第二章：Docker架构、镜像与容器

**一、Docker概述；**

**二、Docker核心概念；**

**三、Docker的应用场景；**

**四、案例：安装Docker并实现镜像、容器管理；**

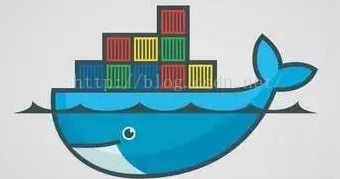
**五、案例：Docker容器资源控制；**

**一、Docker概述；**

**集装箱：**集装箱，英文名container.集装箱的出现，大大降低了货物运输的成本,实现了货物运输的标准化,以此为基础逐步建立全球范围内的船舶、港口、航线、公路、中转站、桥梁、隧道、多式联运相配套的物流系统,世界经济形态因此而改变。集装箱最大的成功在于其产品的标准化以及由此建立的一整套运输体系。



**背景：**Docker诞生于2013年，最初的设计者为dotCloud公司，后来改名为Docker Inc。Docker项目已经加入到了linux基金会，全部代码均在https://github.com/docker上进行维护。其设计图标如下，Docker，直译是码头工人。这个码头工人搬运的是集装箱（Container）；其设计宗旨：Build，Ship and Run any App，Any where，即通过对应用组件的封装、发布、部署、运行等生命周期的管理，达到应用组件级别的“一次封装，到处运行”的目的，这里的组件可以理解为一个应用，一个服务，或者是一个完整的系统；



**概述：**Docker通过Linux Container技术将任意类型的应用进行包装，由GO语言开发而成，变成一种轻量级的，标准化的，可移植的，自管理的组件。因为这种对应用的包装不仅包装了应用程序本身，更将应用的依赖和环境一并进行打包，所以她允许我们可以将这个包拿到任意环境去运行，提高效率的同时，不再有兼容性的问题。比如：需要将腾讯云上的服务迁移到阿里云，如若使用传统迁移的方法会非常麻烦，但是如果使用docker，则只需要将docker容易迁移到新的服务器上，启动容器即可；

**优势：**

1.体积小：减小系统的开销值，一台主机可以运行上千个容器；

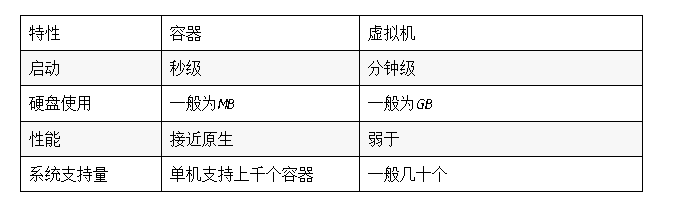
2.启动迅捷：更快速的交付和部署，docker容器启动、停止可以实现秒级，比传统虚拟机要快很多，docker核心解决的问题就是利用容器实现VM类似的功能；

