概述 3](#_Toc99613218)

[1.1 什么是Docker 3](#_Toc99613219)

[1.2 Docker的应用场景 4](#_Toc99613220)

[1.3 Docker的优点 4](#_Toc99613221)

[二、 Docker 基础 4](#_Toc99613222)

[2.1 Docker的三大概念 4](#_Toc99613223)

[2.2 为什么容器技术会出现？ 5](#_Toc99613224)

[2.3 Docker架构图 8](#_Toc99613225)

[2.4 Docker安装 8](#_Toc99613226)

[2.5 Docker仓库 12](#_Toc99613227)

[三、 镜像的创建和管理 16](#_Toc99613228)

[3.1 拉取镜像 16](#_Toc99613229)

[3.2 搜索镜像 19](#_Toc99613230)

[3.3 删除镜像 20](#_Toc99613231)

[3.4 创建镜像 21](#_Toc99613232)

[3.5 上传镜像 26](#_Toc99613233)

[四、 容器的管理 27](#_Toc99613234)

[4.1 容器的创建 27](#_Toc99613235)

[4.2 容器的启动 28](#_Toc99613236)

[4.3 容器的交互 29](#_Toc99613237)

[4.4 容器的删除 30](#_Toc99613238)

[4.5 容器的导入和导出 30](#_Toc99613239)

[五、 Dockerfile详解 31](#_Toc99613240)

[5.1 基础镜像的选择 31](#_Toc99613241)

[5.2 Dockerfile中常用命令 32](#_Toc99613242)

[5.3 Dockerfile的使用技巧 34](#_Toc99613243)

[六、 当前1.2.0系统中的镜像和容器 37](#_Toc99613244)

[6.1 系统中使用的镜像 37](#_Toc99613245)

[6.2 系统中正在运行中的容器 39](#_Toc99613246)

[七、 附录：常用命令及参数 43](#_Toc99613247)

[7.1 run 43](#_Toc99613248)

[7.2 start-stop-restart 44](#_Toc99613249)

[7.3 rm 44](#_Toc99613250)

[7.4 exec 44](#_Toc99613251)

[7.5 ps 45](#_Toc99613252)

[7.6 inspect 47](#_Toc99613253)

[7.7 logs 47](#_Toc99613254)

[7.8 top 48](#_Toc99613255)

[7.9 port 48](#_Toc99613256)

[7.10 login 49](#_Toc99613257)

[7.11 pull 49](#_Toc99613258)

[7.12 push 49](#_Toc99613259)

[7.13 search 50](#_Toc99613260)

[7.14 image 50](#_Toc99613261)

[7.15 rmi 51](#_Toc99613262)

[7.16 tag 52](#_Toc99613263)

[7.17 build 52](#_Toc99613264)

[7.18 history 53](#_Toc99613265)

[7.19 save 53](#_Toc99613266)

[7.20 load 53](#_Toc99613267)

[7.21 import 54](#_Toc99613268)

# Docker概述

## 1.1 什么是Docker

Docker 是一个开源的应用容器引擎，基于[Go语言]并遵从 Apache2.0 协议开源。

Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。

容器是完全使用沙箱机制，相互之间不会有任何接口（类似 iPhone 的 app）,更重要的是容器性能开销极低。

Docker 从 17.03 版本之后分为 CE（Community Edition: 社区版） 和 EE（Enterprise Edition: 企业版），我们用社区版就可以了。

## 1.2 Docker的应用场景

Web 应用的自动化打包和发布。

自动化测试和持续集成、发布。

在服务型环境中部署和调整数据库或其他的后台应用。

## 1.3 Docker的优点

Docker 是一个用于开发，交付和运行应用程序的开放平台。Docker 使您能够将应用程序与基础架构分开，从而可以快速交付软件。借助 Docker，您可以与管理应用程序相同的方式来管理基础架构。通过利用 Docker 的方法来快速交付，测试和部署代码，您可以大大减少编写代码和在生产环境中运行代码之间的延迟。

* 快速，一致地交付您的应用程序

Docker 允许开发人员使用您提供的应用程序或服务的本地容器在标准化环境中工作，从而简化了开发的生命周期。

容器非常适合持续集成和持续交付（CI / CD）工作流程，请考虑以下示例方案：

您的开发人员在本地编写代码，并使用 Docker 容器与同事共享他们的工作。

他们使用 Docker 将其应用程序推送到测试环境中，并执行自动或手动测试。

当开发人员发现错误时，他们可以在开发环境中对其进行修复，然后将其重新部署到测试环境中，以进行测试和验证。

测试完成后，将修补程序推送给生产环境，就像将更新的镜像推送到生产环境一样简单。

* 响应式部署和扩展

Docker 是基于容器的平台，允许高度可移植的工作负载。Docker 容器可以在开发人员的本机上，数据中心的物理或虚拟机上，云服务上或混合环境中运行。

Docker 的可移植性和轻量级的特性，还可以使您轻松地完成动态管理的工作负担，并根据业务需求指示，实时扩展或拆除应用程序和服务。

* 在同一硬件上运行更多工作负载

Docker 轻巧快速。它为基于虚拟机管理程序的虚拟机提供了可行、经济、高效的替代方案，因此您可以利用更多的计算能力来实现业务目标。Docker 非常适合于高密度环境以及中小型部署，而您可以用更少的资源做更多的事情。

# Docker 基础

## 2.1 Docker的三大概念

镜像（Image）、容器（Container）、仓库(Repository)

**镜像**

Docker镜像类似于虚拟机镜像，可以将它理解为一个只读的模板。

镜像是创建Docker容器的基础，有了镜像，就可以用这个镜像创建容器。

**容器**

Docker容器类似于一个轻量级的沙箱，Docker利用容器来运行和隔离应用。

容器是从镜像创建的应用运行实例。它可以启动、开始、停止、删除，而这些容器都是彼此相互隔离、互不可见的。

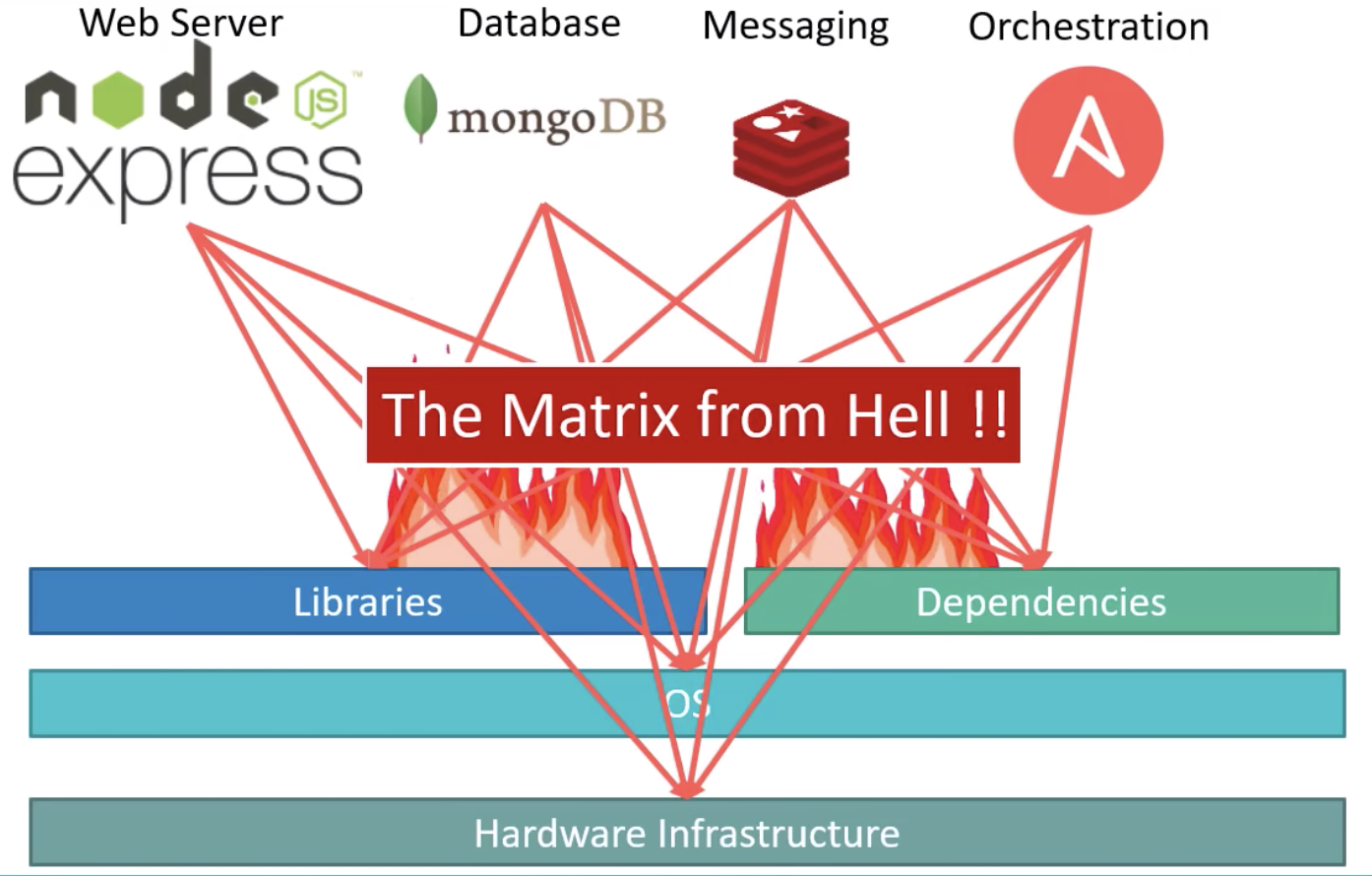
可以把容器看作一个简易版的Linux系统环境（包括root用户权限、进程空间、用户空间和网络空间等）以及运行在其中的应用程序打包而成的盒子。

