# 官网文档

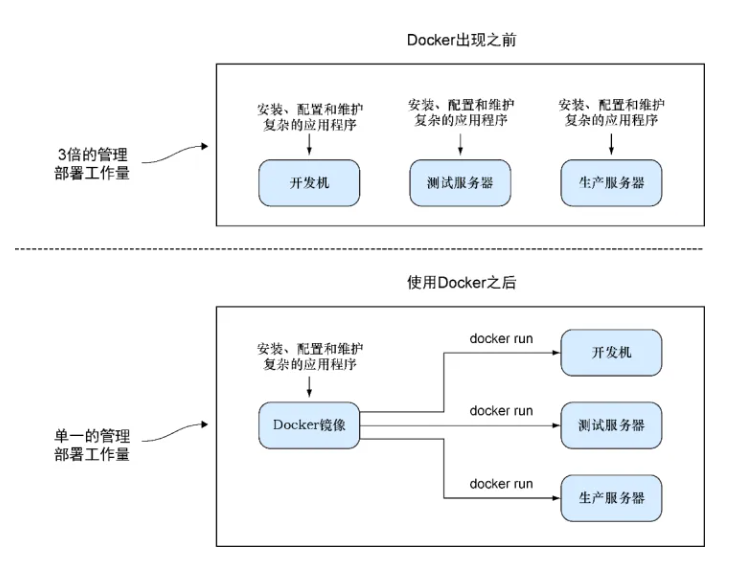
https://docs.docker.com/

# docker hub仓库地址

https://hub.docker.com/

https://hub.docker.com/repositories/jinshihui

# 基本概念



# 优点

# windows上安装docker

https://www.runoob.com/docker/windows-docker-install.html

# Centos上安装docker

## 基本步骤

### 1、使用ROOT权限登录CentOS，并将[yum](https://so.csdn.net/so/search?q=yum&spm=1001.2101.3001.7020)包更新到最新。

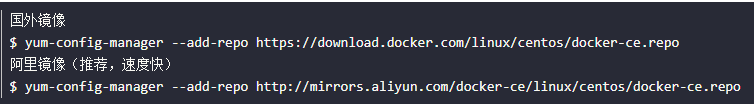
yum update

### 2、安装需要的软件包

yum-utils 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的。

yum install -y yum-utils device-mapper-persistent-data lvm2

### 3、设置yum源



yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

### 4、安装docker

执行一下命令，安装最新版本的 Docker Engine-Community 和 containerd。

sudo yum install -y docker-ce docker-ce-cli containerd.io

### 5、启动并加入开机启动

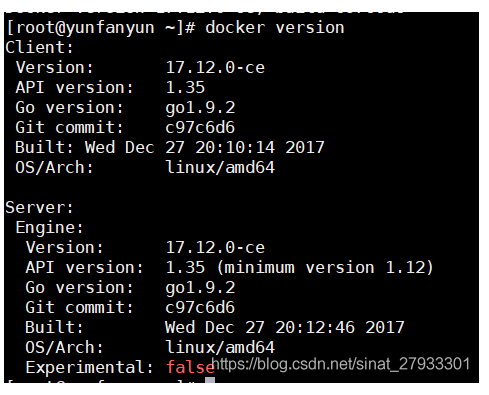
sudo systemctl start docker

sudo systemctl enable docker

### 6、验证安装是否成功

(有client和service两部分表示docker安装启动都成功了)

docker version



## 启动docker命令

systemctl start docker

## 查看docker的状态

systemctl status docker

# docker官方镜像仓库地址

https://hub.docker.com/

# docker配置阿里云镜像仓库

## 1、配置仓库地址

/etc/docker/daemon.json文件中添加aliyun镜像仓库地址（国内用阿里云，不然只能翻墙用国外的）

添加内容：

{

"registry-mirrors":[

"https://72idtxd8.mirror.aliyuncs.com"

],

"insecure-registries":[

"服务器ip:5000"

]

}

## 2、重启daemon.json文件

sudo systemctl daemon-reload

# docker客户端工具

使用 Portainer

# 将本地镜像上传到docker hub

# 容器rootfs

## 基本概念

容器 rootfs 是容器中用于挂载的根文件系统（Root File System）的概念。它是容器运行时环境中的一个重要组成部分，提供容器所需的文件和目录结构。

在容器技术中，每个容器都有自己的独立文件系统。这个文件系统是基于主机操作系统的某个目录或文件创建的。容器 rootfs 是指将此目录或文件视为容器的根文件系统，并在其上进行文件操作。容器中的所有文件和目录都从这个 rootfs 开始。

## 使用方法

1、创建容器 rootfs：创建容器时，可以通过不同的方式设置容器的根文件系统。常见的方式包括使用镜像作为根文件系统、利用模板创建根文件系统、或者手动创建一个根文件系统。

2、挂载容器 rootfs：在容器启动时，容器运行时会将根文件系统挂载到容器的文件系统层级结构中。这使得容器内部的应用程序可以访问和操作这个根文件系统上的文件和目录。

3、容器镜像和容器 rootfs 的关系：容器镜像是用于创建容器的模板，其中包含了初始化容器的基本文件系统结构。容器启动时，容器镜像的根文件系统会被挂载为容器 rootfs。

4、容器的隔离性：每个容器拥有自己的根文件系统，这使得容器之间的文件和目录相互隔离，避免了相互干扰。容器内部可以独立安装软件、创建文件和目录，而不会影响其他容器或主机系统。

# docker网络类型

## 基本概念

Docker的网络有四种类型,分别为host模式、none模式、container模式、bridge模式。

## host模式

创建容器时，可以使用"-net=host"指定。

启动的容器如果指定了网络类型为host模式，那么新创建的容器不会创建自己的虚拟网卡,而是直接使用宿主机的网卡和IP地址,因此容器里面查看到的IP信息就是宿主机的信息，访问容器的时候直接使用"宿主机IP:容器端口"即可,不过容器的其它资源们比如文件系统，系统进程等还是和宿主机保持隔离。