3.操作方便：通过配置dockerfile配置文件支持灵活的自动化创建和部署；

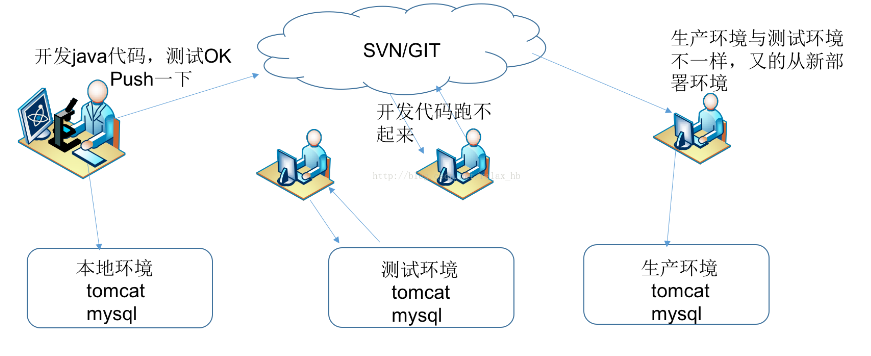
4.更轻松的扩展：可以实现更简单的、更可靠的迁移，避免了兼容性等问题；

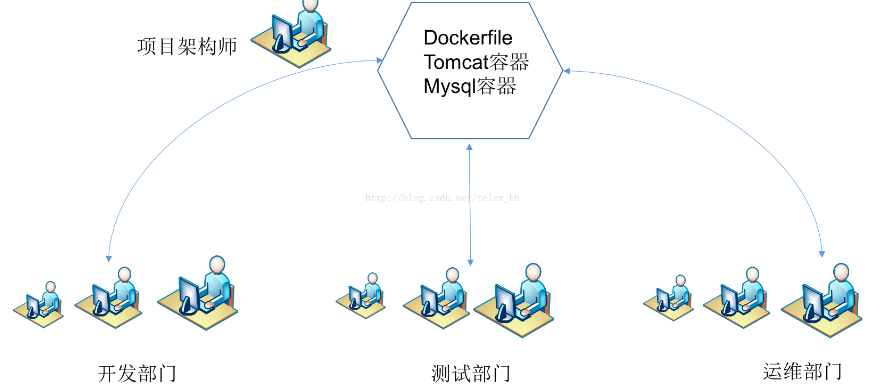
5.更强的可扩展性和可移植性：





**Docker所解决的问题：**





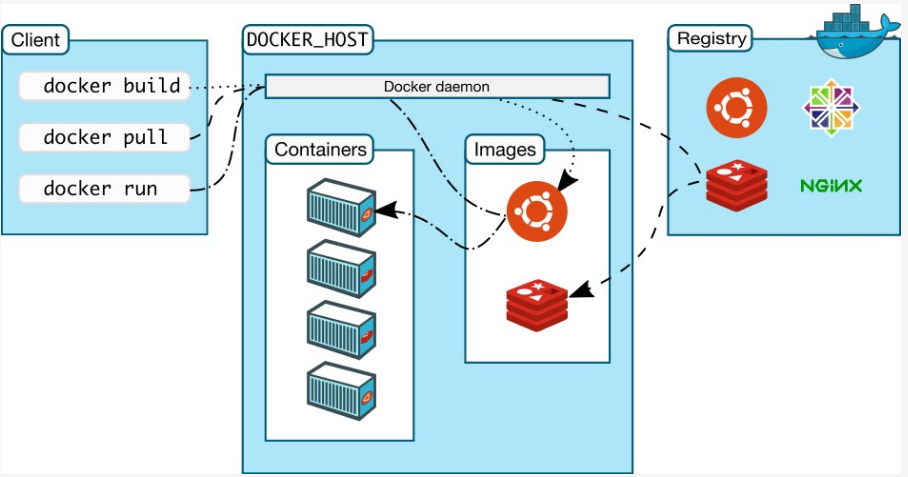
**二、Docker核心概念；**

**角色：**

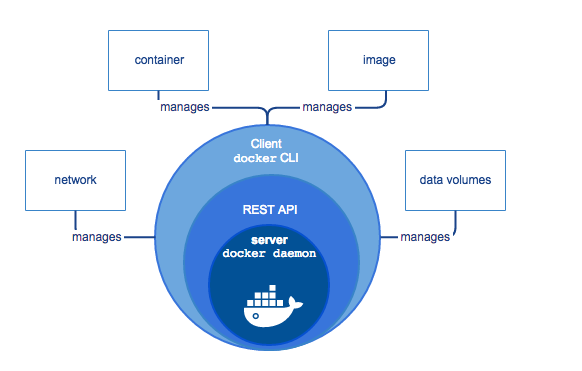
1.client：客户端

2.server：服务端

通常两个角色都集成在一台主机中；



**核心概念：**

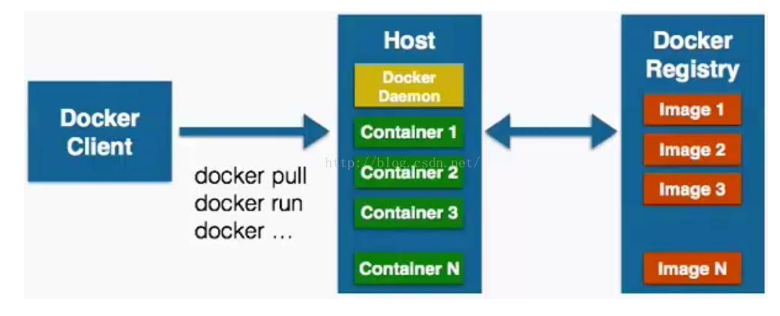


**1.镜像（Image）：**创建容器的基础，类似于虚拟机的快照功能，可以理解为一个面向docker容器引擎的只读模板，每个镜像都有一个tag标签，作为其标识；比如：一个镜像可以是一个完整的centos系统，称之为centos镜像；也可以是一个安装了mysql的应用程序，称之为mysql镜像；