**仓库**

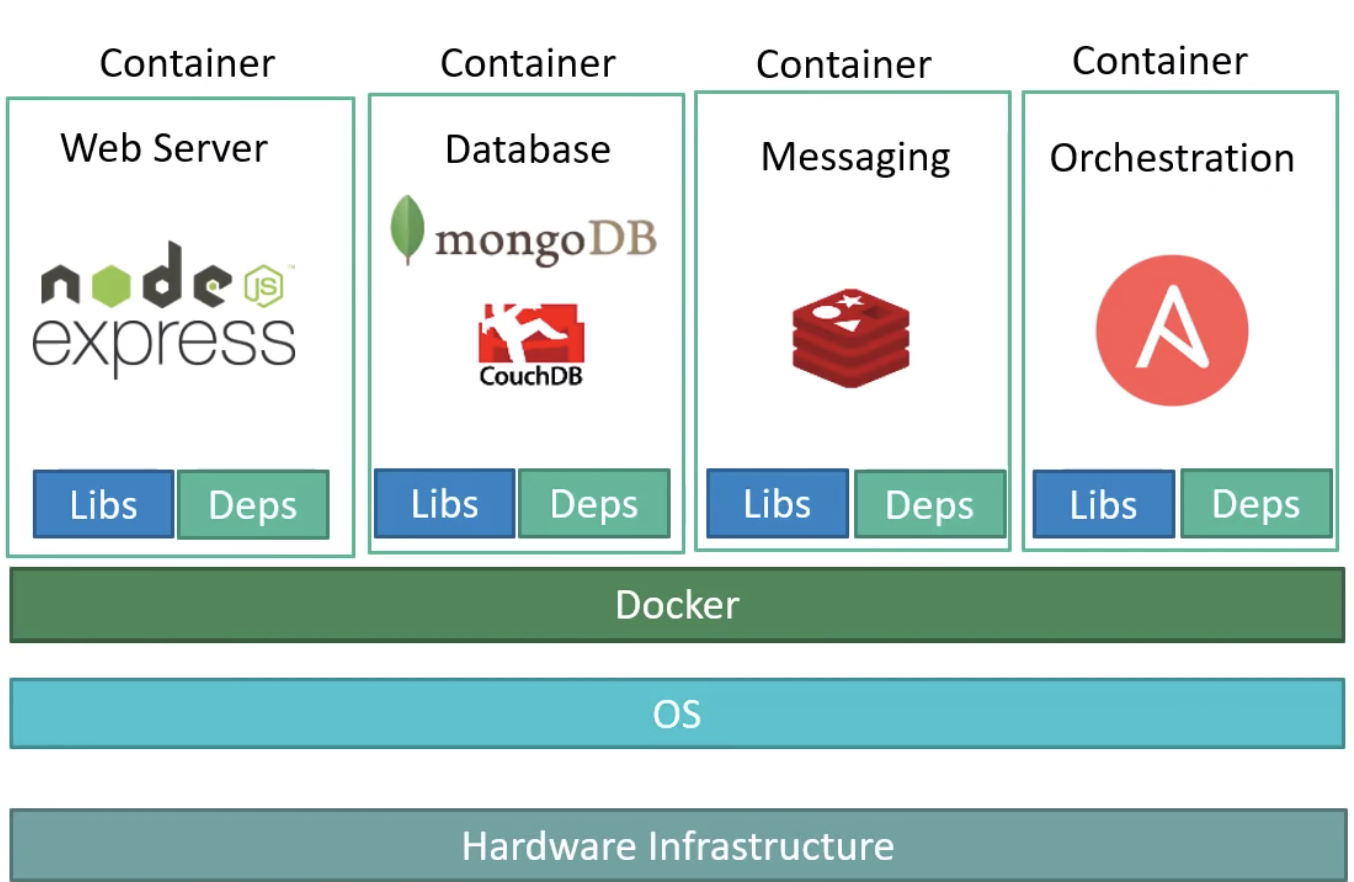
Docker仓库类似于代码仓库，是Docker集中存放镜像文件的场所。

## 2.2 为什么容器技术会出现？

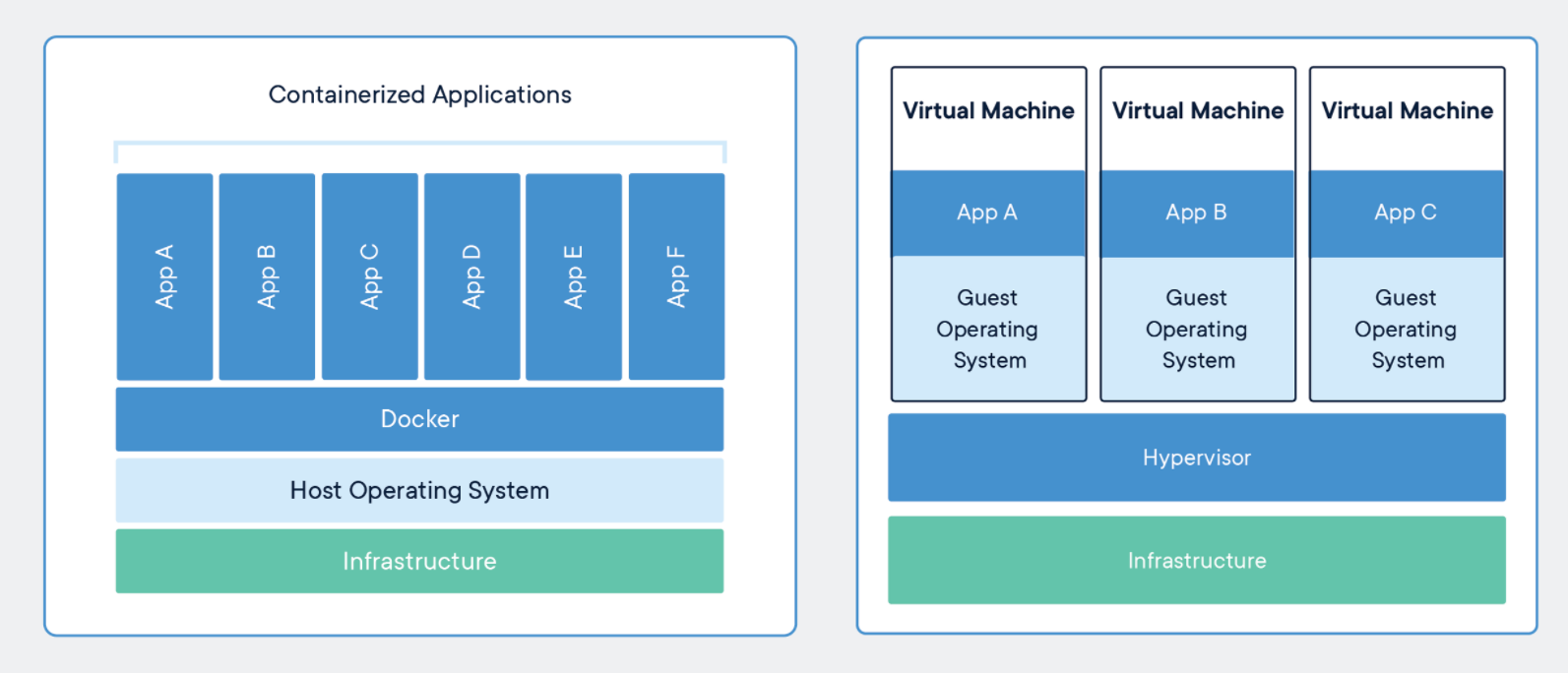
容器技术出现之前

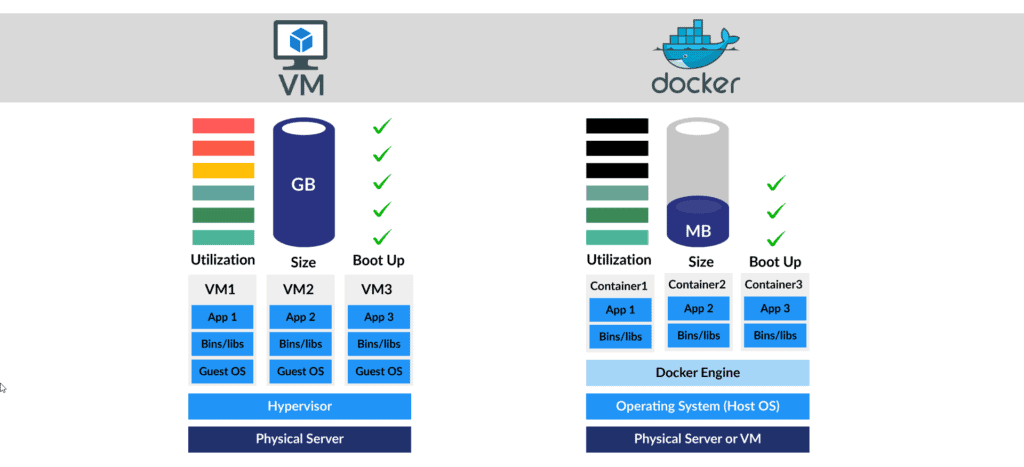


容器技术出现之后



容器 vs 虚拟机





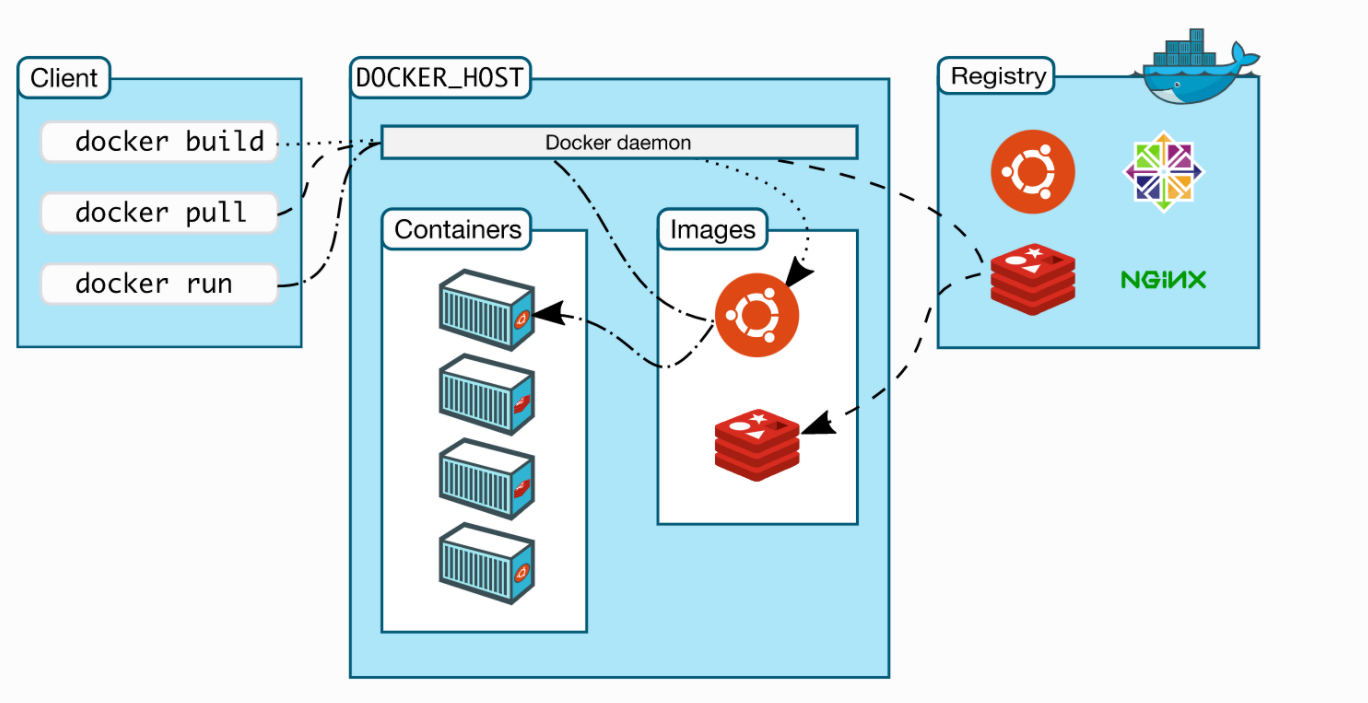
Linux Container容器技术的诞生于2008年（Docker诞生于2013年），解决了IT世界里“集装箱运输”的问题。Linux Container（简称LXC）它是一种内核轻量级的操作系统层虚拟化技术。Linux Container主要由Namespace  和Cgroups  两大机制来保证实现

Namespace命名空间主要用于资源的隔离（诞生于2002年）

Cgroups(Control Groups)就负责资源管理控制作用，比如进程组使用CPU/MEM的限制，进程组的优先级控制，进程组的挂起和恢复等等。（由Google贡献，2008年合并到了Linux Kernel）

|  |  |  |
| --- | --- | --- |
| 容器和虚拟机对比 | 容器 | 虚拟机 |
| 开机时间 | 几秒钟 | 几分钟 |
| 运行 | 利用一个或数个可执行的引擎 | 利用虚拟机的监控程序 |
| 内存效率 | 虚拟化不需要空间，需要内存很少 | 启动前需要运行整个OS，效率较低 |
| 独立性 | 容易受到不利影响，因为没有隔离系统的规定 | 由于有效的隔离机制，干扰可能性最小 |
| 部署 | 部署很容易，因为只有一个映像，可以在所有平台上使用 | 部署相对较长，因为独立的实例负责执行 |
| 性能 | 有限制地性能 | 本机原生性能限制 |

## 2.3 Docker架构图



## 2.4 Docker安装

一键安装Docker

$ curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun

Dcloud安装

$ curl -sSL https://get.daocloud.io/docker | sh

**Ubuntu搭建Docker**

Docker目前支持的最低Ubuntu版本为14.04 LTS，但实际上从稳定性上考虑，推荐使用16.04 LTS或18.0.4 LTS版本，或者20.04LTS版本，并且系统内核越新越好，以支持Docker最新的特性。

官方文档:  
<https://docs.docker.com/engine/install/ubuntu/>

更换源(可选,如果更换为清华源，则后面的安装步骤参数需要指定为清华源)

* 清华大学ubuntu源:
* <https://mirror.tuna.tsinghua.edu.cn/ubuntu>
* 官方文档:
* <https://mirror.tuna.tsinghua.edu.cn/help/ubuntu/>
* vim /etc/apt/sources.list替换源
* 更新包索引
* sudo apt-get update

手动安装docker

移除系统中可能存在的旧版本的Docker

$ sudo apt-get remove docker docker-engine docker.io containerd runc

安装可以通过https使用仓库的一些相关包

$ sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common

添加GPG key  
清华源:

$ curl -fsSL https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/ubuntu/gpg | sudo apt-key add -

ubuntu源：

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

验证指纹

sudo apt-key fingerprint 0EBFCD88

应该得到如下内容:

pub rsa4096 2017-02-22 [SCEA]  
9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88  
uid [ unknown] Docker Release (CE deb)[docker@docker.com](http://mailto:docker@docker.com/)  
sub rsa4096 2017-02-22 [S]

添加docker-ce的仓库

清华源：

$ sudo add-apt-repository \

"deb [arch=amd64] https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/ubuntu \

$(lsb\_release -cs) stable"

ubuntu源:

$ echo \

"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

安装docker

**安装前需要再次更新源:**

$ sudo apt update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io

**更换docker镜像源:(可选)**

$ sudo vim /etc/docker/daemon.json

{

"registry-mirrors": ["http://hub-mirror.c.163.com"]

}

启动docker

sudo systemctl start docker

**Centos搭建Docker**

Docker目前支持CentOS 7及以后的版本。

CentOS更换源方法:

更换为网易源:

$ mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.bakup

$ wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.163.com/.help/CentOS7-Base-163.repo

$ yum clean all

$ yum makecache

手动安装docker

$ sudo yum update

$ sudo yum install -y yum-utils \

device-mapper-persistent-data \

lvm2

$ sudo yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo

$ sudo yum update

$ sudo yum install -y docker-ce

启动Docker服务：

$ sudo systemctl docker start

配置Docker用户

为了避免每次使用Docker命令时都需要切换到特权身份，可以将当前用户加入安装中自动创建的docker用户组

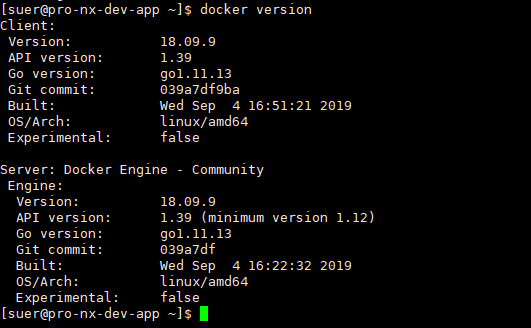
$ sudo usermod -aG docker $USER

更新docker组

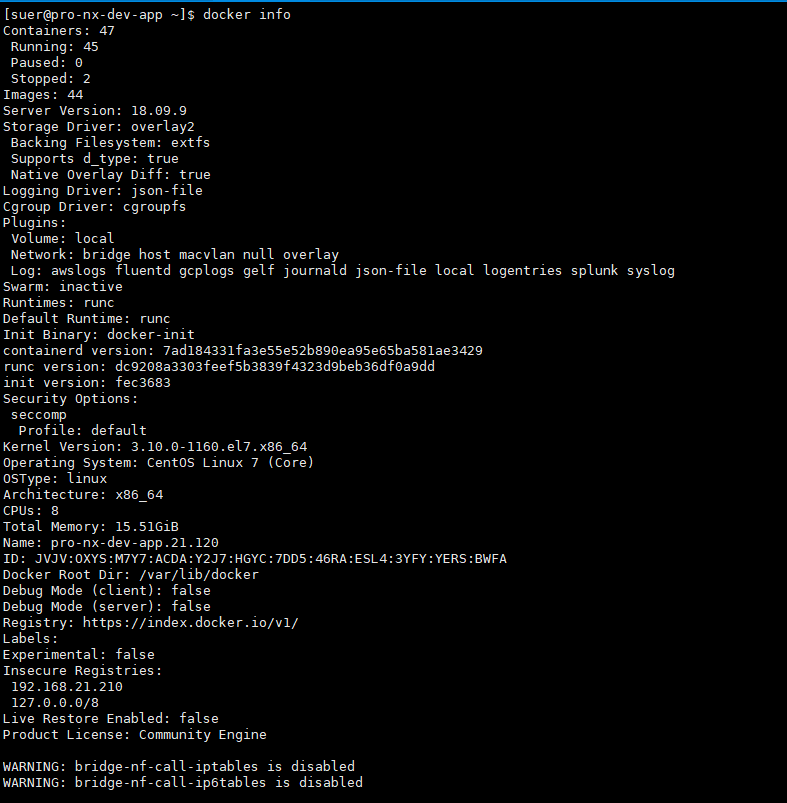
$ newgrp docker

安装好之后，可以查看docker版本等具体信息：

docker version



docker info

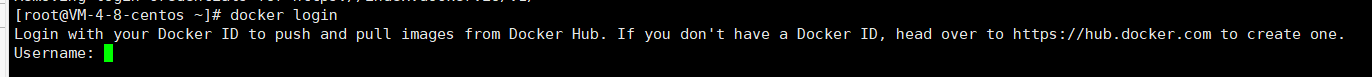


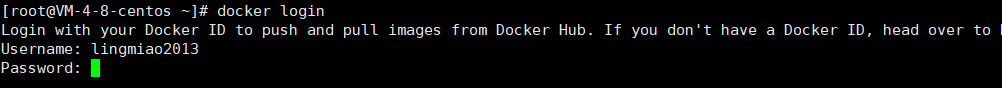
## 2.5 Docker仓库

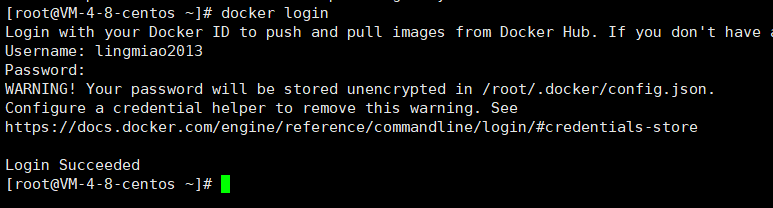
**DockerHub为docker的官方仓库**.  
其网址为：[https://hub.docker.com](https://hub.docker.com/).

在其官网上注册一个帐号后，可以使用命令登录和退出：  
登录：

docker login

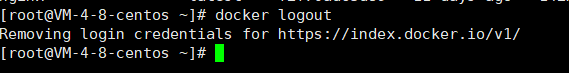






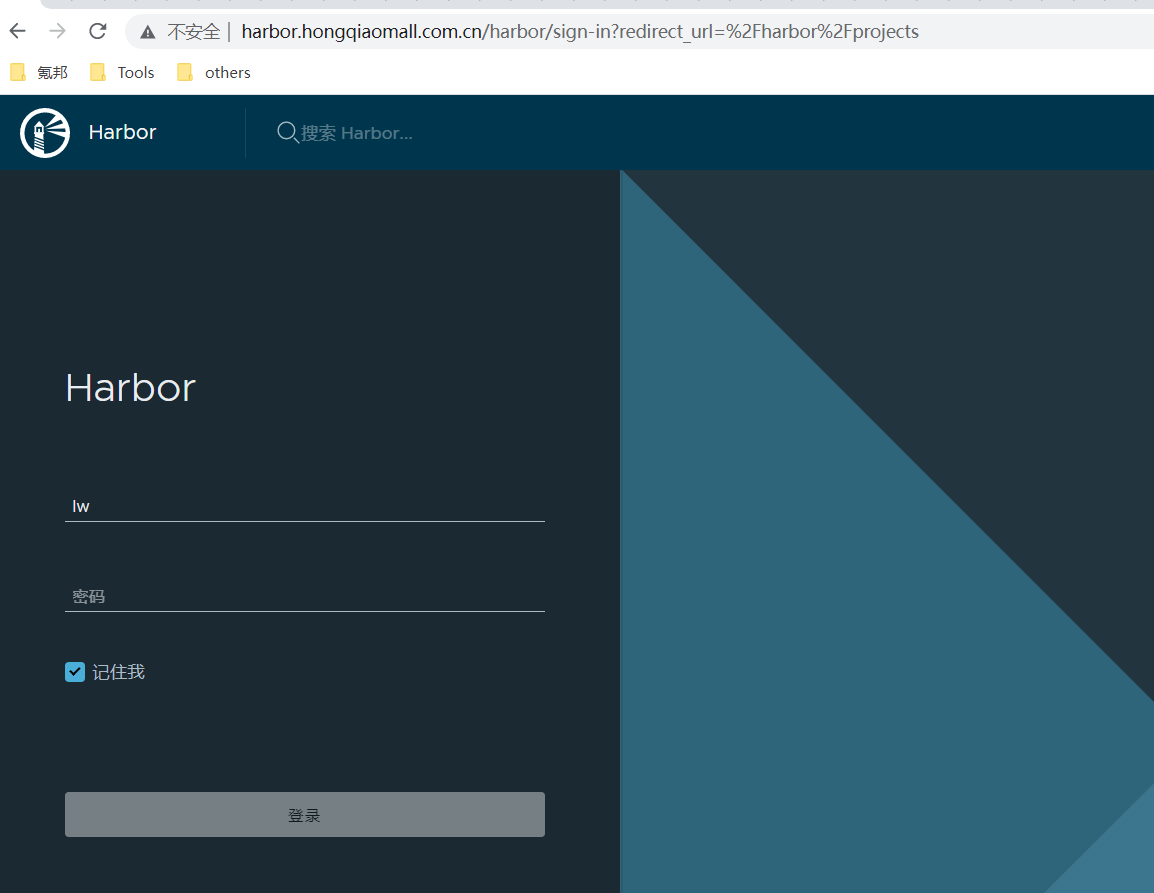
退出：

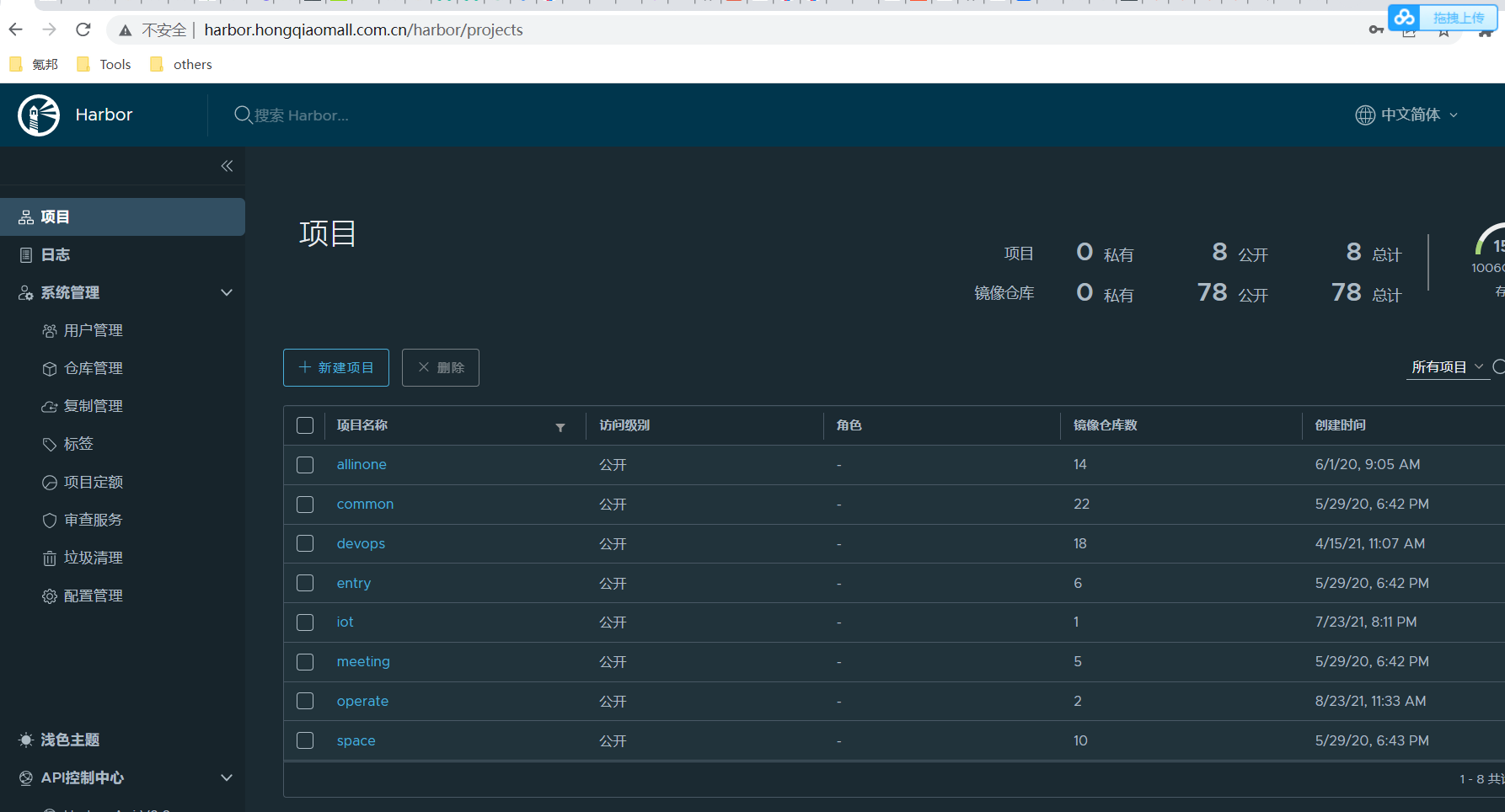
docker logout



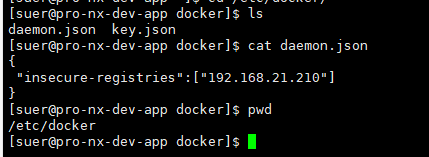
**公司内部docker harbor**

<http://harbor.hongqiaomall.com.cn/> (账号：lw/Kb123456)

