# Docker 的基本概念

## Docker是什么

Docker 最初是 dotCloud 公司创始人 Solomon Hykes 在法国期间发起的一个公司内部项目，它是基于 dotCloud 公司多年云服务技术的一次革新，并于 2013 年 3 月以 Apache 2.0 授权协议开源，主要项目代码在 GitHub 上进行维护。Docker 项目后来还加入了 Linux 基金会，并成立推动 开放容器联盟（OCI）。

Docker 使用 Google 公司推出的 **Go 语言** 进行开发实现，基于 Linux 内核的cgroup，

namespace，以及 AUFS 类的 Union FS 等技术，对进程进行封装隔离，属于操作系统层面的虚拟化技术。由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为**容器**。

Docker 在容器的基础上，进行了进一步的封装，从文件系统、网络互联到进程隔离等等，极

大的简化了容器的创建和维护。使得 **Docker 技术比虚拟机技术更为轻便、快捷**。

## 为什么 使用docker

Docker 的优势

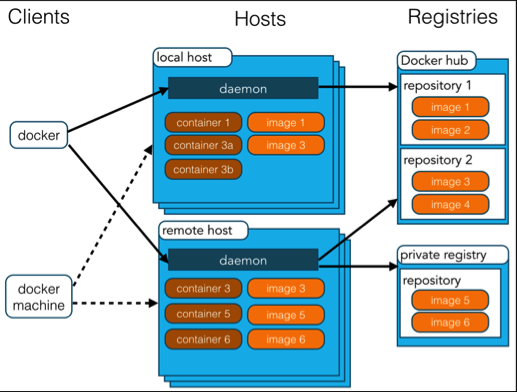
* 更高效的利用系统资源
* 更快速的启动时间
* 一致的运行环境
* 持续交付和部署
* 更轻松的迁移
* 更轻松的维护和扩展

对比传统虚拟机总结

|  |  |  |
| --- | --- | --- |
| 特性 | 容器 | 虚拟机 |
| 启动 | 秒级 | 分钟级 |
| 硬盘使用 | 一般为 MB | 一般为 GB |
| 性能 | 接近原生 | 较弱 |
| 系统支持量 | 单机支持上千个容器 | 一般几十个 |

## Docker 架构

Docker 使用客户端-服务器 (C/S) 架构模式，使用远程API来管理和创建 Docker容器。



Docker 基本概念

Docker 镜像 容器 仓库

### Docker 镜像

我们都知道，操作系统分为内核和用户空间。对于 Linux 而言，内核启动后，会挂载 root文件系统为其提供用户空间支持。而 Docker 镜像（Image），就相当于是一个 root 文件系统。比如官方镜像 centos:7.6 就包含了完整的一套 centos 7.6 最小系统的 root 文件系统。

Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

因为镜像包含操作系统完整的 root 文件系统，其体积往往是庞大的，因此在 Docker 设计时将其设计为分层存储的架构。镜像只是一个虚拟的概念，其实际体现并非由一个文件组成，而是由一组文件系统组成，或者说，由多层文件系统联合组成。

镜像构建时，会一层层构建，前一层是后一层的基础。每一层构建完就不会再发生改变，后一层上的任何改变只发生在自己这一层。在构建镜像的时候，需要额外小心，每一层尽量只包含该层需要添加的东西，任何额外的东西应该在该层构建结束前清理掉。

分层存储的特征还使得镜像的复用、定制变的更为容易。甚至可以用之前构建好的镜像作为基础层，然后进一步添加新的层，以定制自己所需的内容，构建新的镜像。

镜像（ Image ）和容器（ Container ）的关系，就像 Java 中的 类 和 实例一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等。

前面讲过镜像使用的是分层存储，容器也是如此。每一个容器运行时，是以镜像为基础层，在其上创建一个当前容器的存储层，我们可以称这个为容器运行时读写而准备的存储层为容器存储层。

容器存储层的生存周期和容器一样，容器消亡时，容器存储层也随之消亡。因此，任何保存于容器存储层的信息都会随容器删除而丢失。

### Docker 容器

镜像（ Image ）和容器（ Container ）的关系，就像 Java 中的 类 和 实例一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等。

前面讲过镜像使用的是分层存储，容器也是如此。每一个容器运行时，是以镜像为基础层，在其上创建一个当前容器的存储层，我们可以称这个为容器运行时读写而准备的存储层为容器存储层。

容器存储层的生存周期和容器一样，容器消亡时，容器存储层也随之消亡。因此，任何保存于容器存储层的信息都会随容器删除而丢失。

按照 Docker 最佳实践的要求，容器不应该向其存储层内写入任何数据，容器存储层要保持无状态化。所有的文件写入操作，都应该使用 Volume 数据卷、或者绑定宿主目录，在这些位置的读写会跳过容器存储层，直接对宿主（或网络存储）发生读写，其性能和稳定性更高。

数据卷的生存周期独立于容器，容器消亡，数据卷不会消亡。因此，使用数据卷后，容器删除或者重新运行之后，数据却不会丢失。

### Docker 仓库

镜像构建完成后，可以很容易的在当前宿主机上运行，但是，如果需要在其它服务器上使用这个镜像，我们就需要一个集中的存储、分发镜像的服务，Docker Registry 就是这样的服务。

一个 Docker Registry 中可以包含多个仓库（ Repository ）；每个仓库可以包含多个标签（ Tag ）；每个标签对应一个镜像。

通常，一个仓库会包含同一个软件不同版本的镜像，而标签就常用于对应该软件的各个版本。我们可以通过 <仓库名>:<标签> 的格式来指定具体是这个软件哪个版本的镜像。如果不给出标签，将以 latest 作为默认标签。

以 centos 镜像 为例， centos 是仓库的名字，其内包含有不同的版本标签，如， 6.9 、7.5 。我们可以通过 centos:6.9 ，或者 centos:7.5 来具体指定所需哪个版本的镜像。如果忽略了标签，比如 centos ，那将视为 centos:latest 。