**2.容器（Container）：**基于镜像所创建的实例，可以被启动、停止、删除，多个容器之间互相隔离，互不可见的，可以保证容器的进程隔离性与平台的安全性，也可以把容器看作成为一个简易的linux环境，可以利用容器来部署业务；

**3.仓库（Repository）：**仓库作为docker集中存放镜像的位置，当创建了自己的镜像后，可以使用push命令将镜像上传到公共仓库（public）或者私有仓库（private），这样下次再次使用时，直接从仓库pull下来即可；

仓库注册服务器（registry）是存放仓库的地方，其中包含很多仓库，如果无私有仓库，则使用当前docker最大的公共仓库docker hub，其中存放了数量庞大的镜像供用户下载使用（注：docker hub的账号需要翻墙注册，国内网络无法刷新人机验证码）；



**总结：**

镜像（Image）就是存放在仓库（Repository）中的一堆只读层（read-only layer）的统一视角，可以在主机文件系统上找到有关这些层的文件容器（container）只读层+读写层，在一个运行中的容器内部，这些层是不可见的。

**三、Docker的应用场景：**

**工作职责：**

开发人员在使用docker的过程中，关心的是docker容器中运行的应用程序；

运维人员在使用docker的过程中，关心的是如何管理docker容器；

**docker的应用场景：隔离性、标准性；**

1.加快本地开发和构建流程：更加高效的、轻量级的，本地开发人员构建、运行、分享docker容器，容器可以在开发环境中构建、提交到测试环境中、最终进入生产环境中；

2.能够让独立服务与应用程序在不同的环境中，得到相同的运行结果；

3.使用docker创建隔离的环境用来进行测试：比如jenkins这样的持续集成工具启动一个用于测试的容器；

4.构建一个多用户的平台即服务PAAS基础设施；

5.提供软件即服务SAAS应用程序，如：memcahed、mongodb等；

6.超大规模、高性能的宿主机部署；

**四、案例：安装Docker并实现镜像、容器管理：**

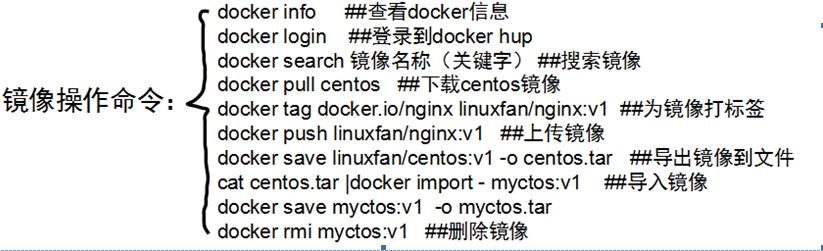
**案例环境：**

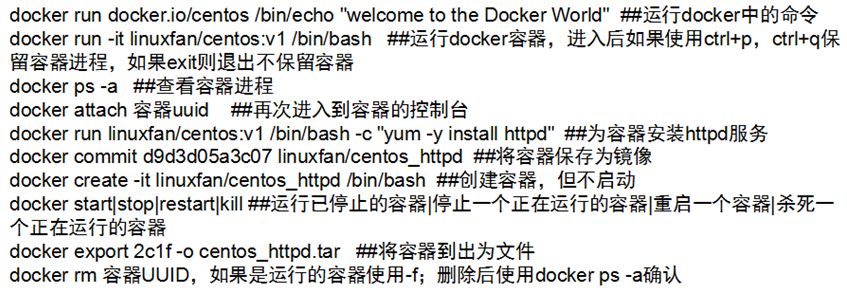
|  |  |  |  |
| --- | --- | --- | --- |
| 系统类型 | IP地址 | 主机名 | 所需软件 |
| Centos7.4 1708 64bit | 192.168.100.101 | docker.linuxfan.cn | docker-engine |

**注解：docker默认的存储目录在/var/lib/docker/，默认的镜像、容器、日志等都存放在此，建议使用LVM逻辑卷，避免出现docker运行过程中存出目录容量不足的问题；**

**案例步骤：**

* 安装docker容器并启动：
* 注册docker hub账号并登陆：
* 注册docker 加速器账号；
* 登陆docker hub账户并设置加速；
* 镜像管理--检索可用的镜像：
* 镜像管理--下载镜像：
* 镜像管理--删除镜像：
* 镜像管理--存出与载入镜像：
* 镜像管理--上传镜像：
* **容器管理--创建容器：**
* 容器管理--启动容器：
* 容器管理--容器的终止：
* 容器管理--容器的进入：
* 容器管理--容器的导出和导入：
* 容器管理--为centos容器安装httpd服务：
* 容器管理--将容器保存至镜像；
* 容器管理--容器的删除：