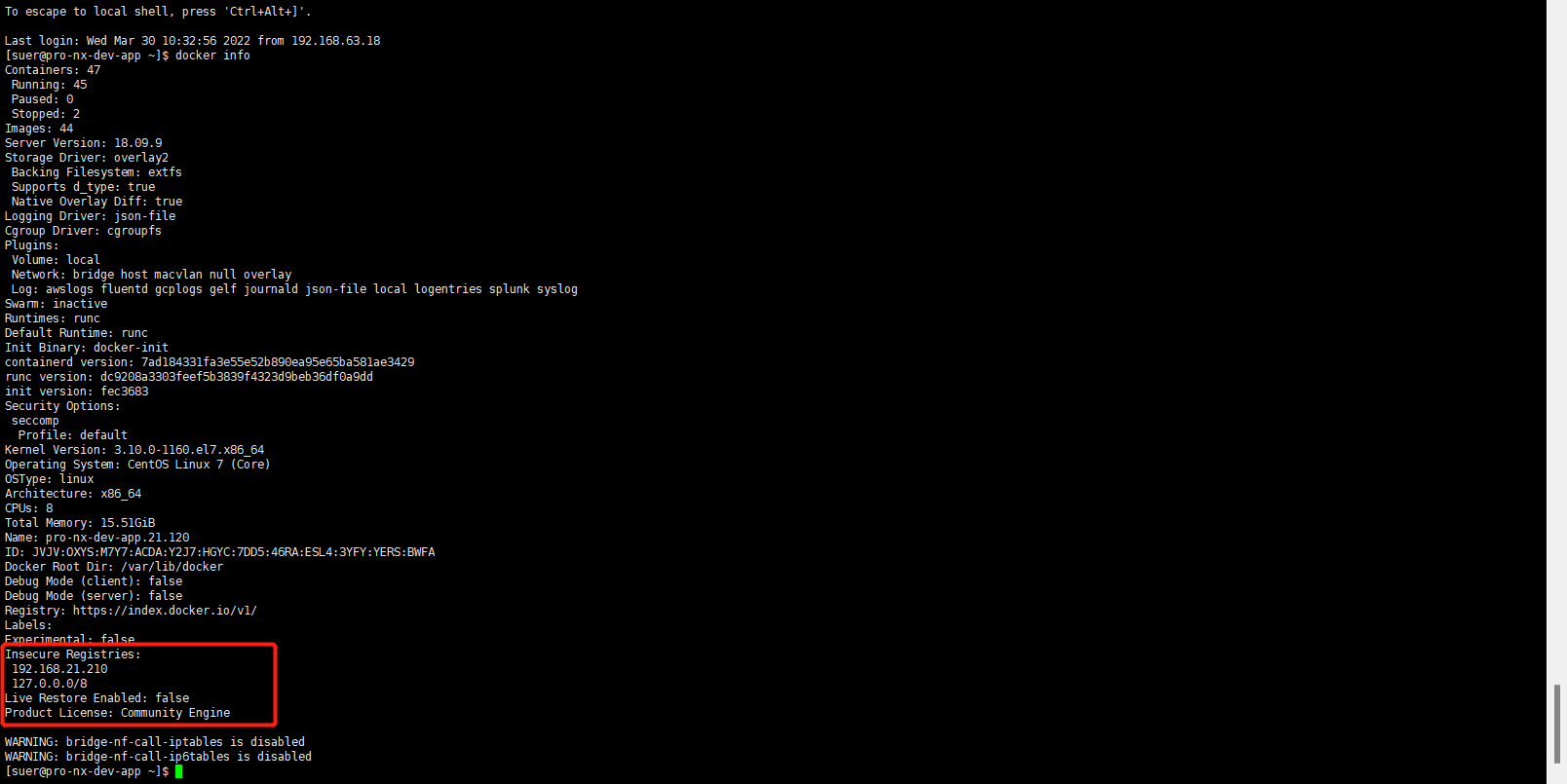




将私有库IP：192.168.21.210添加到配置文件daemon.json中

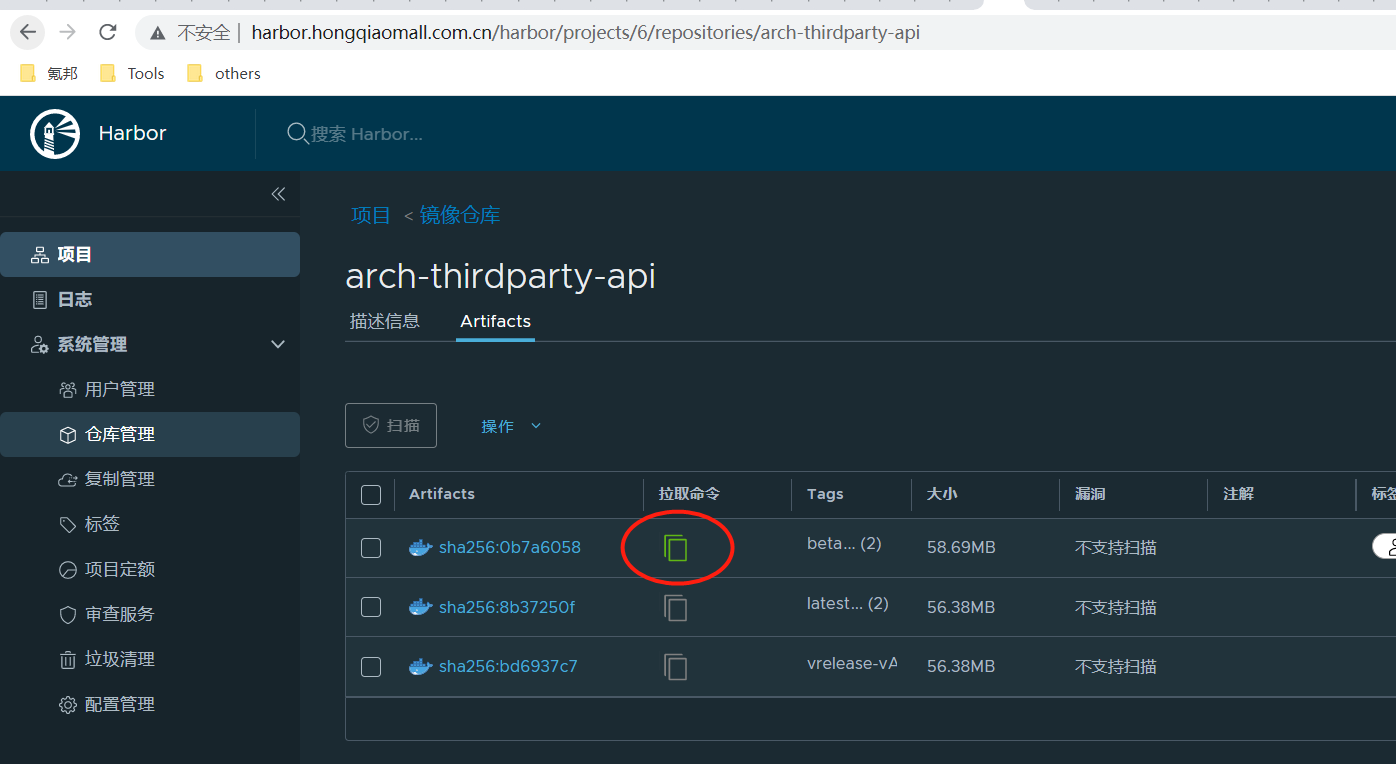


运行docker info查看详情：



镜像拉取命令：

|  |
| --- |
| docker pull 192.168.21.210/allinone/arch-thirdparty-api@sha256:0b7a6058bdff08db4f1fe0bec68e43dbafa0add4f53b2636f1ae7d688c32bd0d |



# 镜像的创建和管理

Docker镜像的常用操作

Docker运行容器前需要本地存在对应的镜像，如果镜像不存在，Docker会尝试先从默认镜像仓库下载（默认使用Docker Hub公共注册服务器中的仓库），用户也可以通过配置，使用自定义的镜像仓库。

关于镜像主要有下面几种常用操作：

* 拉取镜像
* 搜索镜像
* 删除镜像
* 创建镜像
* 上传镜像

## 拉取镜像

拉取ubuntu指定版本： 格式: docker pull name:tag

$ docker pull ubuntu:18.04

不指定版本：  
如果不显式指定TAG，则默认会选择latest标签，这会下载仓库中最新版本的镜像。

$ docker pull ubuntu

##### docker pull时，那些hash串是什么？

镜像文件一般由若干层（layer）组成，6c953ac5d795这样的串是层的唯一id

##### docker pull的完整命令

* docker pull ubuntu:18.04命令相当于docker pull[registry.hub.docker.com/ubuntu:18.04](http://registry.hub.docker.com/ubuntu:18.04)  
  即从默认的注册服务器Docker Hub Registry中的ubuntu仓库来下载标记为18.04的镜像
* 如果从非官方的仓库下载，则需要在仓库名称前指定完整的仓库地址。例如从网易蜂巢的镜像源来下载ubuntu:18.04镜像

$ docker pull hub.c.163.com/public/ubuntu:18.04

##### 全局更换docker仓库源：

sudo vim /etc/docker/daemon.json

{

"registry-mirrors": ["http://hub-mirror.c.163.com"]

}

**关于网络的一些问题:**

1. 虚拟机连接外网不稳定,重启网卡命令:

$ sudo /etc/init.d/networking force-reload

1. 拉某些镜像时报错:  
   tcp: lookup[production.cloudflare.docker.com](http://production.cloudflare.docker.com/): Temporary failure in name resolution

$ vim /etc/resolv.conf

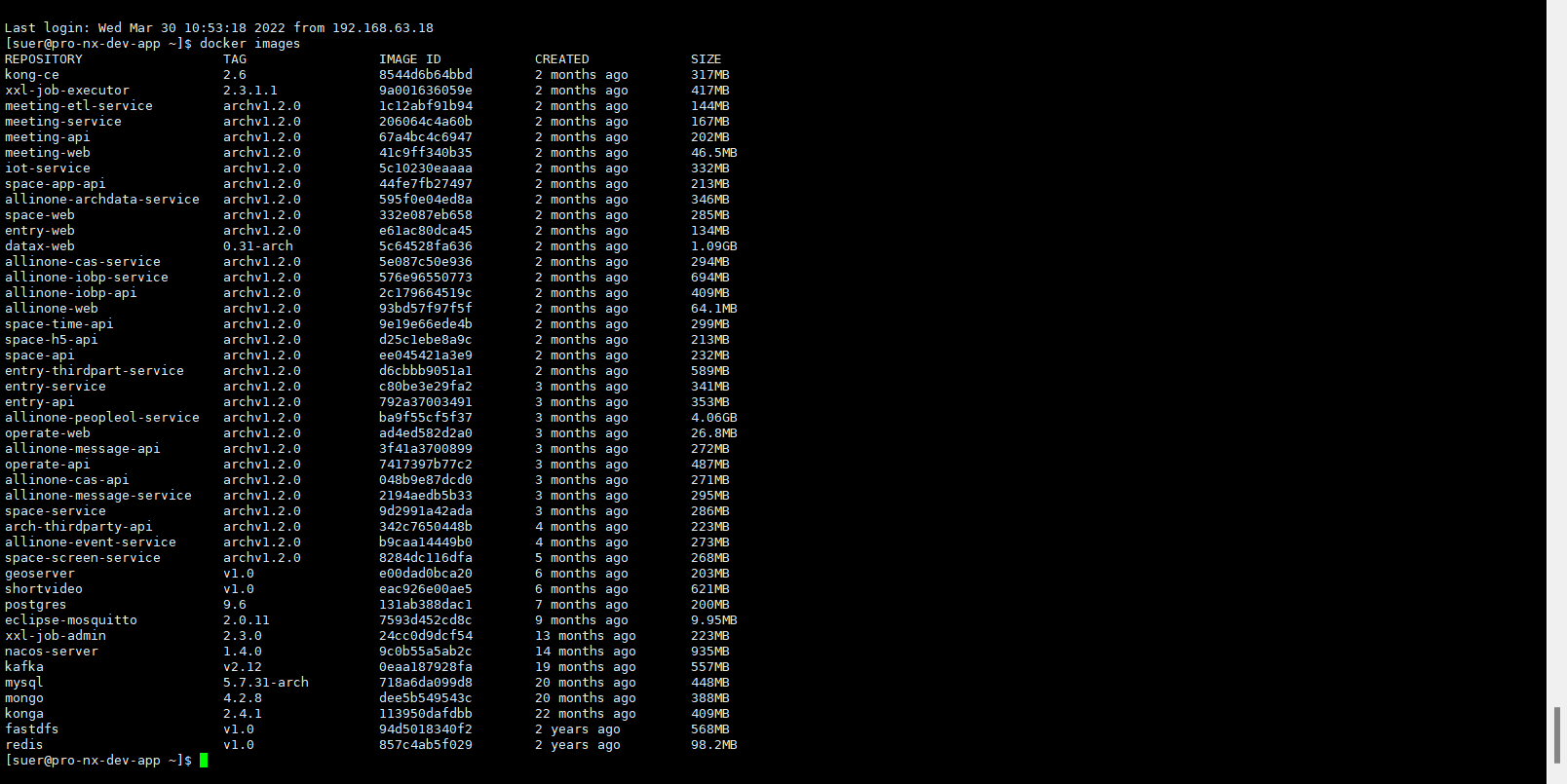
nameserver 8.8.8.8

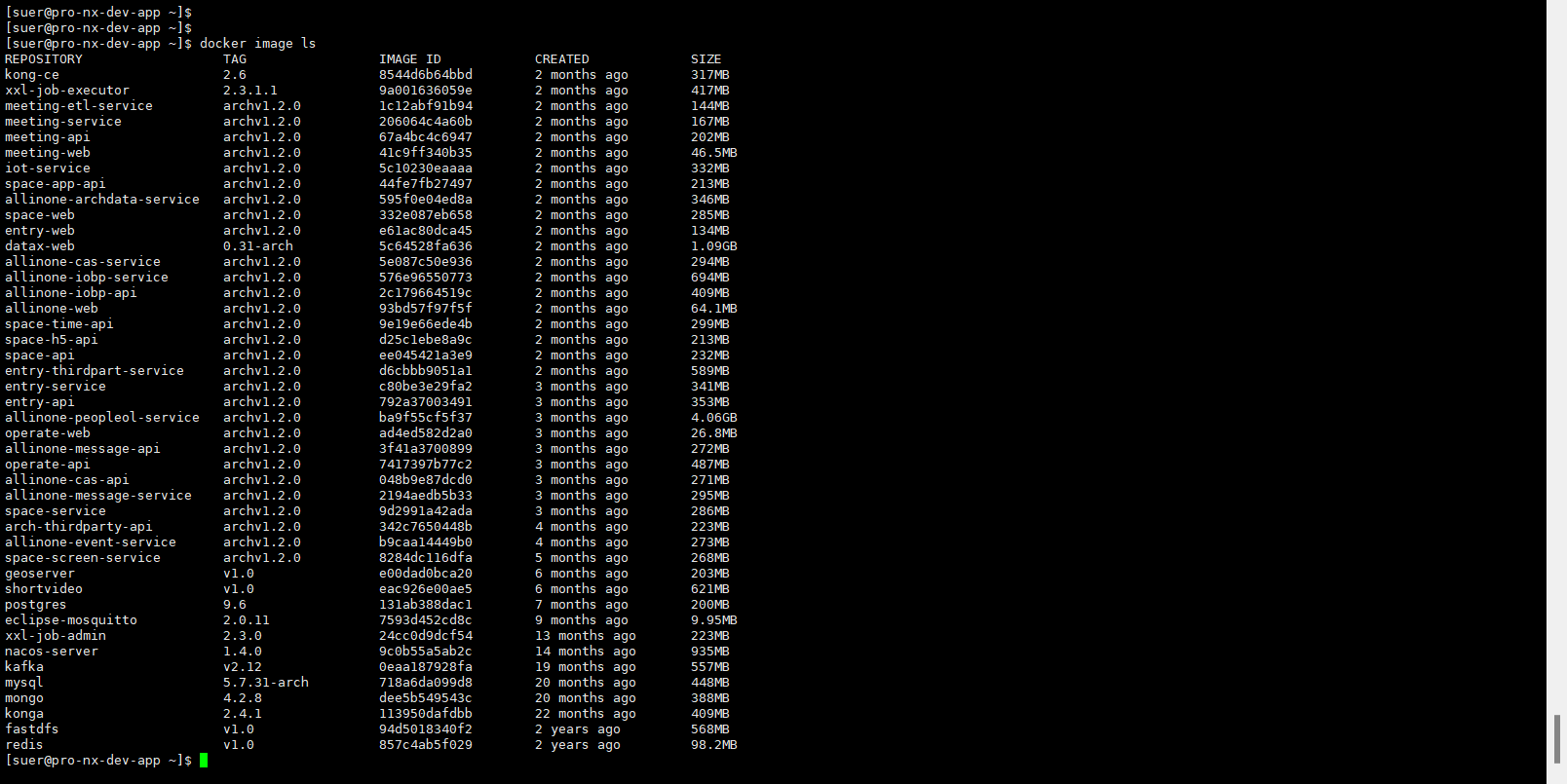
查看镜像信息：  
使用docker images或docker image ls命令可以列出本地主机上已有镜像的基本信息。