仓库名经常以 两段式路径 形式出现，比如 study/nginx，前者往往意味着 Docker Registry 多用户环境下的用户名，后者则往往是对应的软件名。但这并非绝对，取决于所使用的具体 Docker

Registry 的软件或服务。

公开仓库

常用的 Registry 是官方的 Docker Hub，这也是默认的 Registry。除此以外，还有 CoreOS 的 Quay.io，CoreOS 相关的镜像存储在这里；Google 的 Google Container Registry，Kubernetes 的镜像使用的就是这个服务。

国内的一些云服务商提供了针对 Docker Hub 的镜像服务，这些镜像服务被称为加速器。常见的有 阿里云加速器、DaoCloud 加速器 等。使用加速器会直接从国内的地址下载 Docker Hub 的镜像，比直接从 Docker Hub 下载速度会提高很多。

国内也有一些云服务商提供类似于 Docker Hub 的公开服务。比如 网易云镜像服务、

DaoCloud 镜像市场、阿里云镜像库 等。

这一章，我们详细的介绍了 Docker 及为什么要使用 Docker，Docker的优势等，

并讲解了 Docker 的三个基本概念 镜像、容器、仓库及 Docker 的生命周期。

通过这一章节的学习，相信大家应该对 Docker 有了一个比较深刻的认识，接下来我们就要去实践操作 Docker。

# Docker 安装

## Docker 的版本和命名

Docker 在 1.13 版本之后，从 2017 年的 3 月 1 日开始，版本命名规则变为如下

|  |  |
| --- | --- |
| 项目 | 说明 |
| 版本格式 | YY.MM |
| Stable 版本 | 每个季度发行 |
| Edge 版本 | 每个月发行 |
| 当前 Docker CE Stable 版本 | 17.09 |
| 当前 Docker CE Edge 版本 | 17.11 |

同时 Docker 划分为 CE 和 EE。CE 即社区版（免费，支持周期三个月），EE 即企业版，

强调安全，付费使用。

## Docker 安装

官方网站上有各种环境下的 安装指南，这里主要介绍 Docker CE 在 Linux 、Windows 10 和 MacOS 上的安装。

官方安装指南地址：https://docs.docker.com/engine/installation/

Docker CE 支持 64 位版本 CentOS 7，并且要求内核版本不低于 3.10。

**卸载旧的版本**

旧版本的 Docker 称为 docker 或者 docker-engine ，使用以下命令卸载旧版本：

使用yum 安装

sudo yum install docker-ce

**使用脚本安装**

在测试或开发环境中 Docker 官方为了简化安装流程，提供了一套便捷的安装脚本，CentOS系统上可以使用这套脚本安装：

curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh --mirror Aliyun

执行这个命令后，脚本就会自动的将一切准备工作做好，并且把 Docker CE 的 Edge 版本安

启动 docker

sudo systemctl enable docker #设置开启启动

sudo systemctl start docker

**卸载docker**

首先搜索已经安装的docker 安装包

yum list installed|grep docker

或者使用该命令：rpm -qa|grep docker

分别删除安装包：

yum –y remove docker.x86\_64

yum –y remove docker-client.x86\_64

yum –y remove docker-common.x86\_64

删除docker 镜像

rm -rf /var/lib/docker

#启动 Docker 后台服务

sudo systemctl start docker

#测试运行 hello-world,由于本地没有hello-world这个镜像，所以会下载一个hello-world的镜像，并在容器内运行。

docker run hello-world

## docker的启动 关闭

docker 常用的命令

[root@localhost ~]# service docker

The service command supports only basic LSB actions (start, stop, restart, try-restart, reload, force-reload, status). For other actions, please try to use systemctl.

使用的命令 service docker + 下面的命令

start 启动

stop 关闭

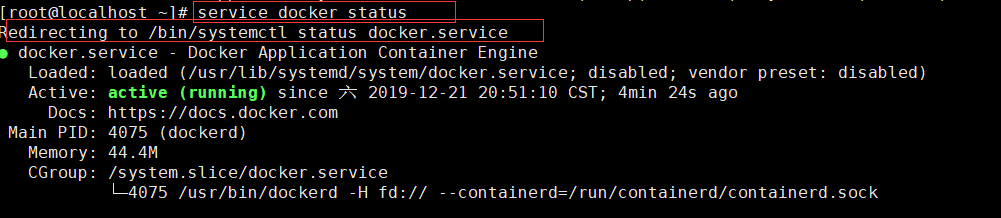
restart 重启

reload 重新加载

force-reload 强制 加载

status 状态

举例查询 docker 的状态



执行

service docker status

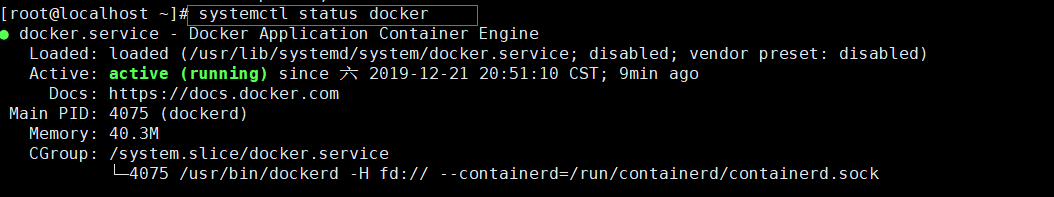
重定向到

/bin/systemctl status docker.service

上面的命令可以简化到

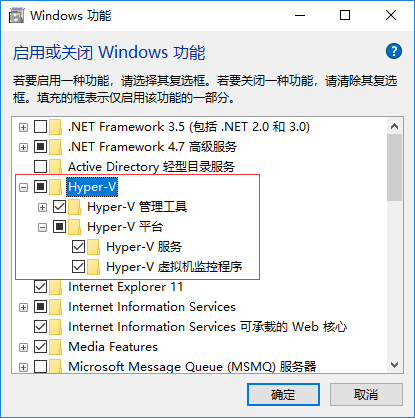
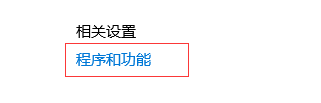
systemctl status docker