****

****

* **安装docker容器并启动：**

[root@docker ~]# cat <<END >>/etc/yum.repos.d/centos.repo

[local2]

name=docker

baseurl=https://yum.dockerproject.org/repo/main/centos/7/

enable=1

gpgcheck=1

gpgkey=https://yum.dockerproject.org/gpg

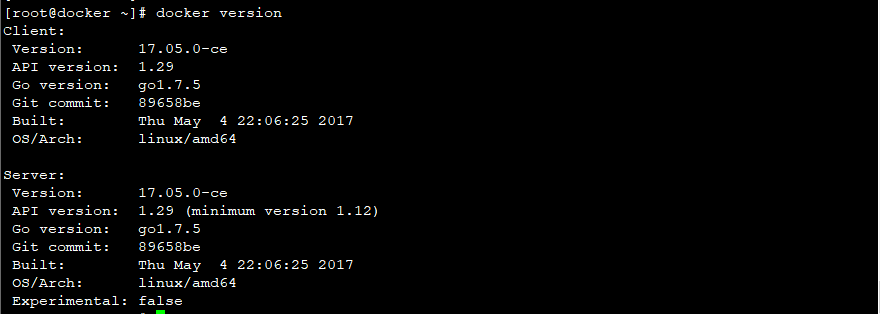
END

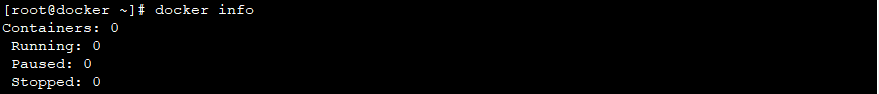
[root@docker ~]# yum -y install docker-engine ##网络卡顿多安装几次

[root@docker ~]# systemctl start docker

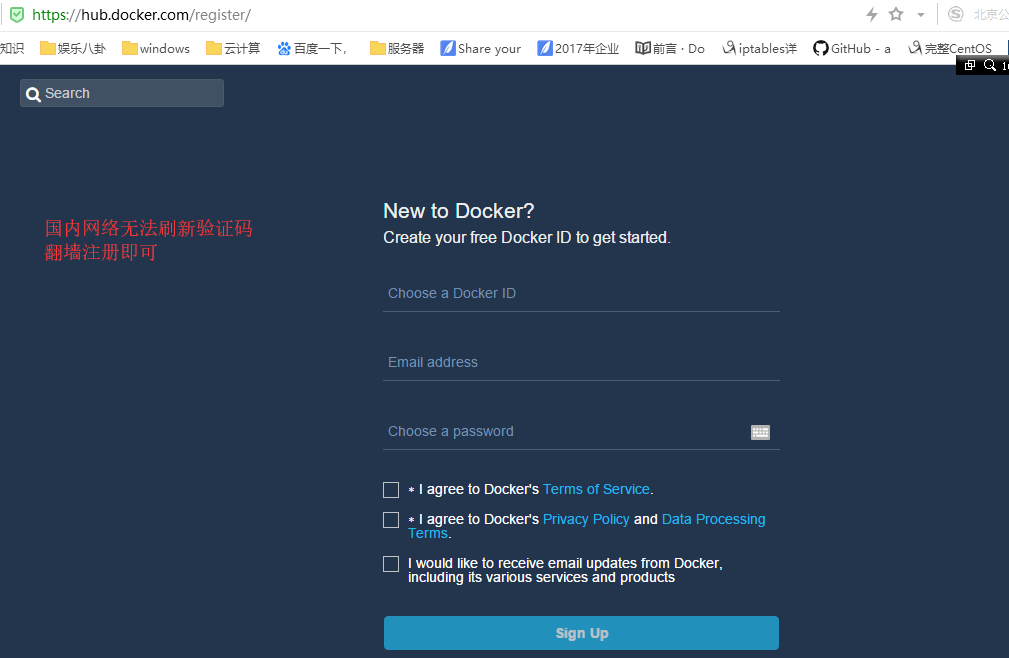
[root@docker ~]# systemctl enable docker