此模式的网络性能最高,但是各容器之间端口不能相同(因为该模式直接使用的是宿主机的网络)，适用于运行容器端口比较固定的业务，比如Mysql,Redis等。

## none模式

创建容器时，可以使用"-net=none"指定。

在使用none模式后,docker容器不会进行任何网络配置，其没有网卡，没有IP也没有路由，因此默认无法与外界通信，需要手动添加配置IP等，所以极少使用。

# Dockerfile

## 基本概念

Dockerfile 是一个用来构建镜像的文本文件，文本内容包含了一条条构建镜像所需的指令和说明。

## 官方文档

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

## 指令

https://www.runoob.com/docker/docker-dockerfile.html

### from

构建镜像基于哪个镜像。

### MAINTAINER

镜像维护者姓名或邮箱地址。

### VOLUME

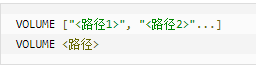
定义匿名数据卷。在启动容器时忘记挂载数据卷，会自动挂载到匿名卷。

作用：

1、避免重要的数据因容器重启而丢失，这是非常致命的。

2、避免容器不断变大。

格式：



在启动容器 docker run 的时候，我们可以通过 -v 参数修改挂载点。

### RUN

构建镜像时运行的指令。

### CMD

运行容器时执行的shell环境。类似于 RUN 指令，用于运行程序，但二者运行的时间点不同:

1、CMD在docker run时运行。

2、RUN是在docker build时运行。

**作用**：为启动的容器指定默认要运行的程序，程序运行结束，容器也就结束。CMD 指令指定的程序可被 docker run 命令行参数中指定要运行的程序所覆盖。

**注意**：如果 Dockerfile 中存在多个 CMD 指令，仅最后一个生效。

格式：



推荐使用第二种格式，执行过程比较明确。第一种格式实际上在运行的过程中也会自动转换成第二种格式运行，并且默认可执行文件是 sh。

### WORKDIR

指定工作目录。用 WORKDIR 指定的工作目录，会在构建镜像的每一层中都存在。以后各层的当前目录就被改为指定的目录，如该目录不存在，WORKDIR 会帮你建立目录。docker build 构建镜像过程中的，每一个 RUN 命令都是新建的一层。只有通过 WORKDIR 创建的目录才会一直存在。

为 RUN、CMD、ENTRYPOINT、COPY 和 ADD 设置工作目录，就是切换目录。

格式：

WORKDIR <工作目录路径>

### COPY

拷贝文件或目录到容器中，跟ADD类似，但不具备自动下载或解压的功能。

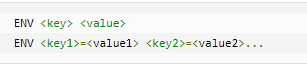
### ADD

拷贝文件或目录到容器中，如果是URL或压缩包便会自动下载或自动解压。

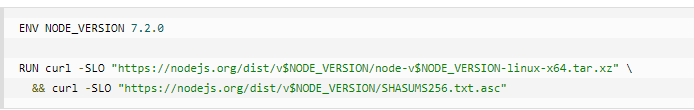
### ENV

设置环境变量，定义了环境变量，那么在后续的指令中，就可以使用这个环境变量。

格式：



以下示例设置 NODE\_VERSION = 7.2.0 ， 在后续的指令中可以通过 $NODE\_VERSION 引用：



### ENTRYPOINT

运行容器时执行的shell命令。

# Docker Compose

## 基本概念

Compose 是用于定义和运行多容器 Docker 应用程序的工具。通过 Compose，您可以使用 YML 文件来配置应用程序需要的所有服务，然后使用一个命令，就可以从 YML 文件配置中创建并启动所有服务。

## 使用的步骤

1、使用 Dockerfile 定义应用程序的环境。

2、使用 docker-compose.yml 定义构成应用程序的服务，这样它们可以在隔离环境中一起运行。

3、最后执行 docker-compose up 命令来启动并运行整个应用程序。

## 安装方法

### Linux操作系统

Linux 上我们可以从 Github 上下载它的二进制包来使用，最新发行的版本地址：https://github.com/docker/compose/releases。

运行以下命令以下载 Docker Compose 的当前稳定版本：

$ sudo curl -L "https://github.com/docker/compose/releases/download/v2.2.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

要安装其他版本的 Compose，请替换 v2.2.2。

Docker Compose 存放在 GitHub，不太稳定。你可以通过执行下面的命令高速安装 Docker Compose。

curl -L https://get.daocloud.io/docker/compose/releases/download/v2.4.1/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

将可执行权限应用于二进制文件：

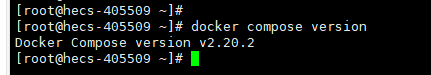
sudo chmod +x /usr/local/bin/docker-compose

创建软链：

sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

测试是否安装成功：

docker compose version



### windows操作系统

Windows 的 Docker 桌面版和 Docker Toolbox 已经包括 Compose 和其他 Docker 应用程序，因此 Windows 用户不需要单独安装 Compose。

## docker-compose.yml配置文件

### 官方文档

## 命令

### 1、docker compose build

默认使用docker-compose.yml构建镜像

### 2、docker-compose build --no-cache

### 3、docker-compose build -f docker-compose1.yml

### 4、docker-compose images

### 5、docker compose up -d

启动所有编排容器服务

### 6、docker compose version

查看版本

# Docker Swarm

## 基本概念

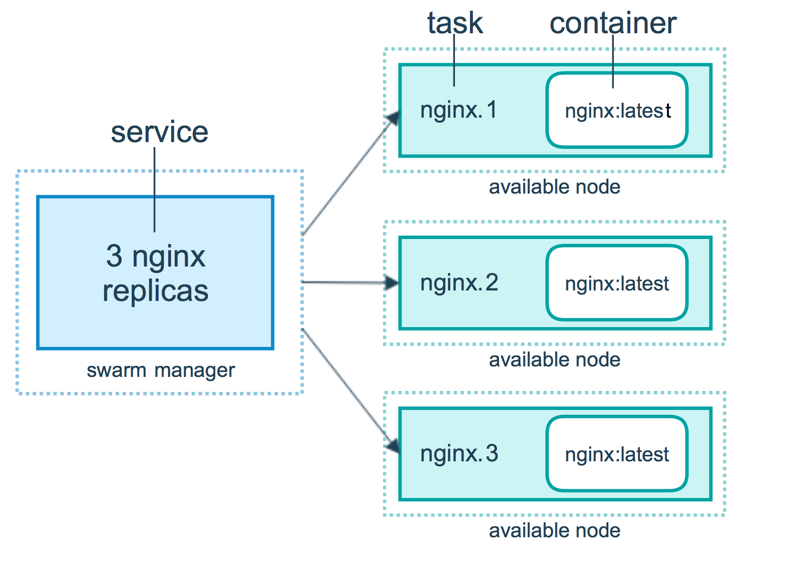
Docker Swarm 是 Docker 的集群管理工具。它将 Docker 主机池转变为单个虚拟 Docker 主机。 Docker Swarm 提供了标准的 Docker API，所有任何已经与 Docker 守护程序通信的工具都可以使用 Swarm 轻松地扩展到多个主机。worker是Docker Swarm中的一个节点，是一个运行在一个或多个Docker容器中的进程。每个Worker节点都运行一个或多个Docker容器，并且与它们交互以处理任务。