同样 status 这个 换成 start restart 这些命令



## Windows 安装docker

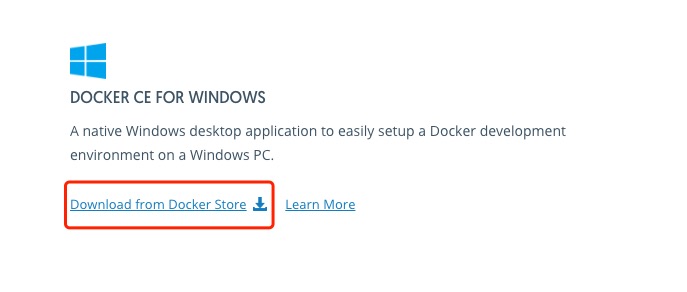
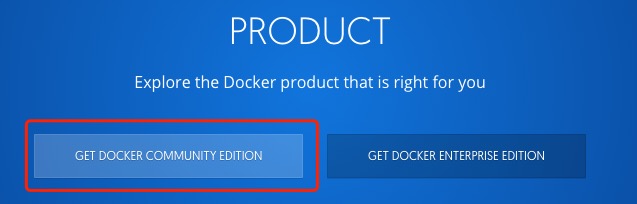
1. 开启 Hyper-V



1. 安装 Toolbox

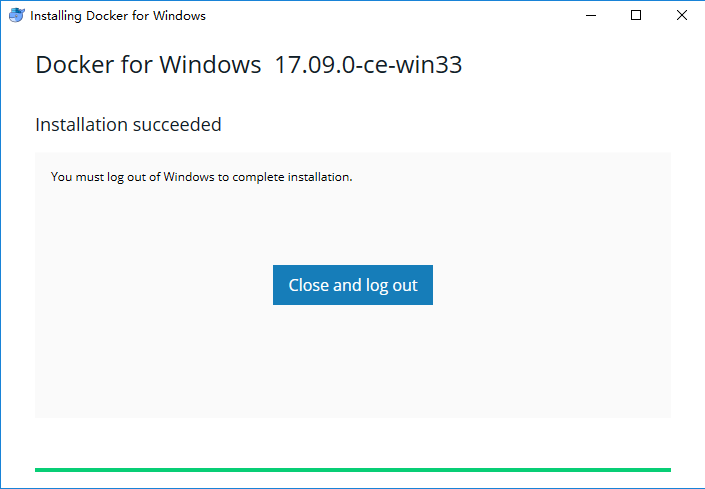
最新版 Toolbox 下载地址：  https://www.docker.com/get-docker

点击 Get Docker Community Edition，并下载 Windows 的版本



1. 安装docker

双击下载的 Docker for Windows Installe 安装文件，一路 Next，点击 Finish 完成安装。



安装完成后，Docker 会自动启动。通知栏上会出现个小鲸鱼的图标，我们可以在命令行执行 docker version 来查看版本号，docker run hello-world 来载入测试镜像测试

如果没启动，你可以在 Windows

搜索 Docker 来启动：

启动后，也可以在通知栏上看到小鲸鱼图标：

http://www.runoob.com/wp-content/uploads/2017/12/1513585123-3777-whale-taskbar-circle.png

## MacOS 安装 Docker

1. 系统要求

Docker for Mac 要求系统最低为 macOS 10.10.3 Yosemite。如果系统不满足需求，可以安装

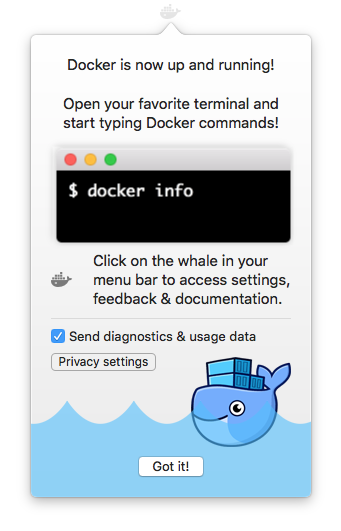
2，使用 Homebrew 安装

使用 Homebrew 安装

Homebrew 的 Cask 已经支持 Docker for Mac，因此可以很方便的使用 Homebrew Cask 来进行安装

b11rew cask install docker

3. 运行



第一次点击图标，可能会看到这个安装成功的界面，点击 "Got it!" 可以关闭这个窗口。

安装docker

启动终端后，通过命令可以检查安装后的 Docker 版本

docker --version #查看 docker 版本

docker-compose --version #查看 compose 版本

docker-machine --version #查看 machine 版本

如果 docker version，可以尝试运行一个 hello-world 服务器

docker run hello-world #启动一个基于hello-world镜像的容器

# docker 加速

国内从 Docker Hub 拉取镜像有时会遇到困难，此时可以配置镜像加速器。Docker 官方和国内很多云服务商都提供了国内加速器服务，例如：

* Docker 官方提供的中国 registry mirror
* 阿里云加速器
* DaoCloud 加速器
* 163 加速器

## Linux 加速

对于使用 systemd 的系统，请在 /etc/docker/daemon.json 中写入如下内容（如果文件不存在请新建该文件）

请在该配置文件中加入（没有该文件，请创建一个文件）；

{

"registry-mirrors": ["https://docker.mirrors.ustc.edu.cn"]

}

配置多个

{

"registry-mirrors" : [

"http://hub-mirror.c.163.com",

"https://docker.mirrors.ustc.edu.cn"

]