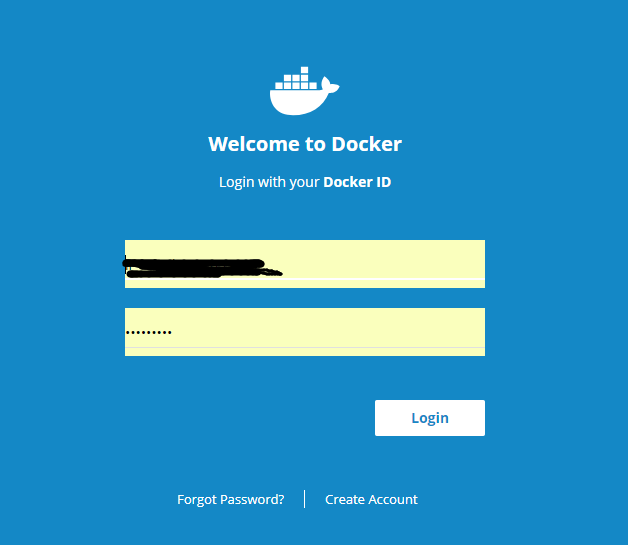
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.



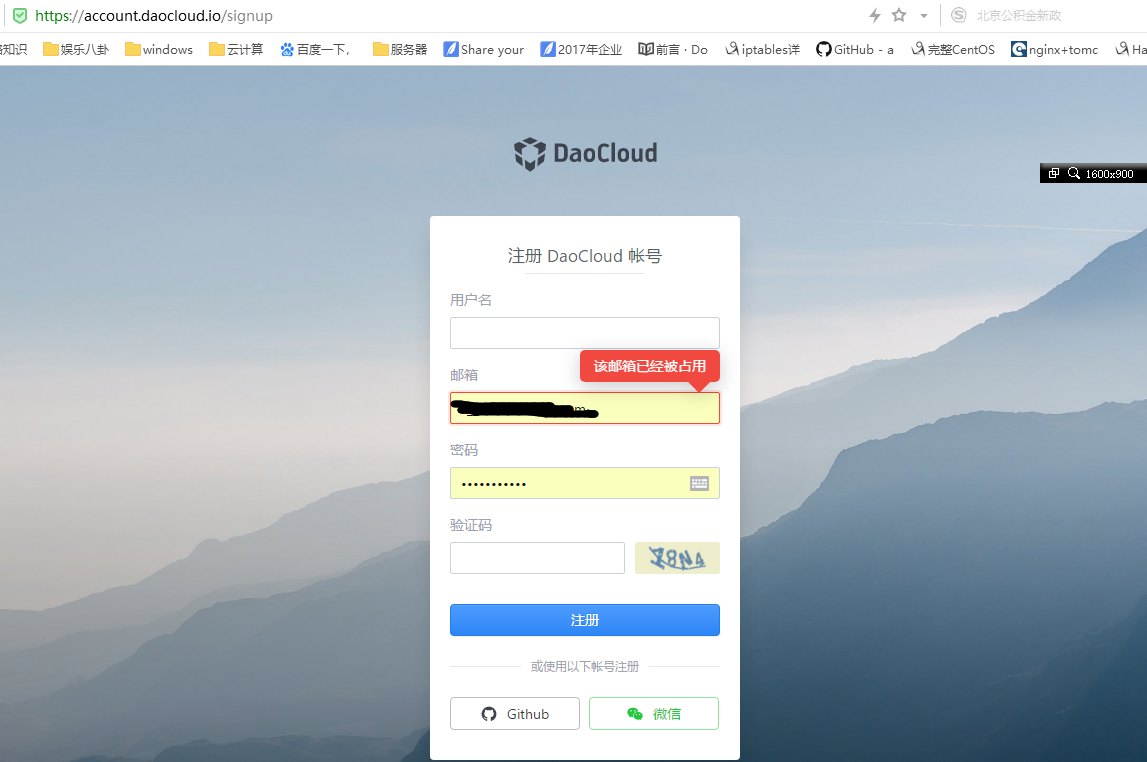