$ docker images

在列出信息中，可以看到几个字段信息：

* 来自于哪个仓库，比如ubuntu表示ubuntu系列的基础镜像；
* 镜像的标签信息，比如18.04、latest表示不同的版本信息。标签只是标记，并不能标识镜像内容；
* 镜像的ID（唯一标识镜像），如果两个镜像的ID相同，说明它们实际上指向了同一个镜像，只是具有不同标签名称而已；
* 创建时间，说明镜像最后的更新时间；
* 镜像大小。

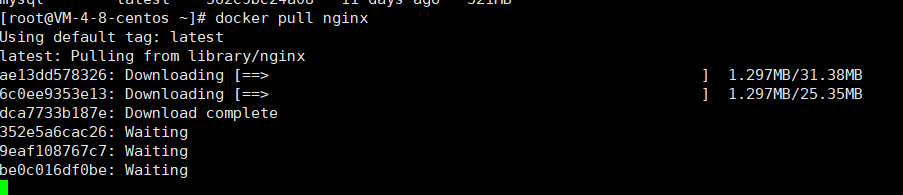


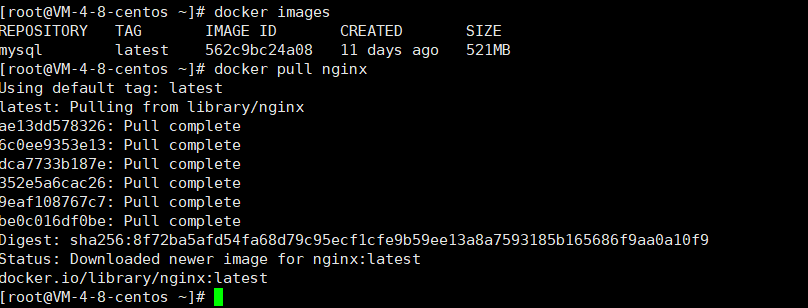


拉取实操：

使用docker pull 拉取镜像到本地：

docker pull nginx





## 搜索镜像

docker search [option] keyword

$ docker search centos

一种是类似于centos这样的基础镜像，也称为根镜像。这些镜像是由Docker公司创建、验证、支持、提供，这样的镜像往往使用单个单词作为名字；

另一种类型的镜像，比如ansible/centos7-ansible镜像，是由Docker用户ansible创建并维护的，带有用户名称为前缀，表明是某用户下的某仓库。可以通过用户名称前缀“user\_name/镜像名”来指定使用某个用户提供的镜像

$ docker search nginx

搜索官方提供的带nginx关键字的镜像:

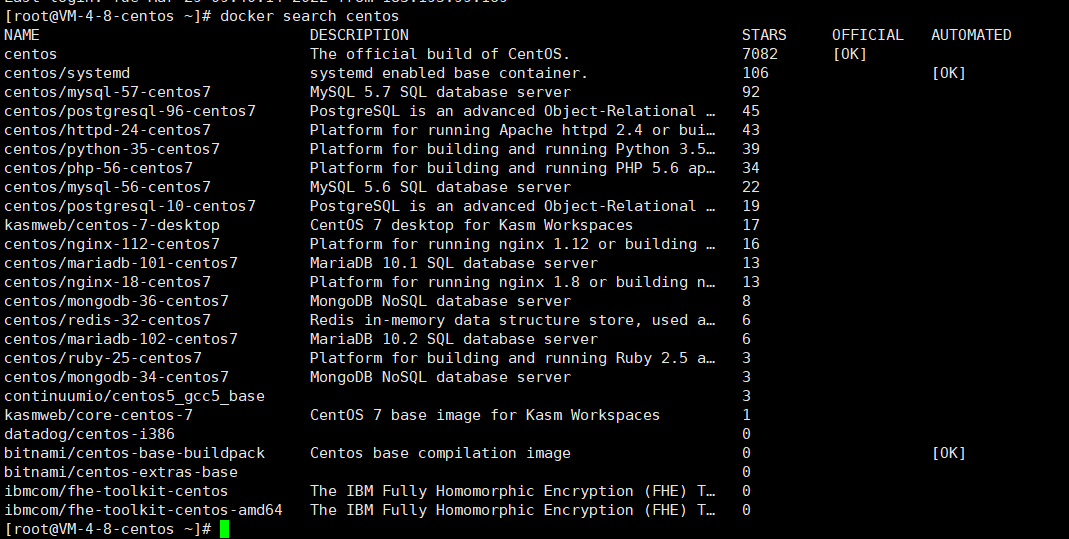
$ docker search --filter=is-official=true ngin

搜索所有star数超过100的nginx镜像:

$ docker search --filter=stars=100 nginx

搜索实操：

使用docker search命令，查找镜像：  
如，查找centos镜像：



在查找结果中，标有OFFICIAL列的为官方维护的镜像

## 删除镜像

使用标签删除镜像  
使用docker rmi或docker image rm命令可以删除镜像  
命令格式：  
docker rmi IMAGE[IMAGE...]，其中IMAGE可以为标签或ID。

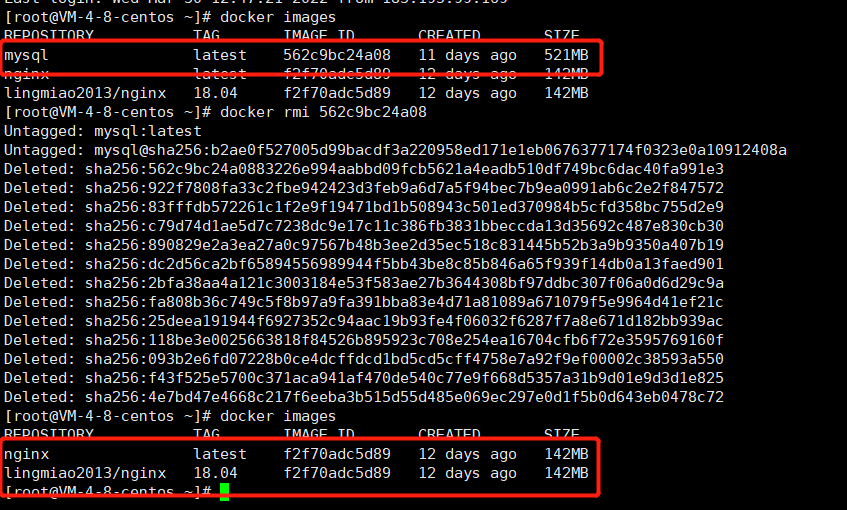
* 使用标签删除镜像

$ docker rmi ubuntu:list

* 使用镜像ID来删除镜像

$ docker rmi 329ed837d508

删除实操：



## 创建镜像

##### 基于已有容器创建

命令格式为docker [container] commit [OPTIONS] CONTAINER [REPOSITORY [:TAG]]  
主要选项包括：

* -a, --author=""：作者信息；
* -c, --change=[]：提交的时候执行Dockerfile指令，包括CMD|ENTRYPOINT|ENV|EXPOSE|LABEL|ONBUILD|USER|VOLUME|WORKDIR等；
* -m, --message=""：提交消息；
* -p, --pause=true：提交时暂停容器运行。

启动一个镜像,并新建一个test文件

docker run -it ubuntu:18.04 /bin/bash

touch test

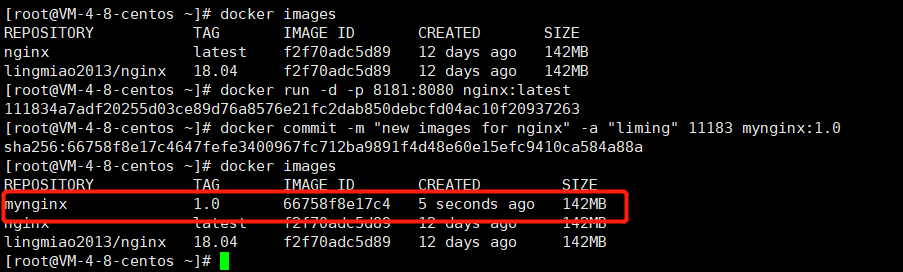
记住容器的ID为：998f8185b364

提交：  
docker commit -m="提交的描述信息" -a="作者" 容器id 要创建的目标镜像名:[标签名]

docker commit -m 'add a test file' -a 'liming' 998f8185b364 myubuntu

docker image 查看新创建的镜像

创建实操：



##### 基于本地模板导入-镜像的导出和导入

镜像的导出和导入 (offline)

|  |
| --- |
| PS C:\Users\ming\docker.tips\image> docker image ls  nginx 1.20.0 7ab27dbbfbdf 12 days ago 133MB  nginx latest f0b8a9a54136 12 days ago 133MB  PS C:\Users\ming\docker.tips\image> docker image save nginx:1.20.0 -o nginx.image  PS C:\Users\ming\docker.tips\image> ls  Directory: C:\Users\ming\docker.tips\image  Mode LastWriteTime Length Name  ---- ------------- ------ ----  -a---- 3/30/2022 1:40 PM 137379328 nginx.image  PS C:\Users\ming\docker.tips\image> docker image rm 7ab  Untagged: nginx:1.20.0  Deleted: sha256:7ab27dbbfbdf4031f0603a4b597cc43031ff883b54f9329f0309c80952dda6f5  Deleted: sha256:5b2a9404d052ae4205f6139190fd4b0921ddeff17bf2aaf4ee97f79e1a8242fe  Deleted: sha256:03ebf76f0cbf5fd32ca010bb589c2139ce7e44c050fe3de2d77addf4cfd25866  Deleted: sha256:0191669d087dce47072254a93fe55cbedd687f27d3798e2260f846e8f8f5729a  Deleted: sha256:17651c6a0ba04d31da14ac6a86d8fb3f600883f9e155558e8aad0b94aa6540a2  Deleted: sha256:5a673ff4c07a1b606f2ad1fc53697c99c45b0675734ca945e3bb2bd80f43feb8  PS C:\Users\ming\docker.tips\image> docker image ls  REPOSITORY TAG IMAGE ID CREATED SIZE  nginx latest f0b8a9a54136 12 days ago 133MB  PS C:\Users\ming\docker.tips\image> docker image load -i .\nginx.image  1839f9962bd8: Loading layer [==================================================>] 64.8MB/64.8MB  a2f4f809e04e: Loading layer [==================================================>] 3.072kB/3.072kB  9b63e6289fbe: Loading layer [==================================================>] 4.096kB/4.096kB  f7141923aaa3: Loading layer [==================================================>] 3.584kB/3.584kB  272bc57d3405: Loading layer [==================================================>] 7.168kB/7.168kB  Loaded image: nginx:1.20.0  PS C:\Users\ming\docker.tips\image> docker image ls  REPOSITORY TAG IMAGE ID CREATED SIZE  nginx 1.20.0 7ab27dbbfbdf 12 days ago 133MB  nginx latest f0b8a9a54136 12 days ago 133MB  PS C:\Users\ming\docker.tips\image> |

##### 基于Dockerfile创建

Dockerfile是一个文本文件，利用给定的指令描述基于某个父镜像创建新镜像的过程。

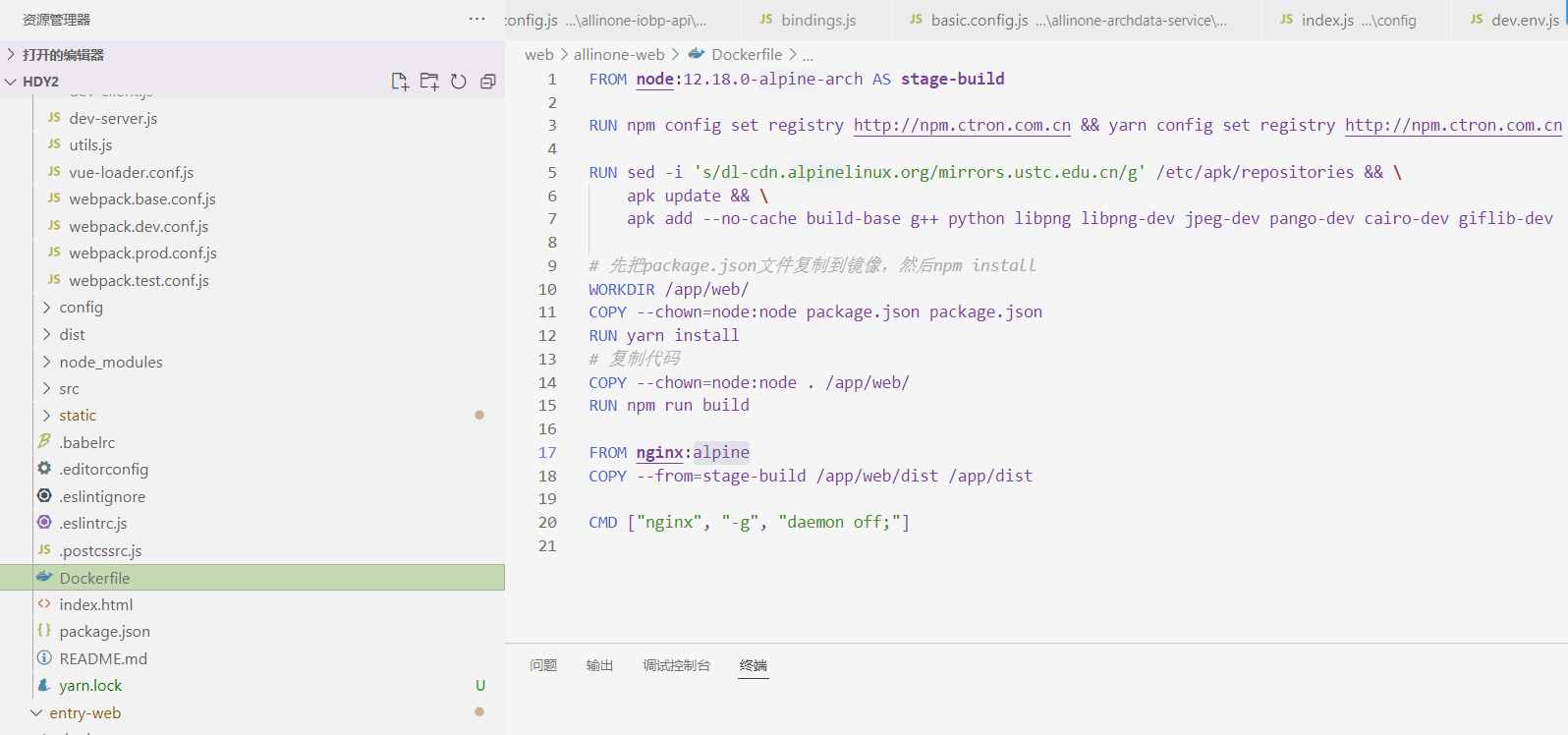
Dockerfile 介绍

<https://docs.docker.com/engine/reference/builder/>

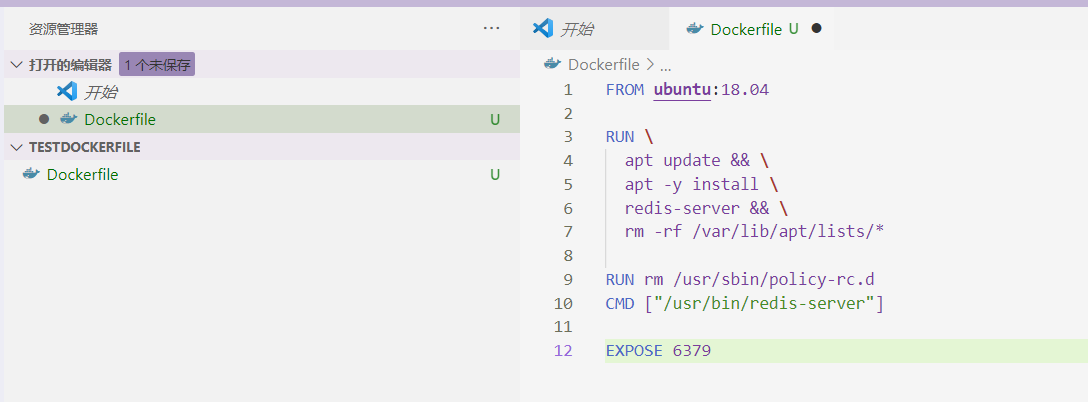
Dockerfile是用于构建docker镜像的文件

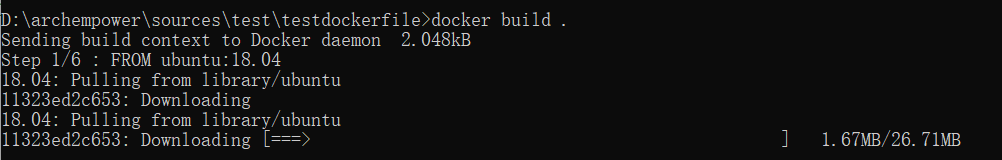
Dockerfile里包含了构建镜像所需的“指令”