}

重启生效

sudo systemctl daemon-reload

sudo systemctl restart docker

从网上找了几个速度比较快的镜像地址：

Docker 官方中国区

https://registry.docker-cn.com

网易

http://hub-mirror.c.163.com

ustc

https://docker.mirrors.ustc.edu.cn

最后，需要重启docker服务

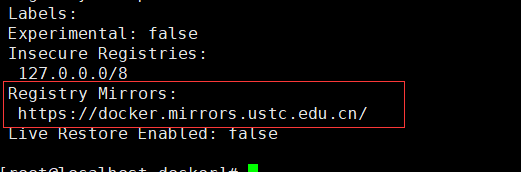
systemctl start docker.service

之后再使用pull 命令拉去镜像，这时候是从ustc 获取镜像。而且速度很快；

检测加速是否生效

docker info

在最后可以看到 加速的镜像地址



## Windows 10 加速

对于使用 Windows 10 的系统，在系统右下角托盘 Docker 图标内右键菜单选择 Settings ，打开配置窗口后左侧导航菜单选择 Docker Daemon 。编辑窗口内的 JSON 串，填写加速器地址，如：

{

"registry-mirrors": [

"http://hub-mirror.c.163.com"

]

}

编辑完成，点击 Apply 保存后 Docker 服务会重新启动

## MacOS 加速

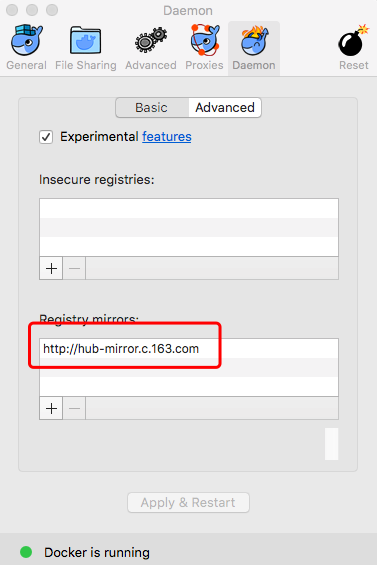
对于使用 macOS 的用户

在任务栏点击 Docker for mac 应用图标 -> Perferences... -> Daemon -> Registry mirrors

在列表中填写加速器地址即可。

修改完成之后，点击 Apply &Restart 按钮，

Docker 就会重启并应用配置的镜像地址了。

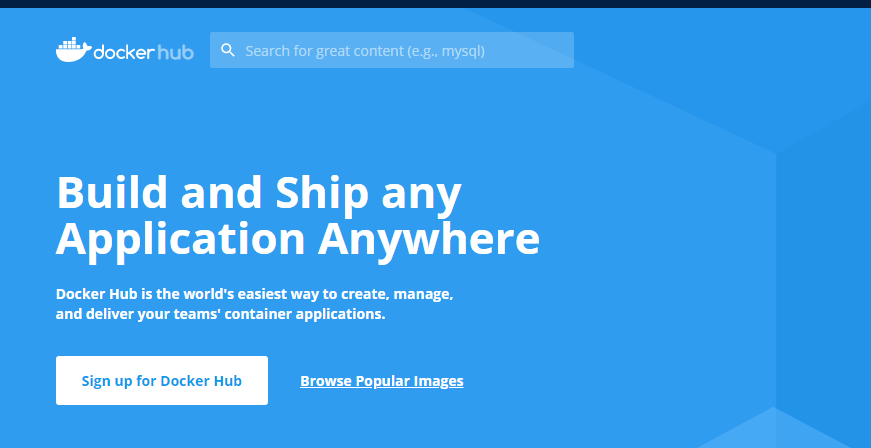


# Docker 镜像的操作

Docker 运行容器前需要本地存在对应的镜像，如果本地不存在该镜像，Docker 会从镜像仓库下载该镜像。

Docker 官网

<https://hub.docker.com/>



接下来将介绍关于镜像的内容，包括：

1. 从仓库获取镜像；
2. 管理本地主机上的镜像；
3. 介绍镜像实现的基本原理。

## 对镜像的操作

**获取镜像**

从 Docker 镜像仓库获取镜像的命令是 docker pull 。其命令格式为：

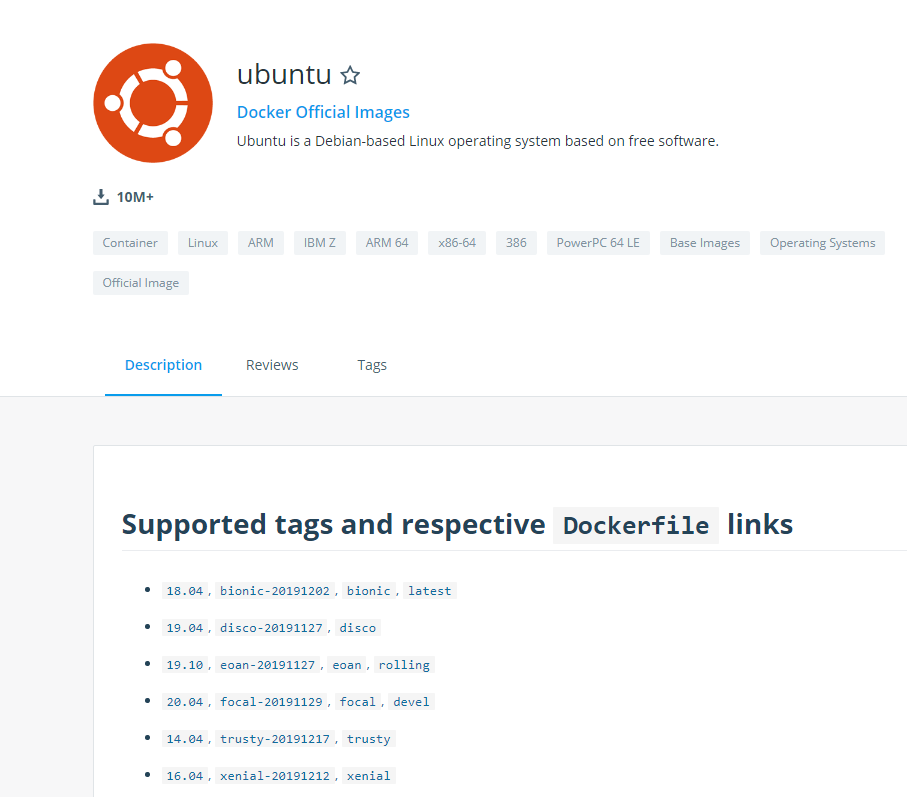
docker pull [选项] [Docker Registry 地址[:端口号]/]仓库名[:标签]