## 原理

swarm 集群由管理节点（manager）和工作节点（work node）构成。

swarm mananger：负责整个集群的管理工作，包括集群配置、服务管理等所有跟集群有关的工作。

work node：即图中的 available node，主要负责运行相应的服务来执行任务（task）。

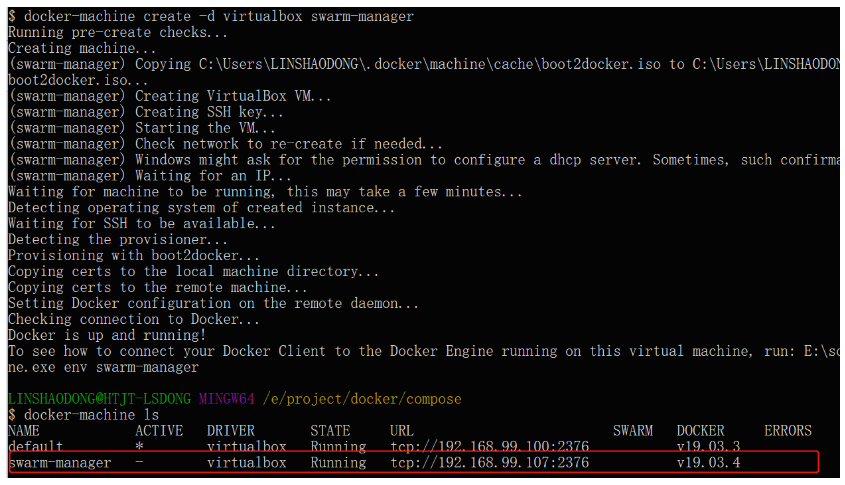


## 示例

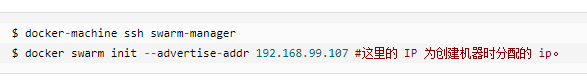
### 创建 swarm 集群管理节点（manager）

创建docker机器

docker-machine create -d virtualbox swarm-manager



初始化 swarm 集群，进行初始化的这台机器，就是集群的管理节点。



### 查看集群信息

## ****swarm create****

## ****swarm manage****

## ****swarm join****

## ****swarm list****

## docker swarm部署微服务

# compose和swarm的区别

# 容器和镜像的关系

1、镜像你可以把它看成Java中的类，而容器可以看做是类的实例化对象。

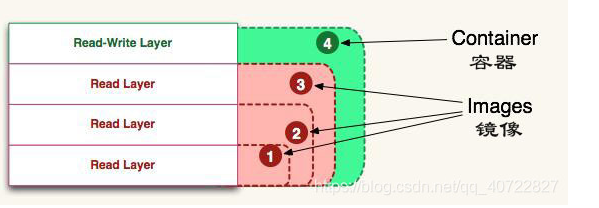
2、一个类可以有多个对象，同理一个镜像可以有多个容器。

3、docker的整个生命周期有三部分组成：镜像（image）+容器（container）+仓库（repository）。 docker 容器=镜像+可读层。

4、容器是由镜像实例化而来。简单来说，镜像是文件，容器是进程。容器是基于镜像创建的，即容器中的进程依赖于镜像中的文件。

5、docker 的镜像概念类似虚拟机的镜像。是一个只读的模板，一个独立的文件系统，包括运行容器所需的数据，可以用来创建新的容器。docker利用容器来运行应用：[docker容器](https://so.csdn.net/so/search?q=docker%E5%AE%B9%E5%99%A8&spm=1001.2101.3001.7020)是由docker镜像创建的运行实例。

6、docker容器类似虚拟机，可以执行包含启动，停止，删除等。每个容器间是相互隔离的。容器中会运行特定的运用，包含特定应用的代码及所需的依赖文件。可以把容器看作一个简易版的linux环境（包含root用户权限，进程空间，用户空间和网络空间等）和运行在其中的应用程序。



7、在Docker的生命周期中，最核心的两个部分，一个是镜像 Images，一个是容器 Containers。镜像运行起来就是容器。容器服务运行的过程中，基于原始镜像做了改变，比如安装了程序，添加了文件，也可以提交回去 (commit)成为镜像。例如：一个镜像可以包含一个完整的ubuntu操作系统环境，里面仅安装了mysql或用户需要的其他应用程序。

docker镜像实际上是由一层一层的系统文件组成，这种层级的文件系统被称为UnionFS( Union file system 统一文件系统)，镜像可以基于dockerfile构建，dockerfile是一个描述文件，里面包含了若干条命令，每条命令都会对基础文件系统创建新的层次结构。docker提供了一个很简单的机制来创建镜像或更新现有的镜像。用户甚至可以从其他人那里下载一个已经做好的镜像直接使用。（镜像是只读的，可以理解为静态文件）

8、 相对于镜像来说容器是动态的，容器在启动的时候创建了一层可写层次作为最上层。（ docker create <image -id > ：为指定的镜像添加一个可读写层，构成一个新的容器；）

docker仓库：如果使用了git和github就很容易理解docker的仓库概念。docker仓库概念和git类似。docker仓库是用来包含镜像的位置，docker提供了一个注册服务器（register）来保存多个仓库，每个仓库又可以包含多个具备不同tag的镜像。