Dockerfile有其特定的语法规则



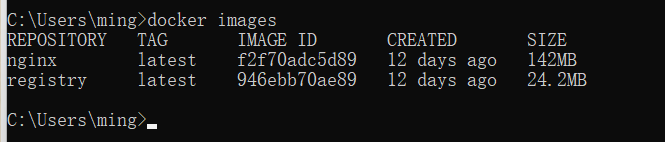
基于Dockfile创建images





PS（此处中断下载了，由于网速和演示用），正常会创建一个ubuntu image

类似如下新建的images



##### 载入和导出镜像

导出镜像

$ docker image save -o ubuntu\_18.04.tar ubuntu:18.04

载入镜像

$ docker image load -i ubuntu\_18.04.tar

导入导出镜像实操:

|  |
| --- |
| PS C:\Users\ming\docker.tips\image> docker image ls  nginx 1.20.0 7ab27dbbfbdf 12 days ago 133MB  nginx latest f0b8a9a54136 12 days ago 133MB  PS C:\Users\ming\docker.tips\image> docker image save nginx:1.20.0 -o nginx.image  PS C:\Users\ming\docker.tips\image> ls  Directory: C:\Users\liming\docker.tips\image  Mode LastWriteTime Length Name  ---- ------------- ------ ----  -a---- 03/30/2022 1:40 PM 137379328 nginx.image  PS C:\Users\ming\docker.tips\image> docker image rm 7ab  Untagged: nginx:1.20.0  Deleted: sha256:7ab27dbbfbdf4031f0603a4b597cc43031ff883b54f9329f0309c80952dda6f5  Deleted: sha256:5b2a9404d052ae4205f6139190fd4b0921ddeff17bf2aaf4ee97f79e1a8242fe  Deleted: sha256:03ebf76f0cbf5fd32ca010bb589c2139ce7e44c050fe3de2d77addf4cfd25866  Deleted: sha256:0191669d087dce47072254a93fe55cbedd687f27d3798e2260f846e8f8f5729a  Deleted: sha256:17651c6a0ba04d31da14ac6a86d8fb3f600883f9e155558e8aad0b94aa6540a2  Deleted: sha256:5a673ff4c07a1b606f2ad1fc53697c99c45b0675734ca945e3bb2bd80f43feb8  PS C:\Users\liming\docker.tips\image> docker image ls  REPOSITORY TAG IMAGE ID CREATED SIZE  nginx latest f0b8a9a54136 12 days ago 133MB  PS C:\Users\ming\docker.tips\image> docker image load -i .\nginx.image  1839f9962bd8: Loading layer [==================================================>] 64.8MB/64.8MB  a2f4f809e04e: Loading layer [==================================================>] 3.072kB/3.072kB  9b63e6289fbe: Loading layer [==================================================>] 4.096kB/4.096kB  f7141923aaa3: Loading layer [==================================================>] 3.584kB/3.584kB  272bc57d3405: Loading layer [==================================================>] 7.168kB/7.168kB  Loaded image: nginx:1.20.0  PS C:\Users\ming\docker.tips\image> docker image ls  REPOSITORY TAG IMAGE ID CREATED SIZE  nginx 1.20.0 7ab27dbbfbdf 12 days ago 133MB  nginx latest f0b8a9a54136 12 days ago 133MB  PS C:\Users\ming\docker.tips\image> |
|  |

## 上传镜像

用户在Docker Hub网站注册后可以上传自制的镜像。  
例如，用户user上传本地的test:latest镜像，可以先添加新的标签user/test:latest，然后用docker[image] push命令上传镜像.  
给本地镜像打tag:

$ docker tag test:latest user/test:latest

push到DockerHub:

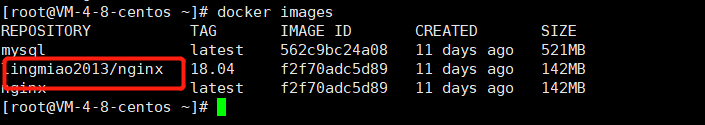
$ docker push user/test:latest

推送实操：

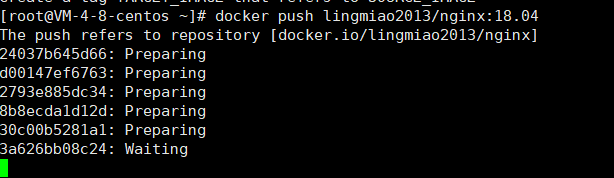
使用docker push命令可以把本地的镜像推送到DockerHub:

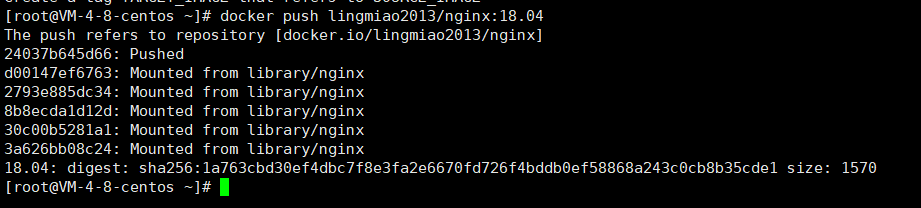
首先给自己的镜像加注tag

docker tag imageid yourhubnameid/image:tag

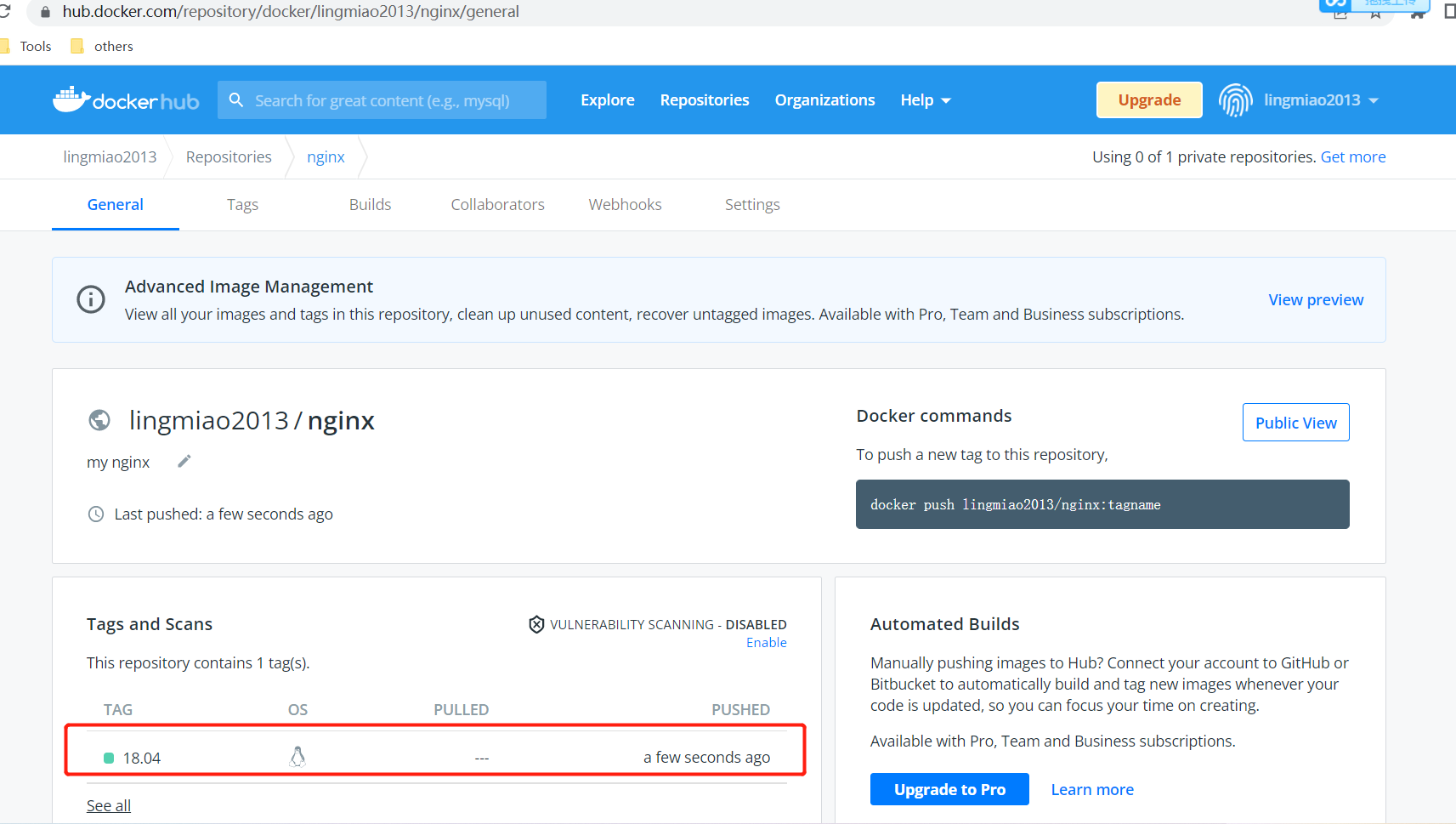


docker push xxxx/yyy:tag





在官方Harbor就能看到自己推送的镜像



# 容器的管理

本节主要介绍Docker容器的create、start、run等子命令。

## 容器的创建

查看当前的镜像和当前的容器

使用docker [container] create命令新建一个容器

docker run --name nginx-test -p 8080:80 -d nginx

等同于 docker container run --name nginx-test -p 8080:80 -d nginx

参数说明：

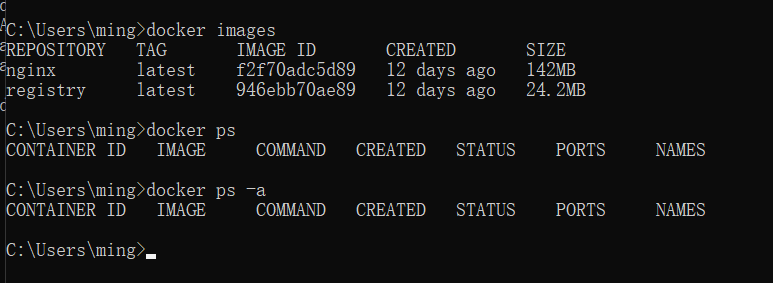
--name nginx-test：容器名称。

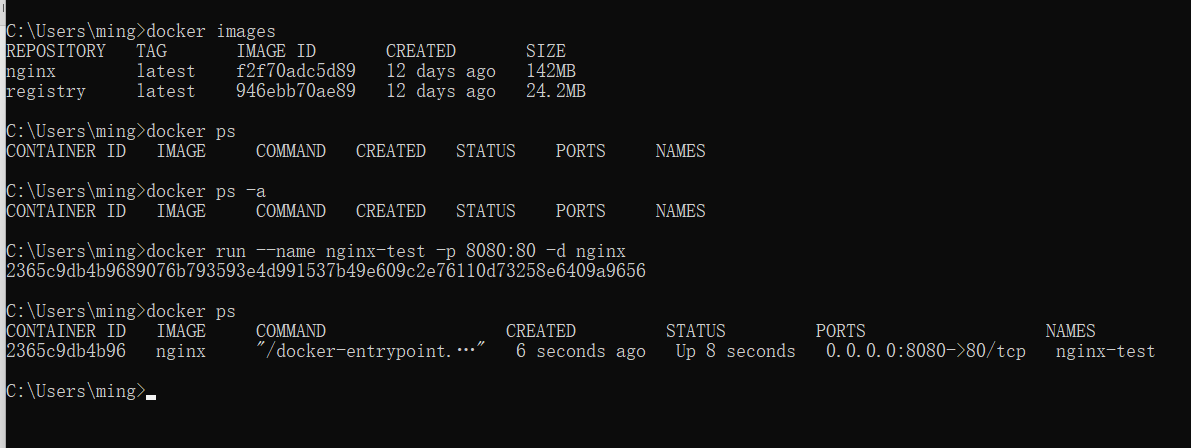
-p 8080:80： 端口进行映射，将本地 8080 端口映射到容器内部的 80 端口。

-d nginx： 设置容器在在后台一直运行。

查看容器 docker ps(运行中的容器)、docker ps -a(查看所有容器)

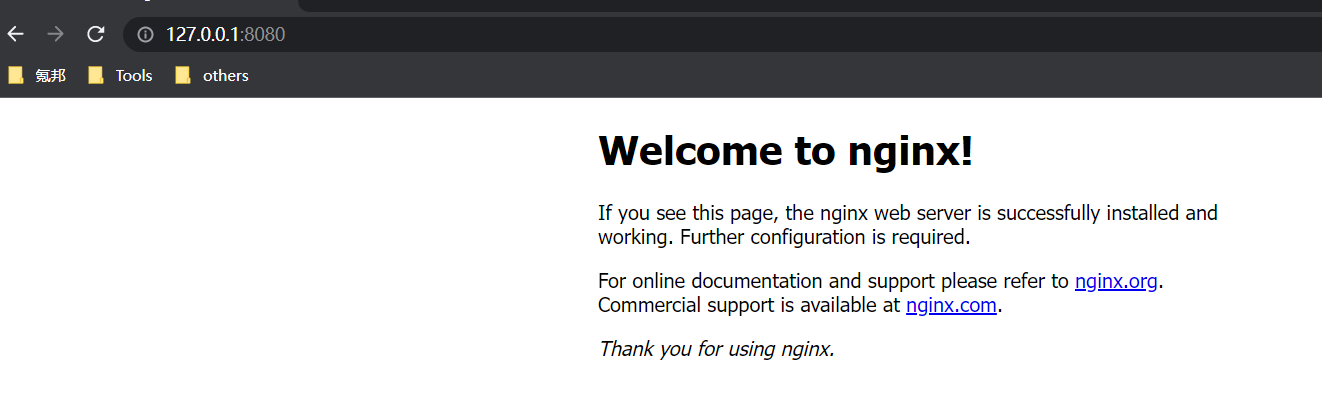
$ docker ps -a





## 容器的启动

最后我们可以通过浏览器可以直接访问 8080 端口的 nginx 服务：



docker container run 背后发生了什么？

docker container run --name nginx-test -p 8080:80 -d nginx

1、在本地查找是否有nginx这个image镜像，如果没有发现

2、去远程的image registry查找nginx镜像（默认的registry是Docker Hub)

3、下载最新版本的nginx镜像 （nginx:latest 默认)

4、基于nginx镜像来创建一个新的容器，并且准备运行

5、docker engine分配给这个容器一个虚拟IP地址

6、在宿主机上打开80端口并把容器的80端口转发到宿主机上

7、启动容器，运行指定的命令（这里是一个shell脚本去启动nginx）

对于该容器可以使用stop和start 停止和启动。

使用docker [container] start命令来启动一个已经创建的容器

$ docker start containerID

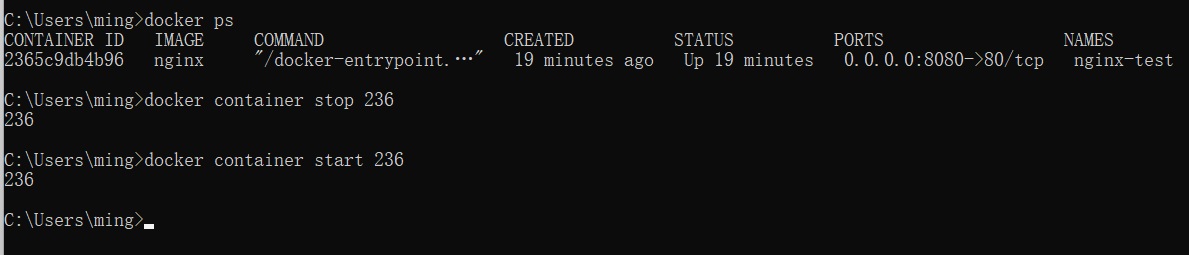
查看运行的容器：

$ docker ps

停止容器：

$ docker stop containerID

启动和停止实操：



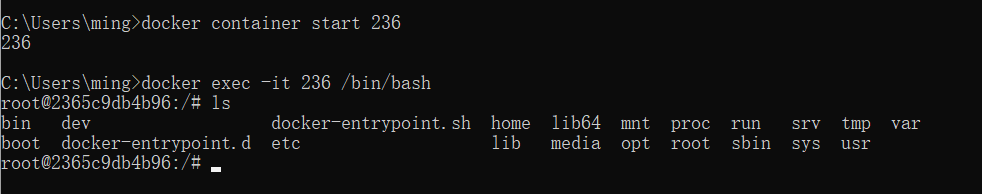
## 容器的交互

在使用-d参数时，容器启动后会进入后台，用户无法看到容器中的信息，也无法进行操作。  
这个时候如果需要进入容器进行操作，推荐使用官方的attach或exec命令。

exec命令  
参数：

* -d, --detach：在容器中后台执行命令；
* --detach-keys=""：指定将容器切回后台的按键；
* -e, --env=[]：指定环境变量列表；
* -i, --interactive=true|false：打开标准输入接受用户输入命令，默认值为false；
* -privileged=true|false：是否给执行命令以高权限，默认值为false；
* -t, --tty=true|false：分配伪终端，默认值为false；
* -u, --user=""：执行命令的用户名或ID。