具体的选项可以通过 docker pull --help 命令看到，这里我们说一下镜像名称的格式。

Docker 镜像仓库地址：地址的格式一般是 <域名/IP>[:端口号] 。默认地址是 Docker Hub。

仓库名：如之前所说，这里的仓库名是两段式名称，即 <用户名>/<软件名> 。对于 Docker Hub，如果不给出用户名，则默认为 library ，也就是官方镜像。

上面的命令中没有给出 Docker 镜像仓库地址，因此将会从 Docker Hub 获取镜像。而镜像名称是 ubuntu:16.04 ，因此将会获取官方镜像 library/ubuntu 仓库中标签为 16.04 的镜像。



举例 ：

docker pull ubuntu:16.04

**查看镜像列表**

docker images

[root@localhost ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

centos centos7 5e35e350aded 7 weeks ago 203MB

选择 镜像的两种方法

通过镜像 可以启动一个容器。

选择一个容易的办法 有两种 通过 rep : tag 方式 或者是 IMAGE ID 方式 ，这两种

REPOSITORY + TAG

IMAGE

要想列出已经下载下来的镜像，可以使用 docker image ls 命令。列表包含了 仓库名 、 标签 、 镜像 ID 、 创建时间 以及 所占用的空间 。

docker image ls

查看镜像、容器、数据卷所占用的空间。

docker system df

仓库名、标签均为 <none> 的镜像称为 虚悬镜像(dangling image) ，显示这类镜像：

docker image ls -f dangling=true

一般来说，虚悬镜像已经失去了存在的价值，是可以随意删除的，可以用下面的命令删除。

docker image prune

**删除镜像**

如果要删除本地的镜像，可以使用 docker image rm 命令，其格式为：

docker image rm [选项] <镜像1> [<镜像2> ...]

其中， <镜像> 可以是 镜像短 ID 、 镜像长 ID 、 镜像名 或者 镜像摘要 。

使用 docker image ls -q 来配合 docker image rm ，这样可以批量删除希望删除的镜像。

docker image rm $(docker image ls -q ubuntu) #删除所有仓库名为 ubuntu 的镜像

或者删除所有在 ubuntu:16.04 之前的镜像：

docker image rm $(docker image ls -q -f before=ubuntu:16.04)

## 容器的操作

容器是独立运行的一个或一组应用，以及它们的运行态环境。对应的，虚拟机可以理解为模拟运行的一整套操作系统（提供了运行态环境和其他系统环境）和跑在上面的应用。

接下来将具体介绍如何来管理一个容器，包括创建、启动和停止等。

**容器的启动**

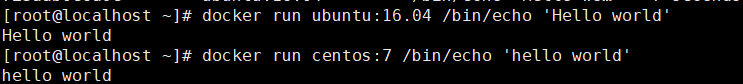
启动容器有两种方式，一种是基于镜像新建一个容器并启动，另外一个是将在终止状态

（ stopped ）的容器重新启动。

因为 Docker 的容器实是轻量级的，用户可以随时删除和新创建容器。

输出一个 “Hello World”，之后终止容器。

docker run ubuntu:16.04 /bin/echo 'Hello world'



**启动 已经终止的容易**

docker container start 或者 docker start

启动一个 bash 终端，允许用户进行交互。

docker run -t -i ubuntu:16.04 /bin/bash

**-t** 让Docker分配一个伪终端并绑定到容器的标准输入上，

**-i** 则让容器的标准输入保持打开。

当利用 docker run 来创建容器时，Docker 在后台运行的标准操作包括：

检查本地是否存在指定的镜像，不存在就从公有仓库下载

利用镜像创建并启动一个容器

分配一个文件系统，并在只读的镜像层外面挂载一层可读写层

从宿主主机配置的网桥接口中桥接一个虚拟接口到容器中去

从地址池配置一个 ip 地址给容器

执行用户指定的应用程序

执行完毕后容器被终止

**后台运行**

很多时候，需要让 Docker 在后台运行而不是直接把执行命令的结果输出在当前宿主机下。

此时，可以通过添加 -d 参数来实现。

如果不使用 -d 参数运行容器，比如 docker run hello-world 会把日志打印在控制台；

如果使用了 -d 参数运行容器，比如 docker run -d hello-world 不会输出日志，只会打印容器id(输出结果可以用 docker logs 查看)；

注： 容器是否会长久运行，是和 docker run 指定的命令有关，和 -d 参数无关。

**容器的关闭重启**

可以使用 docker container stop 来终止一个运行中的容器。

终止状态的容器可以用 docker container ls -a 命令看到。

处于终止状态的容器，可以通过 docker container start 命令来重新启动。

此外， docker container restart 命令会将一个运行态的容器终止，然后再重新启动它

## 导入导出镜像

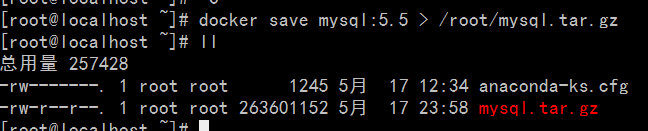
导入镜像:docker save 镜像 > /root/xx.tar.gz

镜像表示的 仓库 + 版本号

或者是 id

例如：需要时间。导出

docker save mysql:5.5 > /root/mysql.tar.gz



这样就导出了mysql 的镜像了

相反的过程

docker load < /root/mysql.tar.gz

# docker 容器的操作

## docker 启动

docker 的启动方式，常用的有三种启动方式 ，容器启动 守护线程 交互方式

这样就能启动一个一直停留在后台运行的Centos了。如果少了/bin/bash的话，Docker会生成一个Container但是马上就停止了，不会一致运行即使有了-d参数。