9、docker使用的默认仓库是docker hub公共仓库。仓库支持的操作类似git，当用户创建了自己的镜像之后就可以使用push命令将它上传到共有或者私有的仓库。这样下次在另外一台机器上使用这个镜像的时候只需要从仓库里面pull下来就可以了。

# docker容器重启策略

https://mp.weixin.qq.com/s/mqwvrisZyiXdc8k5OAYj8A

# docker中镜像和容器的存储位置

Linux 存放位置如下

cd /var/lib/docker - 容器与镜像存放在此目录下

镜像位置: /var/lib/docker/image

容器位置: /var/lib/docker/containers

# docker设置DNS

# 下载常用的镜像

## docker下载jdk镜像

https://blog.csdn.net/JavaWeb\_Hao/article/details/103537865

## docker下载maven镜像

### 在Linux中单独安装JDK和在Docker中安装JDK的区别

1、环境隔离：在Linux中单独安装JDK时，JDK将直接安装在主机的操作系统上，与其他软件共享相同的操作系统资源。而在Docker中安装JDK时，JDK将作为一个独立的容器运行，与宿主机和其他容器相互隔离。

2、管理和维护：在Linux中单独安装JDK时，您需要手动下载、安装和配置JDK，并负责更新和维护JDK的版本和依赖关系。而在Docker中安装JDK时，可以使用Docker镜像来快速部署和管理JDK。Docker镜像是可重复、可移植的，可以方便地进行版本控制和升级，简化了管理和维护的工作。

3、可移植性：在Linux中单独安装JDK时，JDK的配置和依赖关系可能会因操作系统的差异而导致不兼容或需要特定的调整。而在Docker中安装JDK时，可以将包含JDK的Docker镜像在不同的主机上运行，提供了更好的可移植性和一致性。

4、资源利用率：在Linux中单独安装JDK时，JDK占用主机操作系统的资源，包括内存、处理器和磁盘空间。而在Docker中安装JDK时，JDK作为一个独立的容器运行，可以更好地控制和优化资源的使用，避免浪费。

需要注意的是，在某些情况下，可能需要在Docker容器中安装JDK，以便在容器内运行Java应用程序。这样可以确保应用程序在不同环境中的一致性，并且更方便地进行部署、扩展和管理。总结来说，单独在Linux中安装JDK更适合直接在主机上运行Java应用程序，而在Docker中安装JDK更适合在容器化环境中部署和管理Java应用程序。选择哪种方式取决于具体需求，包括环境隔离、管理和维护成本、可移植性以及资源利用率等因素。

## docker下载mysql8.0镜像

### 1、下载镜像

docker pull mysql:8.0.32

### 2、运行镜像

**docker run -p 3306:3306 --name mysql -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.6**

**正常使用docker启动mysql容器,当关闭容器时,容器中的数据会消失,所以要将mysql容器中的数据持久化到主机上,持久化数据启动mysql容器命令如下:**

**docker run -p 3306:3306 --name mysql -v /usr/local/mysql/conf:/etc/mysql/conf.d**

**-v /usr/local/mysql/logs:/logs**

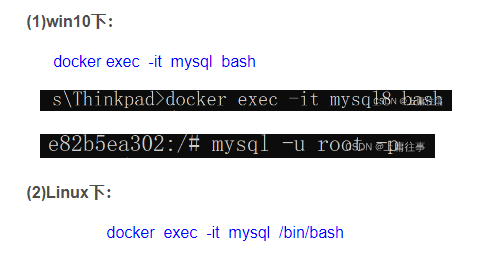
**-v /usr/local/mysql/data:/var/lib/mysql**

**-e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.6**

### 3、查看运行中的容器

docker ps

### 4、进入运行成功的容器



docker exec -it mysql /bin/bash

### 5、连接mysql

mysql -uroot -p123456

### 6、设置远程访问

ALTER USER 'root'@'%' IDENTIFIED WITH mysql\_native\_password BY '123456';

flush privileges;

## docker下载redis镜像

https://blog.csdn.net/jxysgzs/article/details/105966300#:~:text=%E7%82%B9%E5%87%BB%E8%BF%99%E9%87%8C%20%E6%9F%A5%E7%9C%8Bdocker%E6%94%AF%E6%8C%81%E7%9A%84Redis%E7%89%88%E6%9C%AC%EF%BC%8C%E6%89%BE%E5%88%B06.0.1%EF%BC%8C%E5%A4%8D%E5%88%B6%E5%8F%B3%E4%BE%A7%E5%91%BD%E4%BB%A4%E5%86%99%E5%9C%A8Linux%E4%B8%8A%E3%80%82%20%23%20docker%20pull%20redis%3A6.0.1-buster%20%E6%9F%A5%E7%9C%8B%E6%98%AF%E5%90%A6%E5%AE%89%E8%A3%85%E5%AE%8C%E6%88%90%20%23,--requirepass%20%27123456%27%20--bind%200.0.0.0%20--daemonize%20NO%20--protected-mode%20yes

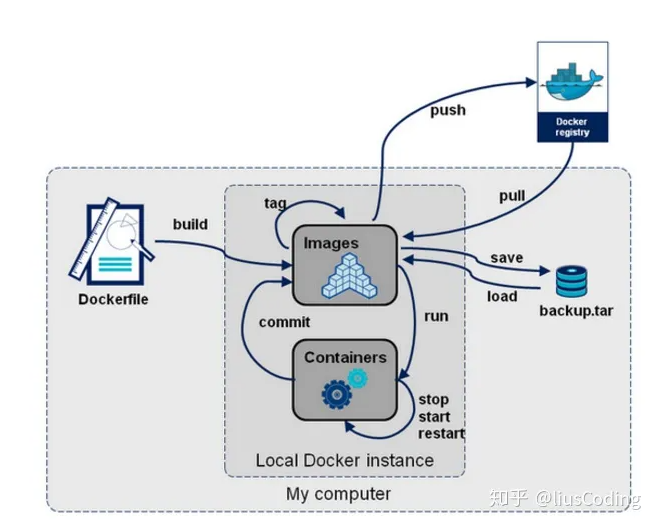
## docker下载nacos2.x镜像

## docker下载nginx镜像

https://blog.csdn.net/BThinker/article/details/123507820

## docker下载python镜像

# docker常用的命令



## 容器生命周期管理

### run

#### 基本说明

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：

-d：后台运行容器，并返回容器ID。

-p：指定端口映射，格式为：主机(宿主)端口:容器端口。

--name="nginx-lb": 为容器指定一个名称。

-v：用于挂载（Mount）宿主机上的目录或文件到容器中。

docker run -v <宿主机目录>:<容器目录> [OPTIONS] IMAGE [COMMAND] [ARG...]