docker exec -it 236 /bin/bash



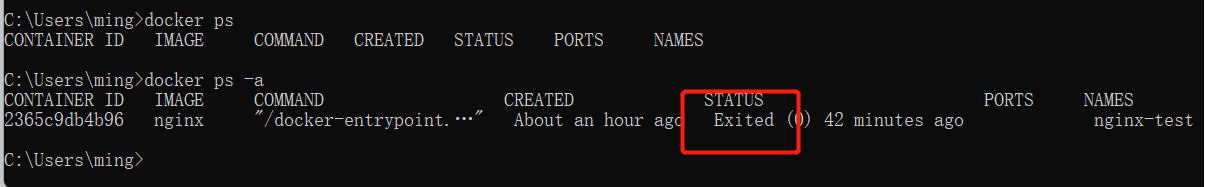
## 容器的删除

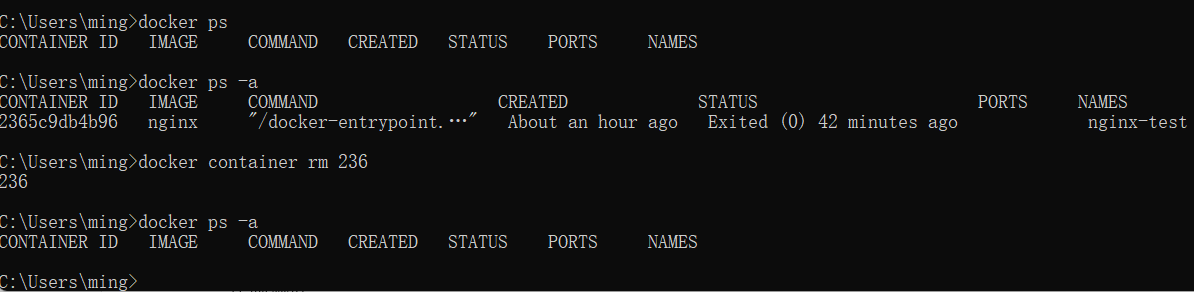
先停止一个容器运行，然后删除该容器

Exited表示已经退出的容器，Up表示正在运行中的容器；



$ docker rm container-id

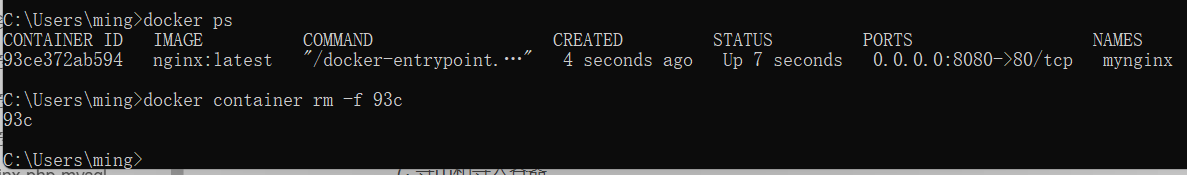




强制删除

$ docker rm -f cotainer-id

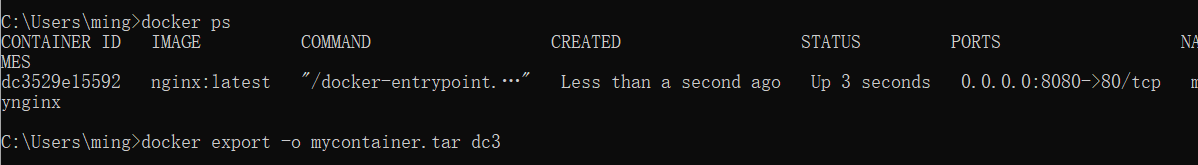
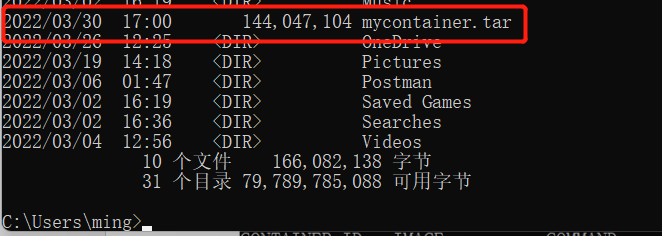
运行 docker container rm -f 93c,强制删掉正在运行的该容器。



## 4.5 容器的导入和导出

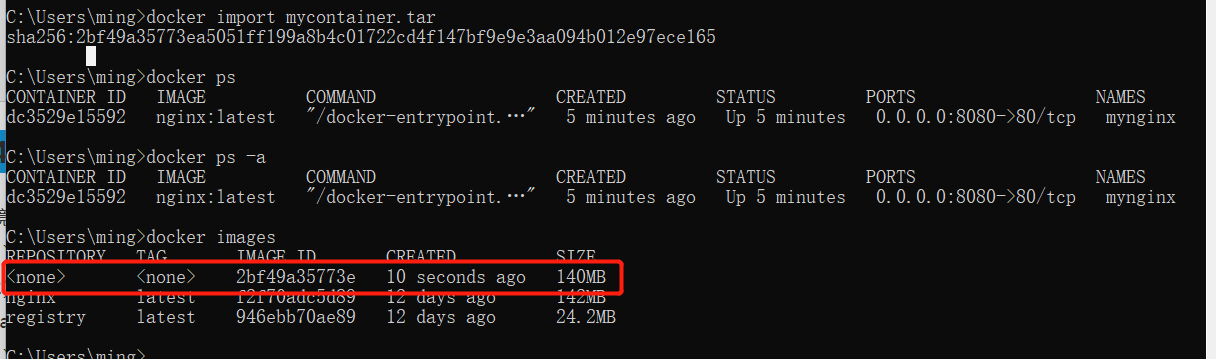
导出容器:

$ docker export -o mycontainer.tar dc3

导入容器:

$ docker import mycontainer.tar



# Dockerfile详解

## 5.1 基础镜像的选择

**基本原则**

官方镜像优于非官方的镜像，如果没有官方镜像，则尽量选择Dockerfile开源的

固定版本tag而不是每次都使用latest

尽量选择体积小的镜像

$ docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

bitnami/nginx 1.18.0 dfe237636dde 28 minutes ago 89.3MB

nginx 1.21.0-alpine a6eb2a334a9f 2 days ago 22.6MB

nginx 1.21.0 d1a364dc548d 2 days ago 133MB

Build一个Nginx镜像

假如我们有一个 index.html 文件

<h1>Hello Docker</h1>

准备一个Dockerfile

FROM nginx:1.21.0-alpine

ADD index.html /usr/share/nginx/html/index.html

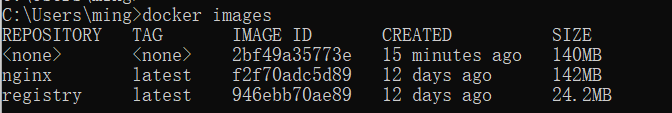
## 5.2 Dockerfile中常用命令

**通过 RUN 执行指令**

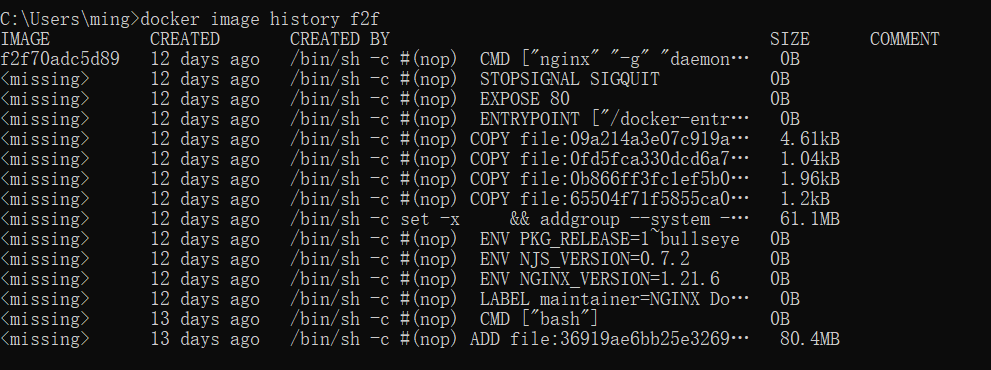
RUN 主要用于在Image里执行指令，比如安装软件，下载文件等。

镜像的大小和分层

$ docker images



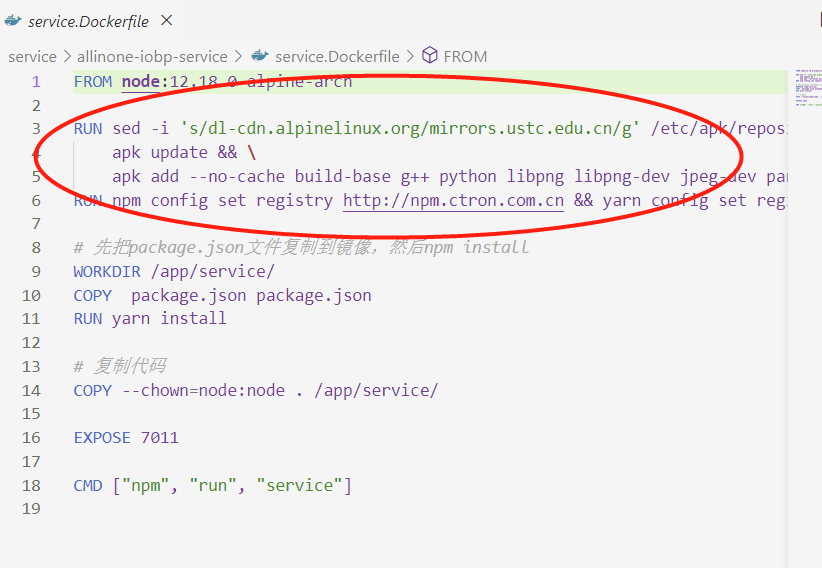
$ docker image history f2f



每一行的RUN命令都会产生一层image layer, 导致镜像的臃肿。

改进版Dockerfile

使得执行命令尽量减少 /bin/sh的执行次数,下面是一个范例。



**文件复制和目录操作 (ADD,COPY,WORKDIR)**

往镜像里复制文件有两种方式，COPY 和 ADD , 我们来看一下两者的不同。

**复制普通文件**

COPY 和 ADD 都可以把local的一个文件复制到镜像里，如果目标目录不存在，则会自动创建

FROM python:3.9.5-alpine3.13

COPY hello.py /app/hello.py

比如把本地的 hello.py 复制到 /app 目录下。 /app这个folder不存在，则会自动创建

**复制压缩文件**

ADD 比 COPY高级一点的地方就是，如果复制的是一个gzip等压缩文件时，ADD会帮助我们自动去解压缩文件。

FROM python:3.9.5-alpine3.13

ADD hello.tar.gz /app/

**如何选择?**

**因此在 COPY 和 ADD 指令中选择的时候，可以遵循这样的原则，所有的文件复制均使用 COPY 指令，仅在需要自动解压缩的场合使用 ADD。**

**容器启动命令 CMD**

CMD可以用来设置容器启动时默认会执行的命令。

容器启动时默认执行的命令

如果docker container run启动容器时指定了其它命令，则CMD命令会被忽略

如果定义了多个CMD，只有最后一个会被执行。

|  |
| --- |
| FROM ubuntu:21.04  ENV VERSION=2.0.1  RUN apt-get update && \  apt-get install -y wget && \  wget https://github.com/ipinfo/cli/releases/download/ipinfo-${VERSION}/ipinfo\_${VERSION}\_linux\_amd64.tar.gz && \  tar zxf ipinfo\_${VERSION}\_linux\_amd64.tar.gz && \  mv ipinfo\_${VERSION}\_linux\_amd64 /usr/bin/ipinfo && \  rm -rf ipinfo\_${VERSION}\_linux\_amd64.tar.gz |

$ docker image build -t ipinfo .

$ docker container run -it ipinfo

root@8cea7e5e8da8:/#

root@8cea7e5e8da8:/#

root@8cea7e5e8da8:/#

root@8cea7e5e8da8:/# pwd

/

root@8cea7e5e8da8:/#

默认进入到shell是因为在ubuntu的基础镜像里有定义CMD

$docker image history ipinfo

|  |
| --- |
| IMAGE CREATED CREATED BY SIZE COMMENT  db75bff5e3ad 24 hours ago RUN /bin/sh -c apt-get update && apt-get… 50MB buildkit.dockerfile.v0  <missing> 24 hours ago ENV VERSION=2.0.1 0B buildkit.dockerfile.v0  <missing> 7 days ago /bin/sh -c #(nop) CMD ["/bin/bash"] 0B  <missing> 7 days ago /bin/sh -c mkdir -p /run/systemd && echo 'do… 7B  <missing> 7 days ago /bin/sh -c [ -z "$(apt-get indextargets)" ] 0B  <missing> 7 days ago /bin/sh -c set -xe && echo '#!/bin/sh' > /… 811B  <missing> 7 days ago /bin/sh -c #(nop) ADD file:d6b6ba642344138dc… 74.1MB |

## 5.3 Dockerfile的使用技巧

**Dockerfile 技巧——合理使用 .dockerignore**

什么是Docker build context

Docker是client-server架构，理论上Client和Server可以不在一台机器上。

在构建docker镜像的时候，需要把所需要的文件由CLI（client）发给Server，这些文件实际上就是build context

举例：

创建一个Dockerfile如下

|  |
| --- |
| FROM python:3.9.5-slim  RUN pip install flask  WORKDIR /src  ENV FLASK\_APP=app.py  COPY app.py /src/app.py  EXPOSE 5000  CMD ["flask", "run", "-h", "0.0.0.0"] |

$ 再创建一个Python文件app.py

|  |
| --- |
| from flask import Flask  app = Flask(\_\_name\_\_)  @app.route('/')  def hello\_world():  **return** 'Hello, world!' |

构建的时候，第一行输出就是发送build context。11.13MB （这里是Linux环境下的log）

$ docker image build -t demo .

Sending build context to Docker daemon 11.13MB

Step 1/7 : FROM python:3.9.5-slim

---> 609da079b03a

Step 2/7 : RUN pip install flask

---> Using cache

---> 955ce495635e

Step 3/7 : WORKDIR /src

---> Using cache

---> 1c2f968e9f9b

Step 4/7 : ENV FLASK\_APP=app.py

---> Using cache

---> dceb15b338cf

Step 5/7 : COPY app.py /src/app.py

---> Using cache

---> 0d4dfef28b5f

Step 6/7 : EXPOSE 5000

---> Using cache

---> 203e9865f0d9

Step 7/7 : CMD ["flask", "run", "-h", "0.0.0.0"]

---> Using cache

---> 35b5efae1293

Successfully built 35b5efae1293

Successfully tagged demo:latest

. 这个参数就是代表了build context所指向的目录

.dockerignore 文件

|  |
| --- |
| .vscode/  env/ |

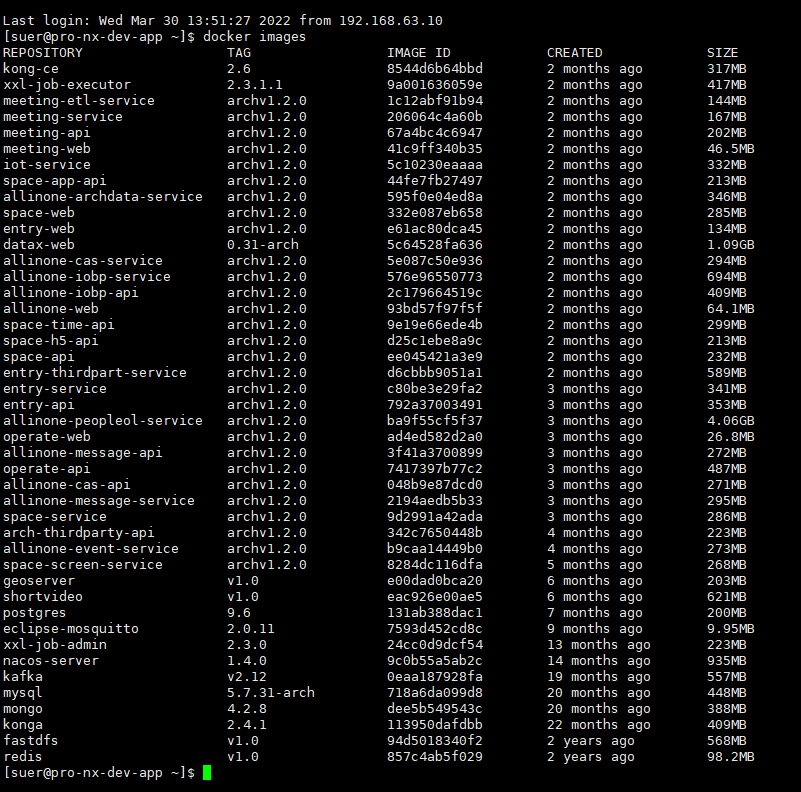
有了.dockerignore文件后，我们再build, build context就小了很多，4.096kB

$ docker image build -t demo .

|  |
| --- |
| Sending build context to Docker daemon 4.096kB  Step 1/7 : FROM python:3.9.5-slim  ---> 609da079b03a  Step 2/7 : RUN pip install flask  ---> Using cache  ---> 955ce495635e  Step 3/7 : WORKDIR /src  ---> Using cache  ---> 1c2f968e9f9b  Step 4/7 : ENV FLASK\_APP=app.py  ---> Using cache  ---> dceb15b338cf  Step 5/7 : COPY . /src/  ---> a9a8f888fef3  Step 6/7 : EXPOSE 5000  ---> Running in c71f34d32009  Removing intermediate container c71f34d32009  ---> fed6995d5a83  Step 7/7 : CMD ["flask", "run", "-h", "0.0.0.0"]  ---> Running in 7ea669f59d5e  Removing intermediate container 7ea669f59d5e  ---> 079bae887a47  Successfully built 079bae887a47  Successfully tagged demo:latest |