### 启动容器：

以交互的方式启动容器：docker run -it --name 容器的名称，镜像

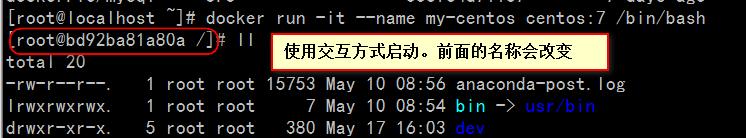
/bin/bash

以守护进程的方式启动容器：docker run -d –name 容器的名称镜像

### 交互方式启动

使用容器启动centeros 7

docker run -it --name my-centos centos:7 /bin/bash



退出命令：exit

### 守护线程启动

启动命令

注意：启动的名称不可以一致

docker run -d --name my-centos2 centos:7



表示启动成功

此时的依然是使用宿主启动命令

## 容器的停止

语法

docker stop 启动的名称

比如上面的

## 容器的删除

删除一个已经停止的；（后面是名称或者是id）

docker rm my-centos

## 容器的交互

**进入交互**

上文有 交互 式启动容器，启动后 进去容器中

docker exec -it dev\_tomcat /bin/bash

红色表示容器的名称。根据实际情况进行改变

* **docker run：启动container**
* **ubuntu：你想要启动的image**
* **-t：进入终端**
* **-i：获得一个交互式的连接，通过获取container的输入**
* **/bin/bash：在container中启动一个bash shell**

（资料来源： <https://blog.csdn.net/dongdong9223/article/details/52998375> ）



进入交互以后，界面就改变了，不是直接的命令行

查看当前目录pwd

目录下的文件ls　没有ll

退出exit

**文件的复制**

docker cp game.html dev\_tomcat:/usr/local/tomcat/webapps/ROOT

把指定的文件拷贝到指定的位置

docker cp 文件的名称 docker容器 冒号 路径

## 容器的保存成镜像

语法

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

OPTIONS说明：

* **-a :**提交的镜像作者；
* **-c :**使用Dockerfile指令来创建镜像；
* **-m :**提交时的说明文字；
* **-p :**在commit时，将容器暂停。

将容器a404c6c174a2 保存为新的镜像,并添加提交人信息和说明信息。

runoob@runoob:~$ docker commit -a "runoob.com" -m "my apache" a404c6c174a2 mymysql:v1

sha256:37af1236adef1544e8886be23010b66577647a40bc02c0885a6600b33ee28057

runoob@runoob:~$ docker images mymysql:v1

REPOSITORY TAG IMAGE ID CREATED SIZE

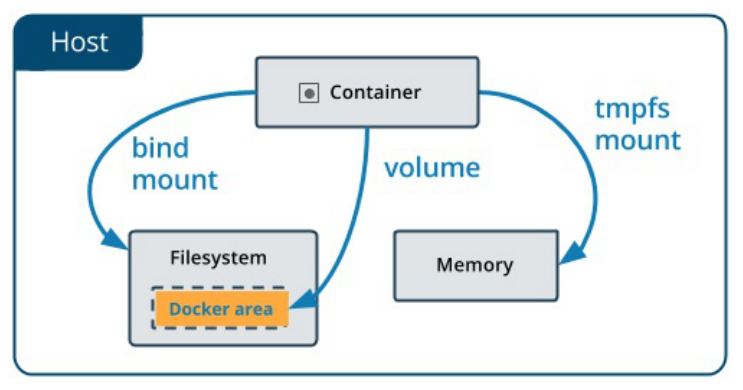
mymysql v1 37af1236adef 15 seconds ago 329 MB

# docker 数据的挂载

## docker的数据管理

在容器中管理数据主要有两种方式：

* 数据卷（Volumes）
* 挂载主机目录 (Bind mounts)



-V

-mount

数据卷是一个可供一个或多个容器使用的特殊目录，它绕过 UFS，可以提供很多有用的特性：

数据卷可以在容器之间共享和重用

对数据卷的修改会立马生效

对数据卷的更新，不会影响镜像

数据卷默认会一直存在，即使容器被删除

注意： 数据卷的使用，类似于Linux下对目录或文件进行mount，镜像中的被指定为挂载点的目录中的文件会隐藏掉，能显示看的是挂载的数据卷 。

**数据卷**

Docker 新用户应该选择 --mount 参数

经验丰富的Docker使用者对 -v 或者 --volume已经很熟悉了，但是推荐使用 –mount 参数。

使用docker volume 查看相关的提示命令

[root@localhost ~]# docker volume

Usage: docker volume COMMAND

Manage volumes

Commands:

create Create a volume

inspect Display detailed information on one or more volumes

ls List volumes

prune Remove all unused local volumes

rm Remove one or more volumes

Run 'docker volume COMMAND --help' for more information on a command.