#### 实例

##### 1、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

##### 2、挂载宿主机目录到容器

docker run -v /host/path:/container/path IMAGE

将宿主机上的 /host/path 目录挂载到容器中的 /container/path 目录。容器内对 /container/path 的修改会直接影响到宿主机上 /host/path 目录，反之亦然。

##### 3、挂载宿主机文件到容器

docker run -v /host/file:/container/file IMAGE

将宿主机上的 /host/file 文件挂载到容器中的 /container/file 文件。容器内对 /container/file 的修改会直接影响到宿主机上 /host/file 文件，反之亦然。

##### 4、使用命名卷挂载数据

docker run -v volume\_name:/container/path IMAGE

通过使用命名卷（Named Volume），可以在容器中持久化存储数据，并且可以在多个容器之间共享数据。

### create

#### 基本概念

用于创建一个新的容器，但并不启动该容器。它会在后台创建一个容器实例，并返回该容器的ID。稍后可以使用 docker start 命令来启动这个容器。

#### 语法

docker create [选项] <镜像名称> [命令] [参数]

-t, --tty：分配一个伪终端（TTY）并绑定到容器的标准输入上，使得容器内部的应用程序能够接收和处理终端输入。

-i, --interactive：保持容器的标准输入打开，即使没有连接到容器的标准输出。这通常与 -t 选项一起使用，以便能够与容器进行交互。

--name <名称>：为容器指定一个名称。

#### 示例

##### 1、创建容器

docker create --name mycontainer -it ubuntu:latest bash

上述命令将创建一个名为 mycontainer 的容器，基于最新的 Ubuntu 镜像，并在容器启动后进入交互式的 Bash shell。

当 docker create 命令成功执行时，它会返回一个容器的唯一标识符（ID），如：



通过这个 ID，你可以使用其他 Docker 命令对容器进行操作，例如启动容器、停止容器、删除容器等。

### start

#### 基本说明

启动一个或多个已经被停止的容器。

#### 语法

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

#### 实例

##### 1、启动已被停止的容器myrunoob

docker start myrunoob

### stop

#### 基本说明

停止一个运行中的容器

#### 语法

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

#### 实例

##### 1、停止运行中的容器myrunoob

docker stop myrunoob

### restart

#### 基本说明

重启容器

#### 语法

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

#### 实例

##### 1、重启容器myrunoob

docker restart myrunoob

### kill

#### 基本说明

杀掉一个运行中的容器。

#### 语法

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

OPTIONS说明：

**-s :**向容器发送一个信号

#### 实例

##### 1、杀掉运行中的容器mynginx

docker kill -s KILL mynginx

#### stop和kill的区别

stop：

1、行为：发送一个停止信号（SIGTERM）给容器进程，要求容器优雅地停止运行。

2、效果：容器会收到停止信号后，将进行一些清理工作并优雅地终止。这通常涉及处理未完成的请求、保存状态等。如果容器进程没有在指定的时间内响应停止信号，Docker 会强制发送 SIGKILL 信号给容器。

3、使用场景：当需要安全地停止容器，并允许容器在退出之前完成一些清理工作时，推荐使用 docker stop。

kill

1、行为：发送一个强制终止信号（SIGKILL）给容器进程，立即停止容器运行，无论容器是否愿意。这个信号无法被容器捕获或处理。

2、效果：容器会立即终止，与 docker stop 不同，容器没有机会进行清理工作，中断的操作可能导致数据丢失或其他问题。

3、使用场景：当需要立即停止容器，或者容器无法通过 docker stop 正常停止时，可以使用 docker kill。但因为容器没有机会进行善后工作，可能会有一些副作用。

综上所述，docker stop 发送一个终止信号给容器进程，要求容器优雅地停止运行，而 docker kill 发送一个强制终止信号给容器进程，立即停止容器的运行。在大多数情况下，推荐使用 docker stop，除非确实需要强制终止容器。

### rm

#### 基本说明

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

##### 1、强制删除容器 db01、db02

docker rm -f db01 db02

### pause

#### 基本说明

暂停容器中所有的进程。

#### 语法

docker pause CONTAINER [CONTAINER...]

#### 实例

##### 1、暂停数据库容器db01提供服务。

docker pause db01

#### stop和pause命令的区别

pause

行为：暂停容器中的所有进程，冻结容器的状态。

效果：容器的所有进程会被暂停，无法进行任何操作。容器的文件系统和资源占用状态都会保持不变。

使用场景：当需要暂时暂停容器中的所有进程，而不改变容器的状态或清理工作时，可以使用 docker pause。例如，当容器运行的应用程序在某个时间段内无需处理请求，可以暂停容器以节省资源。

综上所述，docker stop 发送一个停止信号给容器进程，要求优雅地停止容器运行，而 docker pause 则是冻结容器中的所有进程，暂时停止操作。使用 docker stop 可以安全地停止容器，并允许容器进行清理工作，而使用 docker pause 可以暂停容器的所有进程，节省资源但不改变容器状态。根据具体需求选择适当的命令。

### unpause

#### 基本说明

恢复容器中所有的进程。

#### 语法

docker unpause CONTAINER [CONTAINER...]