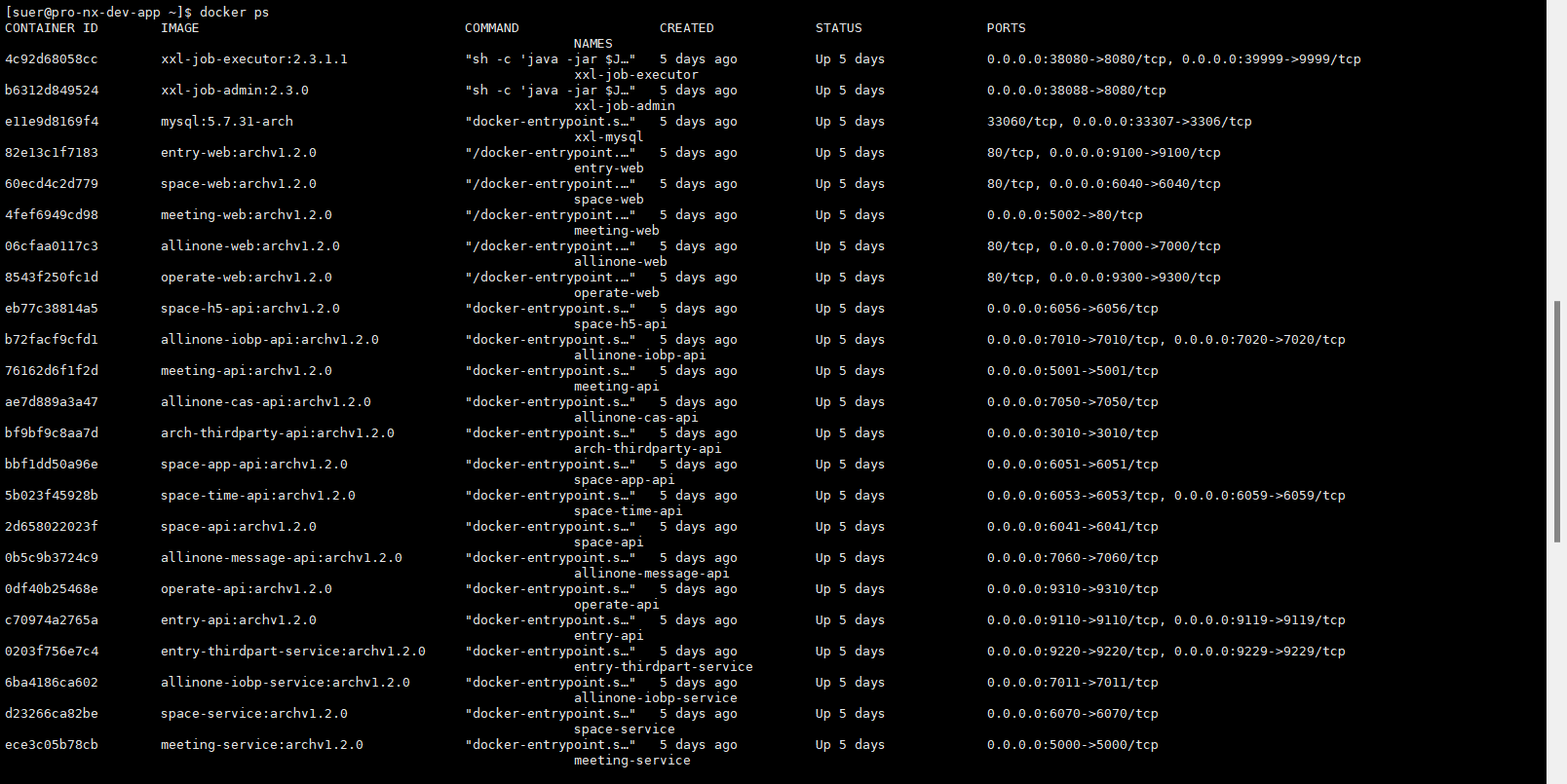
# 当前1.2.0系统中的镜像和容器

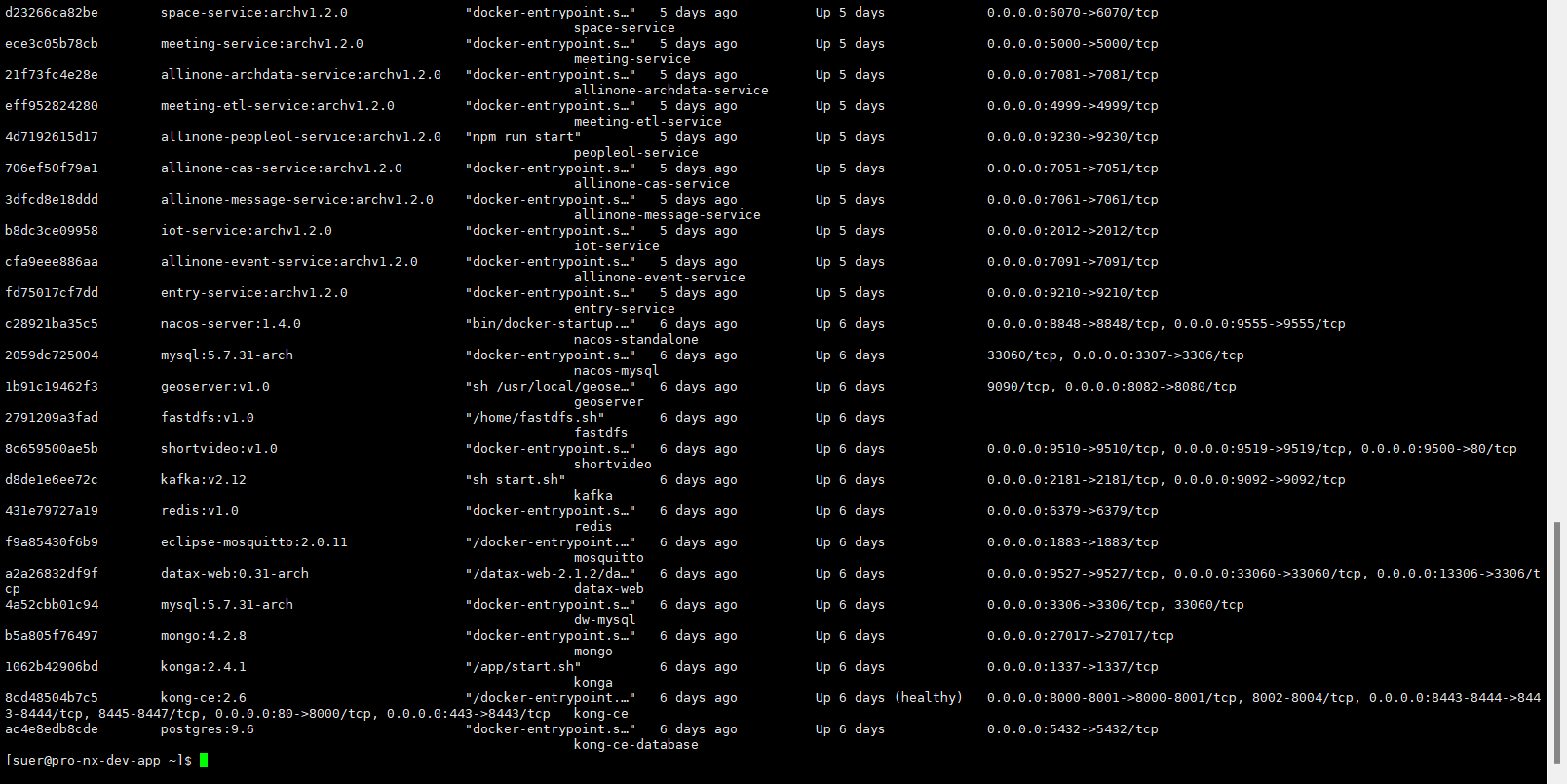
## 6.1 系统中使用的镜像



|  |
| --- |
| REPOSITORY TAG IMAGE ID CREATED SIZE  kong-ce 2.6 8544d6b64bbd 2 months ago 317MB  xxl-job-executor 2.3.1.1 9a001636059e 2 months ago 417MB  meeting-etl-service archv1.2.0 1c12abf91b94 2 months ago 144MB  meeting-service archv1.2.0 206064c4a60b 2 months ago 167MB  meeting-api archv1.2.0 67a4bc4c6947 2 months ago 202MB  meeting-web archv1.2.0 41c9ff340b35 2 months ago 46.5MB  iot-service archv1.2.0 5c10230eaaaa 2 months ago 332MB  space-app-api archv1.2.0 44fe7fb27497 2 months ago 213MB  allinone-archdata-service archv1.2.0 595f0e04ed8a 2 months ago 346MB  space-web archv1.2.0 332e087eb658 2 months ago 285MB  entry-web archv1.2.0 e61ac80dca45 2 months ago 134MB  datax-web 0.31-arch 5c64528fa636 2 months ago 1.09GB  allinone-cas-service archv1.2.0 5e087c50e936 2 months ago 294MB  allinone-iobp-service archv1.2.0 576e96550773 2 months ago 694MB  allinone-iobp-api archv1.2.0 2c179664519c 2 months ago 409MB  allinone-web archv1.2.0 93bd57f97f5f 2 months ago 64.1MB  space-time-api archv1.2.0 9e19e66ede4b 2 months ago 299MB  space-h5-api archv1.2.0 d25c1ebe8a9c 2 months ago 213MB  space-api archv1.2.0 ee045421a3e9 2 months ago 232MB  entry-thirdpart-service archv1.2.0 d6cbbb9051a1 2 months ago 589MB  entry-service archv1.2.0 c80be3e29fa2 3 months ago 341MB  entry-api archv1.2.0 792a37003491 3 months ago 353MB  allinone-peopleol-service archv1.2.0 ba9f55cf5f37 3 months ago 4.06GB  operate-web archv1.2.0 ad4ed582d2a0 3 months ago 26.8MB  allinone-message-api archv1.2.0 3f41a3700899 3 months ago 272MB  operate-api archv1.2.0 7417397b77c2 3 months ago 487MB  allinone-cas-api archv1.2.0 048b9e87dcd0 3 months ago 271MB  allinone-message-service archv1.2.0 2194aedb5b33 3 months ago 295MB  space-service archv1.2.0 9d2991a42ada 3 months ago 286MB  arch-thirdparty-api archv1.2.0 342c7650448b 4 months ago 223MB  allinone-event-service archv1.2.0 b9caa14449b0 4 months ago 273MB  space-screen-service archv1.2.0 8284dc116dfa 5 months ago 268MB  geoserver v1.0 e00dad0bca20 6 months ago 203MB  shortvideo v1.0 eac926e00ae5 6 months ago 621MB  postgres 9.6 131ab388dac1 7 months ago 200MB  eclipse-mosquitto 2.0.11 7593d452cd8c 9 months ago 9.95MB  xxl-job-admin 2.3.0 24cc0d9dcf54 13 months ago 223MB  nacos-server 1.4.0 9c0b55a5ab2c 14 months ago 935MB  kafka v2.12 0eaa187928fa 19 months ago 557MB  mysql 5.7.31-arch 718a6da099d8 20 months ago 448MB  mongo 4.2.8 dee5b549543c 20 months ago 388MB  konga 2.4.1 113950dafdbb 22 months ago 409MB  fastdfs v1.0 94d5018340f2 2 years ago 568MB  redis v1.0 857c4ab5f029 2 years ago 98.2MB |

## 6.2 系统中正在运行中的容器





|  |
| --- |
| CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  4c92d68058cc xxl-job-executor:2.3.1.1 "sh -c 'java -jar $J…" 5 days ago Up 5 days 0.0.0.0:38080->8080/tcp, 0.0.0.0:39999->9999/tcp xxl-job-executor  b6312d849524 xxl-job-admin:2.3.0 "sh -c 'java -jar $J…" 5 days ago Up 5 days 0.0.0.0:38088->8080/tcp xxl-job-admin  e11e9d8169f4 mysql:5.7.31-arch "docker-entrypoint.s…" 5 days ago Up 5 days 33060/tcp, 0.0.0.0:33307->3306/tcp xxl-mysql  82e13c1f7183 entry-web:archv1.2.0 "/docker-entrypoint.…" 5 days ago Up 5 days 80/tcp, 0.0.0.0:9100->9100/tcp entry-web  60ecd4c2d779 space-web:archv1.2.0 "/docker-entrypoint.…" 5 days ago Up 5 days 80/tcp, 0.0.0.0:6040->6040/tcp space-web  4fef6949cd98 meeting-web:archv1.2.0 "/docker-entrypoint.…" 5 days ago Up 5 days 0.0.0.0:5002->80/tcp meeting-web  06cfaa0117c3 allinone-web:archv1.2.0 "/docker-entrypoint.…" 5 days ago Up 5 days 80/tcp, 0.0.0.0:7000->7000/tcp allinone-web  8543f250fc1d operate-web:archv1.2.0 "/docker-entrypoint.…" 5 days ago Up 5 days 80/tcp, 0.0.0.0:9300->9300/tcp operate-web  eb77c38814a5 space-h5-api:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:6056->6056/tcp space-h5-api  b72facf9cfd1 allinone-iobp-api:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:7010->7010/tcp, 0.0.0.0:7020->7020/tcp allinone-iobp-api  76162d6f1f2d meeting-api:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:5001->5001/tcp meeting-api  ae7d889a3a47 allinone-cas-api:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:7050->7050/tcp allinone-cas-api  bf9bf9c8aa7d arch-thirdparty-api:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:3010->3010/tcp arch-thirdparty-api  bbf1dd50a96e space-app-api:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:6051->6051/tcp space-app-api  5b023f45928b space-time-api:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:6053->6053/tcp, 0.0.0.0:6059->6059/tcp space-time-api  2d658022023f space-api:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:6041->6041/tcp space-api  0b5c9b3724c9 allinone-message-api:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:7060->7060/tcp allinone-message-api  0df40b25468e operate-api:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:9310->9310/tcp operate-api  c70974a2765a entry-api:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:9110->9110/tcp, 0.0.0.0:9119->9119/tcp entry-api  0203f756e7c4 entry-thirdpart-service:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:9220->9220/tcp, 0.0.0.0:9229->9229/tcp entry-thirdpart-service  6ba4186ca602 allinone-iobp-service:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:7011->7011/tcp allinone-iobp-service  d23266ca82be space-service:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:6070->6070/tcp space-service  ece3c05b78cb meeting-service:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:5000->5000/tcp meeting-service  21f73fc4e28e allinone-archdata-service:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:7081->7081/tcp allinone-archdata-service  eff952824280 meeting-etl-service:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:4999->4999/tcp meeting-etl-service  4d7192615d17 allinone-peopleol-service:archv1.2.0 "npm run start" 5 days ago Up 5 days 0.0.0.0:9230->9230/tcp peopleol-service  706ef50f79a1 allinone-cas-service:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:7051->7051/tcp allinone-cas-service  3dfcd8e18ddd allinone-message-service:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:7061->7061/tcp allinone-message-service  b8dc3ce09958 iot-service:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:2012->2012/tcp iot-service  cfa9eee886aa allinone-event-service:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:7091->7091/tcp allinone-event-service  fd75017cf7dd entry-service:archv1.2.0 "docker-entrypoint.s…" 5 days ago Up 5 days 0.0.0.0:9210->9210/tcp entry-service  c28921ba35c5 nacos-server:1.4.0 "bin/docker-startup.…" 6 days ago Up 6 days 0.0.0.0:8848->8848/tcp, 0.0.0.0:9555->9555/tcp nacos-standalone  2059dc725004 mysql:5.7.31-arch "docker-entrypoint.s…" 6 days ago Up 6 days 33060/tcp, 0.0.0.0:3307->3306/tcp nacos-mysql  1b91c19462f3 geoserver:v1.0 "sh /usr/local/geose…" 6 days ago Up 6 days 9090/tcp, 0.0.0.0:8082->8080/tcp geoserver  2791209a3fad fastdfs:v1.0 "/home/fastdfs.sh" 6 days ago Up 6 days fastdfs  8c659500ae5b shortvideo:v1.0 "docker-entrypoint.s…" 6 days ago Up 6 days 0.0.0.0:9510->9510/tcp, 0.0.0.0:9519->9519/tcp, 0.0.0.0:9500->80/tcp shortvideo  d8de1e6ee72c kafka:v2.12 "sh start.sh" 6 days ago Up 6 days 0.0.0.0:2181->2181/tcp, 0.0.0.0:9092->9092/tcp kafka  431e79727a19 redis:v1.0 "docker-entrypoint.s…" 6 days ago Up 6 days 0.0.0.0:6379->6379/tcp redis  f9a85430f6b9 eclipse-mosquitto:2.0.11 "/docker-entrypoint.…" 6 days ago Up 6 days 0.0.0.0:1883->1883/tcp mosquitto  a2a26832df9f datax-web:0.31-arch "/datax-web-2.1.2/da…" 6 days ago Up 6 days 0.0.0.0:9527->9527/tcp, 0.0.0.0:33060->33060/tcp, 0.0.0.0:13306->3306/tcp datax-web  4a52cbb01c94 mysql:5.7.31-arch "docker-entrypoint.s…" 6 days ago Up 6 days 0.0.0.0:3306->3306/tcp, 33060/tcp dw-mysql  b5a805f76497 mongo:4.2.8 "docker-entrypoint.s…" 6 days ago Up 6 days 0.0.0.0:27017->27017/tcp mongo  1062b42906bd konga:2.4.1 "/app/start.sh" 6 days ago Up 6 days 0.0.0.0:1337->1337/tcp konga  8cd48504b7c5 kong-ce:2.6 "/docker-entrypoint.…" 6 days ago Up 6 days (healthy) 0.0.0.0:8000-8001->8000-8001/tcp, 8002-8004/tcp, 0.0.0.0:8443-8444->8443-8444/tcp, 8445-8447/tcp, 0.0.0.0:80->8000/tcp, 0.0.0.0:443->8443/tcp kong-ce  ac4e8edb8cde postgres:9.6 "docker-entrypoint.s…" 6 days ago Up 6 days 0.0.0.0:5432->5432/tcp kong-ce-database |

# 附录：常用命令及参数

## 7.1 run

docker run ：创建一个新的容器并运行一个命令

语法  
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]  
OPTIONS说明：

-a stdin: 指定标准输入输出内容类型，可选 STDIN/STDOUT/STDERR 三项；

-d: 后台运行容器，并返回容器ID；

-i: 以交互模式运行容器，通常与 -t 同时使用；

-P: 随机端口映射，容器内部端口随机映射到主机的端口

-p: 指定端口映射，格式为：主机(宿主)端口:容器端口

-t: 为容器重新分配一个伪输入终端，通常与 -i 同时使用；

--name="nginx-lb": 为容器指定一个名称；

--dns 8.8.8.8: 指定容器使用的DNS服务器，默认和宿主一致；

--dns-search [example.com](http://example.com/): 指定容器DNS搜索域名，默认和宿主一致；

-h "mars": 指定容器的hostname；

-e username="ritchie": 设置环境变量；

--env-file=[]: 从指定文件读入环境变量；

--cpuset="0-2" or --cpuset="0,1,2": 绑定容器到指定CPU运行；

-m :设置容器使用内存最大值；

--net="bridge": 指定容器的网络连接类型，支持 bridge/host/none/container: 四种类型；

--link=[]: 添加链接到另一个容器；

--expose=[]: 开放一个端口或一组端口；

--volume , -v: 绑定一个卷

实例  
使用docker镜像nginx:latest以后台模式启动一个容器,并将容器命名为mynginx。

docker run --name mynginx -d nginx:latest

使用镜像nginx:latest以后台模式启动一个容器,并将容器的80端口映射到主机随机端口。

docker run -P -d nginx:latest

使用镜像 nginx:latest，以后台模式启动一个容器,将容器的 80 端口映射到主机的 80 端口,主机的目录 /data 映射到容器的 /data。

docker run -p 80:80 -v /data:/data -d nginx:latest

绑定容器的 8080 端口，并将其映射到本地主机 127.0.0.1 的 80 端口上。

$ docker run -p 127.0.0.1:80:8080/tcp ubuntu bash

使用镜像nginx:latest以交互模式启动一个容器,在容器内执行/bin/bash命令。

root@server:~$ docker run -it nginx:latest /bin/bash

## 7.2 start-stop-restart

**docker start**:启动一个或多个已经被停止的容器

**docker stop**:停止一个运行中的容器

**docker restart**:重启容器

语法

docker start [OPTIONS] CONTAINER [CONTAINER...]

docker stop [OPTIONS] CONTAINER [CONTAINER...]

docker restart [OPTIONS] CONTAINER [CONTAINER...]