创建数据卷

docker volume create my-volume

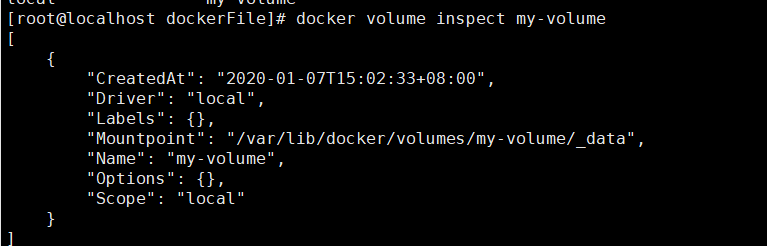
查看数据卷

docker volume ls

查看指定数据卷的信息

docker volume inspect my-volume

命令执行的效果



启动一个挂载数据卷的容器：

在用 docker run 命令的时候，使用 --mount 标记来将 数据卷 挂载到容器里。在一次 docker run 中可以挂载多个 数据卷 。

创建一个名为 session-web 的容器，并加载一个 数据卷 到容器的 /webapp 目录。

docker run --name session-web -d -p 8888:8080 \

# -v my-volume:/wepapp \

--mount type=bind,source=/src/webapp,target=/opt/webapp

session-web:latest

删除数据卷

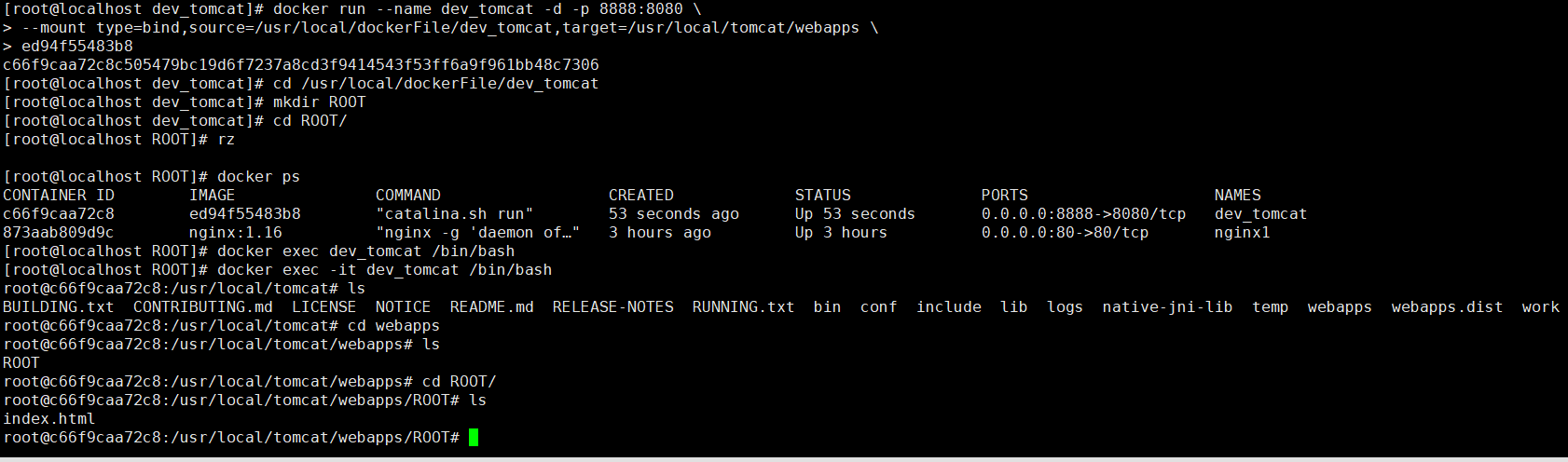
docker volume rm my-volume

数据卷是被设计用来持久化数据的，它的生命周期独立于容器，Docker不会在容器被删除后自动删除数据卷 ，并且也不存在垃圾回收这样的机制来处理没有任何容器引用的数据卷。

如果需要在删除容器的同时移除数据卷。可以在删除容器的时候使用docker rm –v 这个命令。

无主的数据卷可能会占据很多空间，要清理请使用以下命令

docker volume prune



之前文档原文

docker run --name session-web -d -p 8888:8080 \

# -v my-volume:/wepapp \

--mount type=bind,source=/src/webapp,target=/opt/webapp

session-web:latest

上面的命令加载主机的 /src/webapp 目录到容器的 /opt/webapp 目录。这个功能在进行测试的时候十分方便，比如用户可以放置一些程序到本地目录中，来查看容器是否正常工作。

进行挂载的命令

docker run --name dev\_tomcat -d -p 8888:8080 \

--mount type=bind,source=/usr/local/dockerFile/dev\_tomcat,target=/usr/local/tomcat/webapps \

ed94f55483b8

source : 后面是 文件的路径

target : 容器内的路径

本地目录的路径必须是绝对路径

以前------使用 -v 参数时如果本地目录不存在 Docker 会自动为你创建一个文件夹

现在------使用 --mount 参数时如果本地目录不存在，Docker 会报错

Docker 挂载主机目录的默认权限是读写 ，用户也可以通过增加 readonly 指定为只读

**推荐使用 mount 命令**

# Docker 网络

## Docker 网络

使用 docker network 命令 ，进行查看 命令

[root@localhost ~]# docker network

Usage: docker network COMMAND

Manage networks

Commands:

connect Connect a container to a network

create Create a network

disconnect Disconnect a container from a network

inspect Display detailed information on one or more networks

ls List networks

prune Remove all unused networks

rm Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.