#### 实例

##### 1、恢复数据库容器 db01 提供服务。

docker unpause db01

### exec

#### 基本说明

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

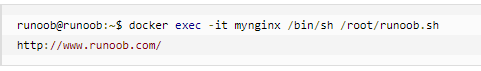
**-i :**即使没有附加也保持STDIN（标准输入流） 打开

**-t :**分配一个伪终端

#### 实例

##### 1、在容器 mynginx 中以交互模式执行容器内 /root/runoob.sh脚本

docker exec -it mynginx /bin/sh /root/runoob.sh



##### 2、在容器 mynginx 中开启一个交互模式的终端

docker exec -i -t mynginx /bin/bash



## 容器操作

### ps

#### 基本说明

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。

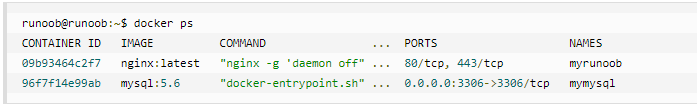
**-q :**静默模式，只显示容器编号。

**-s :**显示总的文件大小。

#### 实例

##### 1、列出所有在运行的容器信息。

docker ps



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）

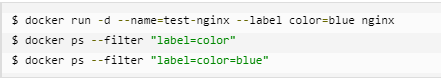
exited（停止）

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。

##### 2、根据标签过滤



##### 3、根据名称过滤



docker ps –filter “name=test-nginx”

##### 4、查看某个容器

docker ps -a | grep container-id

### inspect

#### 基本说明

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 :**如果类型为容器，显示总的文件大小。

**--type :**为指定类型返回JSON。

#### 实例

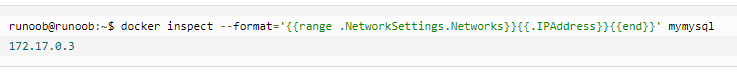
##### 1、获取镜像mysql:5.6的元信息。

docker inspect mysql:5.6



##### 2、获取正在运行的容器mymysql的 IP地址。

docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mymysql



也可以使用这个命令：

docker inspect --format='{{.NetworkSettings.IPAddress}}' 容器名称|容器ID

##### 3、查看容器的挂载目录

### top

#### 基本说明

查看容器中运行的进程信息，支持 ps 命令参数。

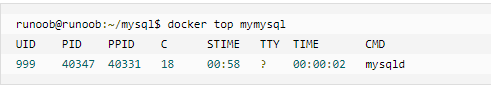
#### 语法

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

##### 1、查看容器mymysql的进程信息。



### port

#### 基本说明

用于列出指定容器的端口映射，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]]

<容器名称或ID>: 要查看端口映射的容器的名称或容器ID。

[私有端口/协议]：可选参数，指定要查看的具体端口和协议，格式为 私有端口/协议，例如 80/tcp。不指定此参数时，将显示容器中所有的端口映射。

使用 docker port 命令后，它将返回一个或多个端口映射的结果，每个结果包括：

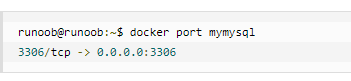
1、私有端口：容器内部正在监听的端口号。

2、宿主机地址：映射到容器内部私有端口的宿主机地址。

3、公开端口：在宿主机上用于访问容器的公共端口。

#### 实例

##### 1、查看容器 mymysql 的端口映射情况



#### 说明

[::] 是一个特殊的 IPv6 地址表示，表示IPv6的 "未指定" 或 "任意地址"。

### logs

#### 基本说明

获取容器的日志。需要注意的是当容器被删除时，其日志也将被删除。因此，在需要长期保存容器日志的情况下，应该将日志输出到外部存储设备或其他容器中。

#### 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

##### 1、跟踪查看容器的日志

docker logs -f mynginx



##### 2、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

说明：日期格式只能是yyyy-MM-dd，不能加时分秒。

##### 3、显示容器最新的50行日志

docker logs --tail 50 容器名称/容器ID

##### 4、查看指定时间的日志

将 UNIX 时间戳传递给 docker logs 命令，以筛选指定时间之后的日志。例如，要查找在 2023 年 6 月 10 日 18:30:00 之后记录的日志，可以将对应的 UNIX 时间戳 1689238200 传递给 docker logs 命令：

docker logs --since 1689238200 container-id

如果要查找在指定时间之前的日志，则使用 --until 参数。例如，要查找在 2023 年 6 月 10 日 18:30:00 之前记录的日志，可以使用以下命令：

docker logs --until 1689238200 container-id

注意，时间戳必须是 UNIX 时间戳，表示自 1970 年 1 月 1 日以来的秒数。可以使用 date 命令将人类可读的时间转换为 UNIX 时间戳，例如：

date -d "2023-06-10 18:30:00" +%s

输出结果为：1689238200。

##### 5、在日志中查找指定的字符串

使用 Linux 的文本搜索工具（如 grep、awk）来查找特定的字符串。例如，要查找包含字符串 "error" 的日志记录，可以将 docker logs 命令的输出通过管道传递给 grep 命令：

docker logs container-id | grep "error"

这将筛选出所有包含 "error" 字符串的日志记录，并将它们打印到终端上。除了 grep，还可以使用其他文本搜索工具来进行高级搜索，比如：

1、使用 awk 命令来查找特定字段，例如：

docker logs container-id | awk '/error/ {print $1, $2}'

这将筛选出所有包含 "error" 字符串的日志记录，并打印出每条日志的第一个和第二个字段。

2、使用正则表达式来匹配更复杂的模式，例如：

docker logs container-id | grep -E '(error|warning)'

这将筛选出所有包含 "error" 或 "warning" 字符串的日志记录。

### stats

#### 基本概念

显示容器资源的使用情况，包括：CPU、内存、网络 I/O 等。

#### 语法

docker stats [OPTIONS] [CONTAINER...]