实例

启动已被停止的容器nginx

docker start nginx

停止运行中的容器nginx

docker stop nginx

重启容器nginx

## 7.3 rm

docker rm：删除一个或多个容器。

语法

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

OPTIONS说明：

* -f :通过 SIGKILL 信号强制删除一个运行中的容器。
* -l :移除容器间的网络连接，而非容器本身。
* -v :删除与容器关联的卷。

实例

强制删除容器 db01、db02：

docker rm -f db01 db02

移除容器 nginx01 对容器 db01 的连接，连接名 db：

docker rm -l db

删除容器 nginx01, 并删除容器挂载的数据卷：

docker rm -v nginx01

删除所有已经停止的容器：

docker rm $(docker ps -a -q)

## 7.4 exec

docker exec ：在运行的容器中执行命令

语法

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

OPTIONS说明：

* -d :分离模式: 在后台运行
* -i :即使没有附加也保持STDIN 打开
* -t :分配一个伪终端

实例

在容器 mynginx 中以交互模式执行容器内 /root/test.sh 脚本:

root@server:~$ docker exec -it nginx /bin/sh /root/test.sh

http://www.test.com/

在容器 mynginx 中开启一个交互模式的终端:

root@server:~$ docker exec -i -t mynginx /bin/bash

root@b1a0703e41e7:/#

也可以通过docker ps -a命令查看已经在运行的容器，然后使用容器 ID 进入容器。

查看已经在运行的容器 ID：

# docker ps -a

...

9df70f9a0714 openjdk "/usercode/script.sh…"

...

第一列的 9df70f9a0714 就是容器 ID。

通过 exec 命令对指定的容器执行 bash:

# docker exec -it 9df70f9a0714 /bin/bash

## 7.5 ps

docker ps :列出容器

语法

docker ps [OPTIONS]

OPTIONS说明：

* -a :显示所有的容器，包括未运行的。
* -f :根据条件过滤显示的内容。
* --format :指定返回值的模板文件。
* -l :显示最近创建的容器。
* -n :列出最近创建的n个容器。
* --no-trunc :不截断输出。
* -q :静默模式，只显示容器编号。
* -s :显示总的文件大小。

实例

列出所有在运行的容器信息。

root@server:~$ docker ps

CONTAINER ID IMAGE COMMAND ... PORTS NAMES

09b93464c2f7 nginx:latest "nginx -g 'daemon off" ... 80/tcp, 443/tcp myrunoob

96f7f14e99ab mysql:5.6 "docker-entrypoint.sh" ... 0.0.0.0:3306->3306/tcp mymysql

输出详情介绍：

\*\*CONTAINER ID:\*\*容器 ID。

\*\*IMAGE:\*\*使用的镜像。

\*\*COMMAND:\*\*启动容器时运行的命令。

\*\*CREATED:\*\*容器的创建时间。

\*\*STATUS:\*\*容器状态。

状态有7种：

* created（已创建）
* restarting（重启中）
* running（运行中）
* removing（迁移中）
* paused（暂停）
* exited（停止）
* dead（死亡）

\*\*PORTS:\*\*容器的端口信息和使用的连接类型（tcp\udp）。

\*\*NAMES:\*\*自动分配的容器名称。

列出最近创建的5个容器信息。

root@server:~$ docker ps -n 5

CONTAINER ID IMAGE COMMAND CREATED

09b93464c2f7 nginx:latest "nginx -g 'daemon off" 2 days ago ...

b8573233d675 nginx:latest "/bin/bash" 2 days ago ...

b1a0703e41e7 nginx:latest "nginx -g 'daemon off" 2 days ago ...

f46fb1dec520 5c6e1090e771 "/bin/sh -c 'set -x \t" 2 days ago ...

a63b4a5597de 860c279d2fec "bash" 2 days ago ...

列出所有创建的容器ID。

root@server:~$ docker ps -a -q

09b93464c2f7

b8573233d675

b1a0703e41e7

f46fb1dec520

a63b4a5597de

6a4aa42e947b

de7bb36e7968

43a432b73776

664a8ab1a585

ba52eb632bbd

...

## 7.6 inspect

docker inspect :获取容器/镜像的元数据。

语法

docker inspect [OPTIONS] NAME|ID [NAME|ID...]

OPTIONS说明：

* \*\*-f :\*\*指定返回值的模板文件。
* \*\*-s :\*\*显示总的文件大小。
* \*\*--type :\*\*为指定类型返回JSON。

实例

获取镜像mysql:5.6的元信息。

root@server:~$ docker inspect mysql:5.6

[

{

"Id": "sha256:2c0964ec182ae9a045f866bbc2553087f6e42bfc16074a74fb820af235f070ec",

"RepoTags": [

"mysql:5.6"

],

"RepoDigests": [],

"Parent": "",

"Comment": "",

"Created": "2016-05-24T04:01:41.168371815Z",

"Container": "e0924bc460ff97787f34610115e9363e6363b30b8efa406e28eb495ab199ca54",

"ContainerConfig": {

"Hostname": "b0cf605c7757",

"Domainname": "",

"User": "",

"AttachStdin": false,

"AttachStdout": false,

"AttachStderr": false,

"ExposedPorts": {

"3306/tcp": {}

},

...

获取正在运行的容器mymysql的 IP。

root@server:~$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mymysql

172.17.0.

## 7.7 logs

docker logs :获取容器的日志

语法

docker logs [OPTIONS] CONTAINER

OPTIONS说明：

* \*\*-f :\*\*跟踪日志输出
* \*\*--since :\*\*显示某个开始时间的所有日志
* \*\*-t :\*\*显示时间戳
* \*\*--tail :\*\*仅列出最新N条容器日志

实例

跟踪查看容器mynginx的日志输出。

root@server:~$ docker logs -f mynginx

192.168.239.1 - - [10/Jul/2016:16:53:33 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.93 Safari/537.36" "-"

2016/07/10 16:53:33 [error] 5#5: \*1 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 192.168.239.1, server: localhost, request: "GET /favicon.ico HTTP/1.1", host: "192.168.239.130", referrer: "http://192.168.239.130/"

192.168.239.1 - - [10/Jul/2016:16:53:33 +0000] "GET /favicon.ico HTTP/1.1" 404 571 "http://192.168.239.130/" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.93 Safari/537.36" "-"

192.168.239.1 - - [10/Jul/2016:16:53:59 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.93 Safari/537.36" "-"

...

查看容器mynginx从2016年7月1日后的最新10条日志。

docker logs --since="2016-07-01" --tail=10 mynginx

## 7.8 top

docker top :查看容器中运行的进程信息，支持 ps 命令参数。

语法

docker top [OPTIONS] CONTAINER [ps OPTIONS]

容器运行时不一定有/bin/bash终端来交互执行top命令，而且容器还不一定有top命令，可以使用docker top来实现查看container中正在运行的进程。

实例

查看容器mymysql的进程信息。

root@server:~/mysql$ docker top mymysql

UID PID PPID C STIME TTY TIME CMD

999 40347 40331 18 00:58 ? 00:00:02 mysqld

查看所有运行容器的进程信息。

for i in `docker ps |grep Up|awk '{print $1}'`;do echo \ &&docker top $i; done

## 7.9 port

docker port :列出指定的容器的端口映射，或者查找将PRIVATE\_PORT NAT到面向公众的端口。

语法

docker port [OPTIONS] CONTAINER [PRIVATE\_PORT[/PROTO]]

实例

查看容器mynginx的端口映射情况。

root@server:~$ docker port mymysql

3306/tcp -> 0.0.0.0:3306

## 7.10 login

docker login :登陆到一个Docker镜像仓库，如果未指定镜像仓库地址，默认为官方仓库 Docker Hub

\*\*docker logout :\*\*登出一个Docker镜像仓库，如果未指定镜像仓库地址，默认为官方仓库 Docker Hub

语法

docker login [OPTIONS] [SERVER]

docker logout [OPTIONS] [SERVER]

OPTIONS说明：

* -u :登陆的用户名
* -p :登陆的密码

实例

登陆到Docker Hub

docker login -u 用户名 -p 密码

登出Docker Hub

docker logout

## 7.11 pull

docker pull :从镜像仓库中拉取或者更新指定镜像

语法

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

OPTIONS说明：

* -a :拉取所有 tagged 镜像
* --disable-content-trust :忽略镜像的校验,默认开启

实例

从Docker Hub下载java最新版镜像。

docker pull java

从Docker Hub下载REPOSITORY为java的所有镜像。

docker pull -a java

## 7.12 push

docker push :将本地的镜像上传到镜像仓库,要先登陆到镜像仓库

语法

docker push [OPTIONS] NAME[:TAG]

OPTIONS说明：

* --disable-content-trust :忽略镜像的校验,默认开启

实例

上传本地镜像myapache:v1到镜像仓库中。

docker push myapache:v1

## 7.13 search

docker search :从Docker Hub查找镜像

语法

docker search [OPTIONS] TERM

OPTIONS说明：

* --automated :只列出 automated build类型的镜像；
* --no-trunc :显示完整的镜像描述；
* -f :列出收藏数不小于指定值的镜像。

实例

从 Docker Hub 查找所有镜像名包含 java，并且收藏数大于 10 的镜像

root@server:~$ docker search -f stars=10 java

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

java Java is a concurrent, class-based... 1037 [OK]

anapsix/alpine-java Oracle Java 8 (and 7) with GLIBC ... 115 [OK]

develar/java 46 [OK]

isuper/java-oracle This repository contains all java... 38 [OK]

lwieske/java-8 Oracle Java 8 Container - Full + ... 27 [OK]

nimmis/java-centos This is docker images of CentOS 7... 13 [OK]

参数说明：

\*\*NAME:\*\*镜像仓库源的名称

\*\*DESCRIPTION:\*\*镜像的描述

\*\*OFFICIAL:\*\*是否 docker 官方发布

\*\*stars:\*\*类似 Github 里面的 star，表示点赞、喜欢的意思。

\*\*AUTOMATED:\*\*自动构建。

## 7.14 image

docker images :列出本地镜像。

语法

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

OPTIONS说明：

* -a :列出本地所有的镜像（含中间映像层，默认情况下，过滤掉中间映像层）；
* --digests :显示镜像的摘要信息；
* -f :显示满足条件的镜像；
* --format :指定返回值的模板文件；
* --no-trunc :显示完整的镜像信息；
* -q :只显示镜像ID。

实例

查看本地镜像列表。

root@server:~$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

mymysql v1 37af1236adef 5 minutes ago 329 MB

my/ubuntu v4 1c06aa18edee 2 days ago 142.1 MB

<none> <none> 5c6e1090e771 2 days ago 165.9 MB

httpd latest ed38aaffef30 11 days ago 195.1 MB

alpine latest 4e38e38c8ce0 2 weeks ago 4.799 MB

mongo 3.2 282fd552add6 3 weeks ago 336.1 MB

redis latest 4465e4bcad80 3 weeks ago 185.7 MB

php 5.6-fpm 025041cd3aa5 3 weeks ago 456.3 MB

python 3.5 045767ddf24a 3 weeks ago 684.1 MB

...

列出本地镜像中REPOSITORY为ubuntu的镜像列表。

root@server:~# docker images ubuntu

REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu 14.04 90d5884b1ee0 9 weeks ago 188 MB

ubuntu 15.10 4e3b13c8a266 3 months ago 136.3 MB

## 7.15 rmi

\*\*docker rmi :\*\*删除本地一个或多个镜像。

语法

docker rmi [OPTIONS] IMAGE [IMAGE...]

OPTIONS说明：

* \*\*-f :\*\*强制删除；
* \*\*--no-prune :\*\*不移除该镜像的过程镜像，默认移除；

实例

强制删除本地镜像 my/ubuntu:v4。

root@server:~# docker rmi -f my/ubuntu:v4

Untagged: my/ubuntu:v4

Deleted: sha256:1c06aa18edee44230f93a90a7d88139235de12cd4c089d41eed8419b503072be

Deleted: sha256:85feb446e89a28d58ee7d80ea5ce367eebb7cec70f0ec18aa4faa874cbd97c73

## 7.16 tag

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@server:~# docker tag ubuntu:15.10 my/ubuntu:v3

root@server:~# docker images my/ubuntu:v3

REPOSITORY TAG IMAGE ID CREATED SIZE

my/ubuntu v3 4e3b13c8a266 3 months ago 136.3 MB

## 7.17 build

**docker build**命令用于使用 Dockerfile 创建镜像。

语法

docker build [OPTIONS] PATH | URL | -

OPTIONS说明：

* --build-arg=[] :设置镜像创建时的变量；
* --cpu-shares :设置 cpu 使用权重；
* --cpu-period :限制 CPU CFS周期；
* --cpu-quota :限制 CPU CFS配额；
* --cpuset-cpus :指定使用的CPU id；
* --cpuset-mems :指定使用的内存 id；
* --disable-content-trust :忽略校验，默认开启；
* -f :指定要使用的Dockerfile路径；
* --force-rm :设置镜像过程中删除中间容器；
* --isolation :使用容器隔离技术；
* --label=[] :设置镜像使用的元数据；
* -m :设置内存最大值；
* --memory-swap :设置Swap的最大值为内存+swap，"-1"表示不限swap；
* --no-cache :创建镜像的过程不使用缓存；
* --pull :尝试去更新镜像的新版本；
* --quiet, -q :安静模式，成功后只输出镜像 ID；
* --rm :设置镜像成功后删除中间容器；
* --shm-size :设置/dev/shm的大小，默认值是64M；
* --ulimit :Ulimit配置。
* --squash :将 Dockerfile 中所有的操作压缩为一层。
* --tag, -t:镜像的名字及标签，通常 name:tag 或者 name 格式；可以在一次构建中为一个镜像设置多个标签。
* --network:默认 default。在构建期间设置RUN指令的网络模式

实例

使用当前目录的 Dockerfile 创建镜像，标签为 my/ubuntu:v1。

docker build -t my/ubuntu:v1 .

## 7.18 history

docker history :查看指定镜像的创建历史。

语法

docker history [OPTIONS] IMAGE

OPTIONS说明：

* -H :以可读的格式打印镜像大小和日期，默认为true；
* --no-trunc :显示完整的提交记录；
* -q :仅列出提交记录ID。

实例

查看本地镜像runoob/ubuntu:v3的创建历史。

root@server:~# docker history my/ubuntu:v3

IMAGE CREATED CREATED BY SIZE COMMENT

4e3b13c8a266 3 months ago /bin/sh -c #(nop) CMD ["/bin/bash"] 0 B

<missing> 3 months ago /bin/sh -c sed -i 's/^#\s\*\(deb.\*universe\)$/ 1.863 kB

<missing> 3 months ago /bin/sh -c set -xe && echo '#!/bin/sh' > /u 701 B

<missing> 3 months ago /bin/sh -c #(nop) ADD file:43cb048516c6b80f22 136.3 MB

## 7.19 save

docker save :将指定镜像保存成 tar 归档文件。

语法

docker save [OPTIONS] IMAGE [IMAGE...]

OPTIONS 说明：

* -o :输出到的文件。

实例

将镜像 runoob/ubuntu:v3 生成 my\_ubuntu\_v3.tar 文档

root@server:~$ docker save -o my\_ubuntu\_v3.tar my/ubuntu:v3

root@server:~$ ll my\_ubuntu\_v3.tar

-rw------- 1 root root 142102016 Jul 11 01:37 my\_ubuntu\_v3.ta

## 7.20 load

docker load :导入使用[docker save]命令导出的镜像。

语法

docker load [OPTIONS]

OPTIONS 说明：

* --input , -i :指定导入的文件，代替 STDIN。
* --quiet , -q :精简输出信息。

实例

导入镜像：

$ docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

$ docker load < busybox.tar.gz

Loaded image: busybox:latest

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox latest 769b9341d937 7 weeks ago 2.489 MB

$ docker load --input fedora.tar

Loaded image: fedora:rawhide

Loaded image: fedora:20

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox latest 769b9341d937 7 weeks ago 2.489 MB

fedora rawhide 0d20aec6529d 7 weeks ago 387 MB

fedora 20 58394af37342 7 weeks ago 385.5 MB

fedora heisenbug 58394af37342 7 weeks ago 385.5 MB

fedora latest 58394af37342 7 weeks ago 385.5 MB

## 7.21 import

docker import :从归档文件中创建镜像。

语法

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

OPTIONS说明：

* -c :应用docker 指令创建镜像；
* -m :提交时的说明文字；

实例

从镜像归档文件my\_ubuntu\_v3.tar创建镜像，命名为my/ubuntu:v4

root@server:~$ docker import my\_ubuntu\_v3.tar my/ubuntu:v4

sha256:63ce4a6d6bc3fabb95dbd6c561404a309b7bdfc4e21c1d59fe9fe4299cbfea39

root@server:~$ docker images my/ubuntu:v4

REPOSITORY TAG IMAGE ID CREATED SIZE

my/ubuntu v4 63ce4a6d6bc3 20 seconds ago 142.1 MB