通过使用 ls 命令来查看

[root@localhost ~]# docker network ls

NETWORK ID NAME DRIVER SCOPE

329c01b134d0 bridge bridge local

ea7daffa4094 host host local

577ffc70a304 none null local

## Docker 网络 模式

<https://www.cnblogs.com/zuxing/articles/8780661.html>

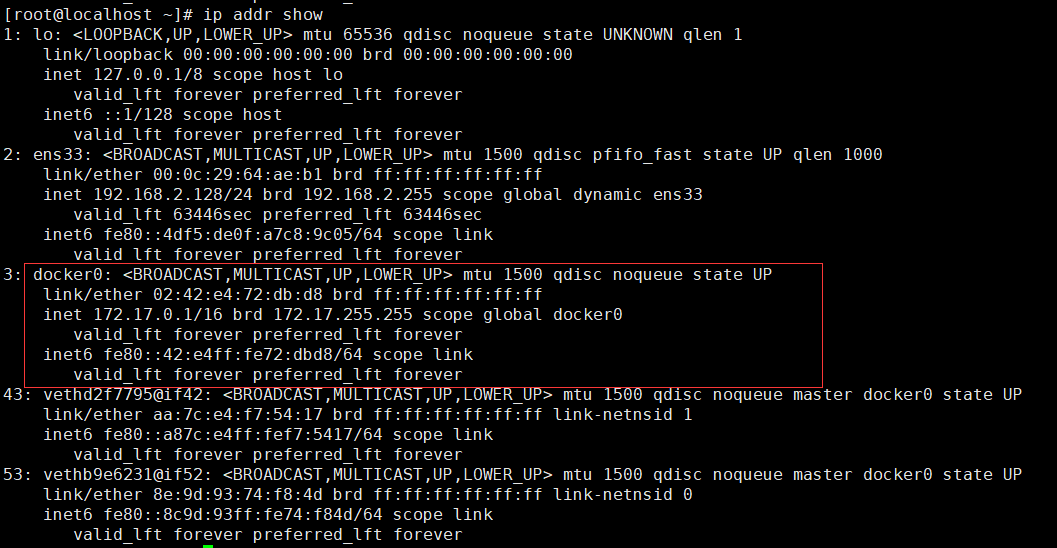
Docker 网络 有四种

none 无

host 仅主机

container

bridge 侨联



查看具体的网络使用情况

ip addr show

通过上面的命令进行查看网络的

出现docker0 这个网络，这个网络 在 172.17.0.1/16 子网内

# Docker 查看日志

Docker 查看日志

资料来源： <https://blog.csdn.net/wen_1108/article/details/78356655>

实时查看docker容器日志

$ sudo docker logs -f -t --tail 行数 容器名

docker logs -f -t --tail 100 dev\_tomcat

例：实时查看docker容器名为s12的最后10行日志

$ sudo docker logs -f -t --tail 10 s12

docker logs -f -t --tail 30 dev\_tomcat

# Dockerfile

# Docker-compose

Dokcer-compose 安装

sudo curl -L https://github.com/docker/compose/releases/download/1.17.1/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

注： x86\_64 架构的 Linux 建议按照上边的方法下载二进制包进行安装，如果您计算机的架构是 ARM (例如，树莓派)，再使用 pip 安装。

这种方式是将 Compose 当作一个 Python 应用来从 pip 源中安装。

执行安装命令：

sudo pip install -U docker-compose

可以看到类似如下输出，说明安装成功。

Collecting docker-compose

Downloading docker-compose-1.17.1.tar.gz (149kB): 149kB downloaded

...

Successfully installed docker-compose cached-property requests texttable websocket-client docker-py dockerpty six enum34 backports.ssl-match-hostname ipaddress

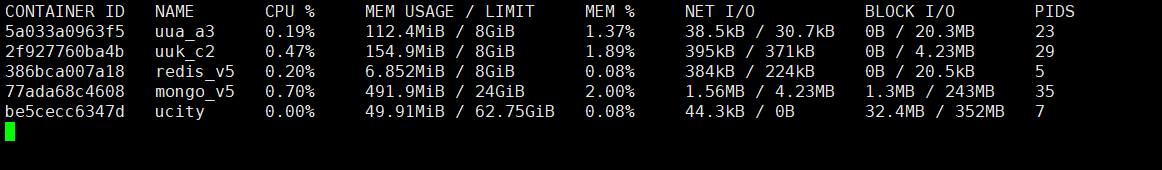
# Docker 容器监控

## Docker 自带的查看

资料来源： <https://www.cnblogs.com/sparkdev/p/7821376.html>

## 默认输出

docker stats

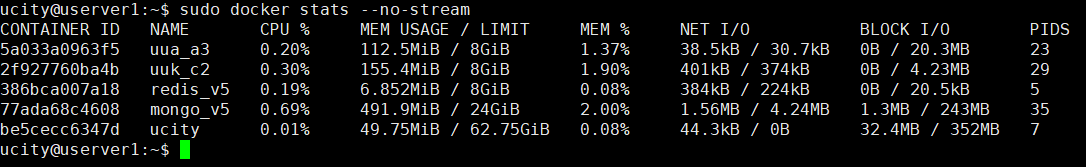


默认情况下，stats 命令会每隔 1 秒钟刷新一次输出的内容直到你按下 ctrl + c。下面是输出的主要内容：  
[**CONTAINER**]：以短格式显示容器的 ID。  
[**CPU %**]：CPU 的使用情况。  
[**MEM USAGE / LIMIT**]：当前使用的内存和最大可以使用的内存。  
[**MEM %**]：以百分比的形式显示内存使用情况。  
[**NET I/O**]：网络 I/O 数据。  
[**BLOCK I/O**]：磁盘 I/O 数据。  
[**PIDS**]：PID 号。

## 只返回当前的状态

如果不想持续的监控容器使用资源的情况，可以通过 --no-stream 选项只输出当前的状态：

docker stats --no-stream



这样输出的结果就不会变化了，看起来省劲不少。

## 只输出指定的容器

sudo docker stats 5a033a0963f5

增加容器的 id 查看单独的

## 格式化输出

我们在前面搞了点小手段把输出中的容器 ID 替换成了名称。其实 docker stats 命令支持我们通过 --format 选项自定义输出的内容和格式：

sudo docker stats --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}"

上面的命令中我们只输出了 Name, CPUPerc 和 Memusage 三列。下面是自定义的格式中可以使用的所有占位符：  
.Container    根据用户指定的名称显示容器的名称或 ID。  
.Name           容器名称。  
.ID                 容器 ID。  
.CPUPerc       CPU 使用率。  
.MemUsage  内存使用量。  
.NetIO           网络 I/O。         
.BlockIO        磁盘 I/O。  
.MemPerc     内存使用率。  
.PIDs             PID 号。

通过 docker stats 命令我们可以看到容器使用系统资源的情况。这为我们进一步的约束容器可用资源或者是调查与资源相关的问题提供了依据。除了 docker 自带的命令，像 glances 等工具也已经支持查看容器使用的资源情况了，有兴趣的朋友可以去了解一下。