OPTIONS 说明：

**--all , -a :**显示所有的容器，包括未运行的。

**--format :**指定返回值的模板文件。

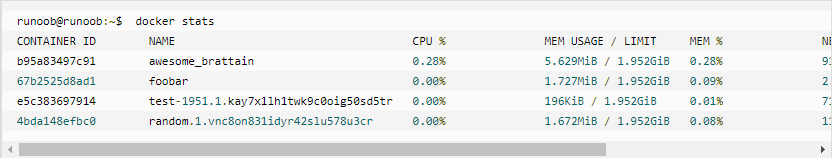
**--no-stream :**展示当前状态就直接退出了，不再实时更新。

**--no-trunc :**不截断输出。

#### 实例

##### 1、列出所有运行的容器信息。

docker stats



输出详情介绍：

**CONTAINER ID 与 NAME:** 容器 ID 与名称。

**CPU % 与 MEM %:** 容器使用的 CPU 和内存的百分比。

**MEM USAGE / LIMIT:** 容器正在使用的总内存，以及允许使用的内存总量。

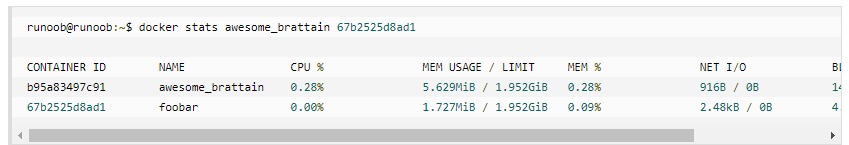
**NET I/O:** 容器通过其网络接口发送和接收的数据量。

**BLOCK I/O:** 容器从主机上的块设备读取和写入的数据量。

**PIDs:** 容器创建的进程或线程数。

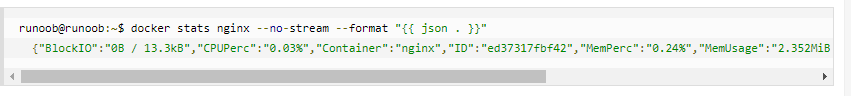
##### 2、根据容器等 ID 或名称显示信息

docker stats awesome\_brattain 67b2525d8ad1



##### 3、以json格式输出

docker stats nginx --no-stream --format "{{ json . }}"



### attach

#### 基本概念

连接到正在运行中的容器。用于将终端附加到运行中的Docker容器。它可以连接到已在后台运行的容器，并将终端输入和输出与容器的标准输入（STDIN）、标准输出（STDOUT）和标准错误（STDERR）进行关联。

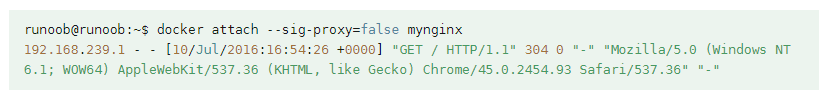
#### 语法

docker attach [OPTIONS] CONTAINER

attach是可以带上--sig-proxy=false来确保CTRL-D或CTRL-C不会关闭容器。

#### 示例

##### 1、容器mynginx将访问日志指到标准输出，连接到容器查看访问信息。



## 容器rootfs命令

### cp

#### 基本概念

用于在 Docker 容器和本地主机之间进行文件或目录的拷贝。

#### 语法

docker cp [OPTIONS] CONTAINER:SRC\_PATH DEST\_PATH

其中，CONTAINER 表示容器的名称或 ID；SRC\_PATH 表示容器内的源文件或目录路径；DEST\_PATH 表示本地主机上目标的文件或目录路径。

以下是 docker cp 命令的一些常用选项：

-a, --archive：以归档模式复制文件。保留文件的权限和特性。

-L, --follow-link：如果源文件是符号链接，则复制链接指向的目标文件。

-R, --recursive：递归地复制目录及其内容。

#### 示例

##### 1、从容器复制文件到本地主机

docker cp mycontainer:/app/file.txt /host/path/file.txt

这将复制容器 mycontainer 中的 /app/file.txt 文件到本地主机的 /host/path/file.txt。

##### 2、从容器复制目录到本地主机

docker cp mycontainer:/app/data /host/path/data

这将复制容器 mycontainer 中的 /app/data 目录及其内容到本地主机的 /host/path/data。

##### 3、从本地主机复制文件到容器

docker cp /host/path/file.txt mycontainer:/app/file.txt

这将复制本地主机上的 /host/path/file.txt 文件到容器 mycontainer 的 /app/file.txt。

##### 4、从本地主机复制目录到容器

docker cp /host/path/data mycontainer:/app/data

这将复制本地主机上的 /host/path/data 目录及其内容到容器 mycontainer 的 /app/data

需要注意的是，通过 docker cp 命令复制文件时，容器和主机之间的路径需指定为绝对路径。如果需要复制整个目录，可以使用 -R 选项。使用 docker cp 命令时，容器和主机之间的文件传输是直接的，不经过网络。

### commit

#### 基本概念

从容器创建一个新的镜像。

#### 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。

#### 实例

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



## 镜像仓库

### login

#### 基本概念

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。

#### 语法

docker login [OPTIONS] [SERVER]

#### 实例

### logout

### pull

#### 基本说明

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

##### 1、从Docker Hub下载java最新版镜像。

docker pull java

##### 2、从Docker Hub下载REPOSITORY为java的所有镜像。

docker pull -a java

### push

#### 基本说明

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

##### 1、上传本地镜像myapache:v1到镜像仓库中。

docker push myapache:v1

### search

#### 基本概念

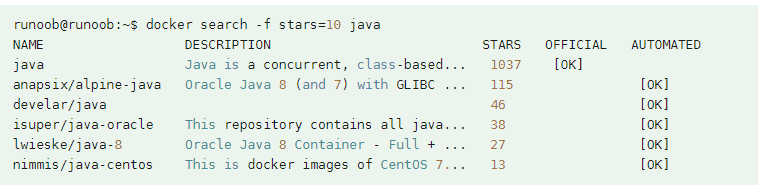
从Docker Hub查找镜像。

#### 语法

docker search [OPTIONS] TERM

#### 示例

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



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。

## 本地镜像管理

### images

#### 基本概念

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；

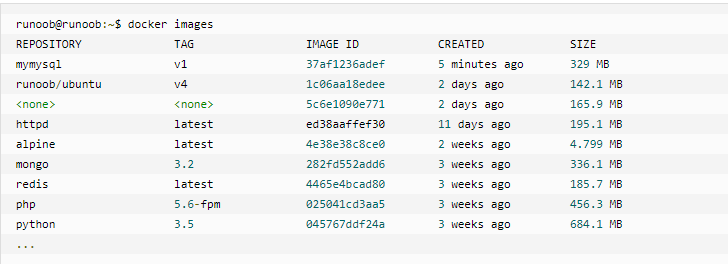
**--no-trunc :**显示完整的镜像信息；

**-q :**只显示镜像ID。

#### 实例

##### 1、查看本地镜像列表。

docker images



EPOSITORY:表示来自于哪个仓库。

TAG:表示镜像的标签信息，标签只是标记，并不能标识镜像内容。

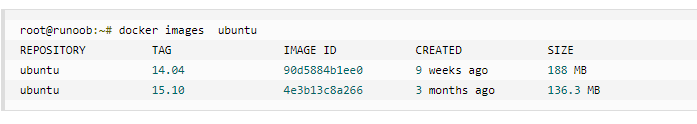
IMAGE ID:镜像ID，镜像的唯一标识，如果两个镜像ID相同，则说明它们实际上指向了同一个镜像，只是具有不同标签名而已。

CREATED:表示镜像最后的更新时间。

VIRTUAL SIZE：表示镜像大小，好的镜像往往体积会较小。

##### 2、列出本地镜像中REPOSITORY为ubuntu的镜像列表。

docker images ubuntu



### rmi

#### 基本说明

删除本地一个或多个镜像。

#### 语法

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

OPTIONS说明：

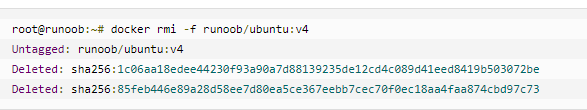
**-f :**强制删除

**--no-prune :**不移除该镜像的过程镜像，默认移除

#### 实例

##### 1、强制删除本地镜像 runoob/ubuntu:v4

docker rmi -f runoob/ubuntu:v4



### tag

#### 基本概念

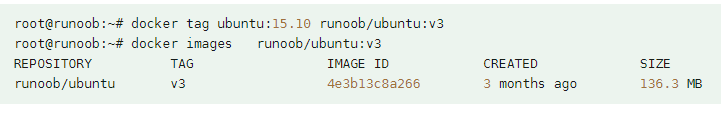
标记本地镜像，将其归入某一仓库。

#### 语法

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

#### 示例

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



### build

#### 基本说明

用于使用 Dockerfile文件创建镜像。

#### 语法

docker build [OPTIONS] PATH | URL | -

OPTIONS说明：

--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。

-f：指定要使用的Dockerfile路径。

#### 实例

##### 1、使用当前目录的 Dockerfile 创建镜像，标签为 runoob/ubuntu:v1。

docker build -t runoob/ubuntu:v1 .

##### 2、使用URL github.com/creack/docker-firefox 的 Dockerfile 创建镜像。

docker build github.com/creack/docker-firefox

##### 3、可以通过 -f Dockerfile 文件的位置

docker build -f /path/to/a/Dockerfile .

### history

#### 基本概念

查看指定镜像的创建历史。

#### 语法

#### 示例

### save

#### 基本概念

将指定镜像保存成 tar 归档文件。

#### 语法

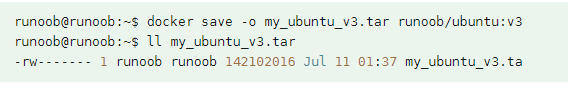
docker save [OPTIONS] IMAGE [IMAGE...]

OPTIONS 说明：

**-o :**输出到的文件。

#### 示例

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



### load

### import

## 版本信息

### info

#### 基本说明

显示 Docker 系统信息，包括镜像和容器数。

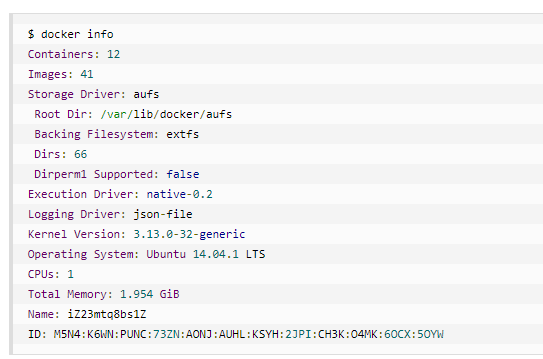
#### 语法

docker info [OPTIONS]

#### 实例

##### 1、查看docker系统信息。

docker info



### version

#### 基本说明

显示 Docker 版本信息。

#### 语法

docker version [OPTIONS]

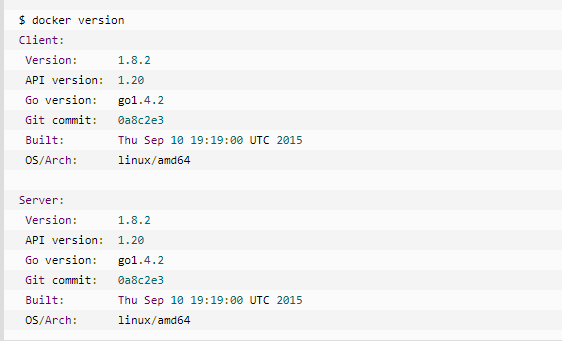
OPTIONS说明：

**-f :**指定返回值的模板文件。

#### 实例

##### 1、显示 Docker 版本信息。

docker version



# docker swarm命令

## docker swarm init

## 初始化swarm manager并制定网卡地址

docker swarm init --advertise-addr 192.168.10.117

这个命令重要

## 查看集群中的节点

docker node ls

将输出一个表格，显示当前 Swarm 集群中每个节点的状态，包括节点 ID、主机名、状态、角色和可用性等信息。

## 创建一个指定name的服务

docker service create –name my-nginx nginx

## 为指定的服务更新一个端口

docker service update --publish-add 80:80 my-nginx

## 强制重启服务

docker service update –force 服务名

## 删除一个服务

docker service rm my-nginx

## 查看应用的详细信息

docker service inspect[OPTIONS]SERVICE [SERVICE...]

参数：

-f,-format string:使用Go模板指定格式化输出

-pretty:以适合阅读的格式输出。

## 查看swarm集群服务列表

docker service ls

这个命令将输出一个表格，显示当前 Swarm 集群中所有服务的信息，包括服务名称、状态、副本数、镜像、端口映射等信息。

## 查看 Swarm 集群任务列表

docker service ps <service\_name>

这个命令将列出指定服务的所有任务并显示它们的状态、节点、版本和运行时间等信息。需要将 <service\_name> 替换为实际的服务名称。

## 查看 Swarm 集群网络列表

docker network ls

将列出当前 Swarm 集群中所有网络的名称、ID、驱动程序和范围等信息。

# 问题

## 1、Connect to localhost:2375 [localhost/127.0.0.1, localhost/0:0:0:0:0:0:0:1] failed: Connection refused: connect -> [Help 1]

## 2、决Docker pull时报错：https://registry-1.docker.io/v2/

3、