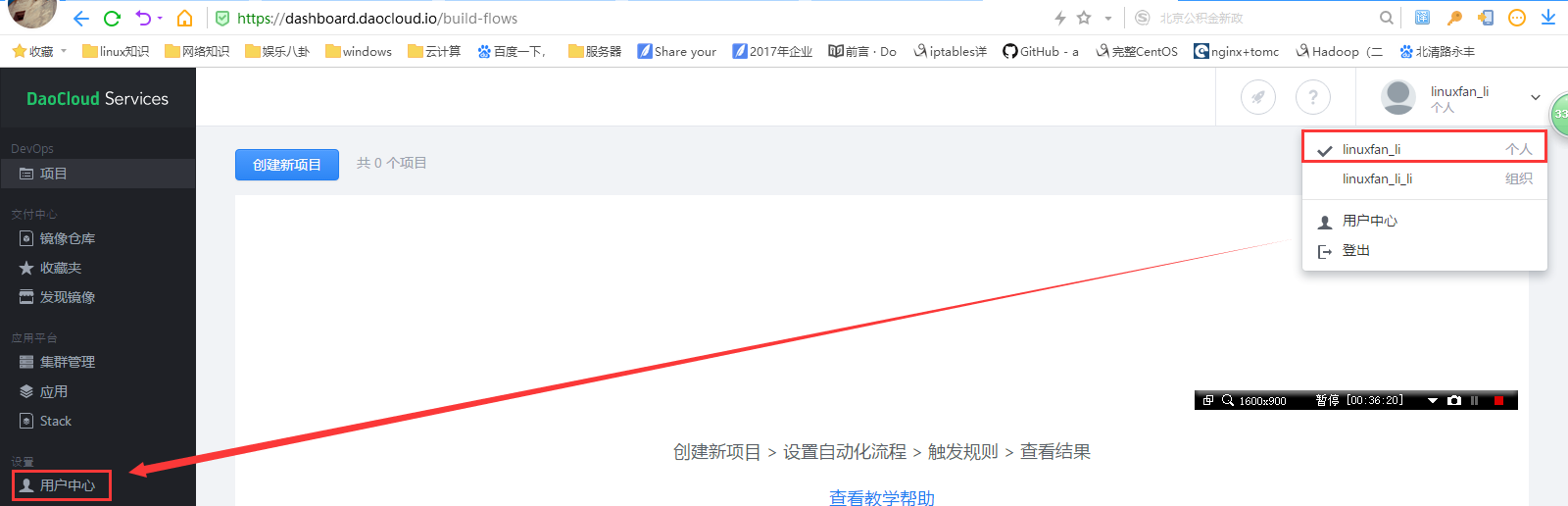
* **注册docker hub账号并登陆：**





* **注册docker 加速器账号并设置加速；**









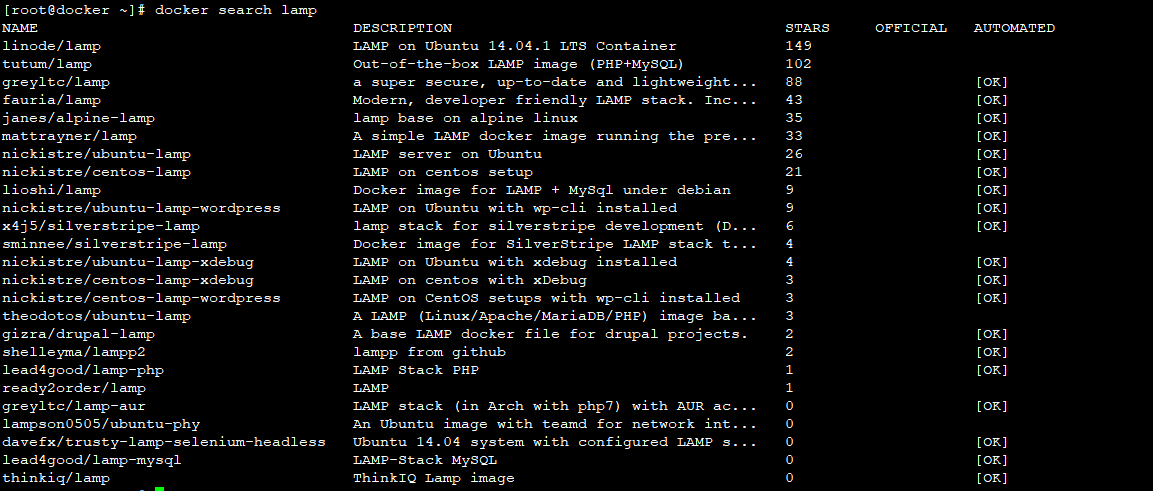
* **登陆docker hub账户并设置加速；**

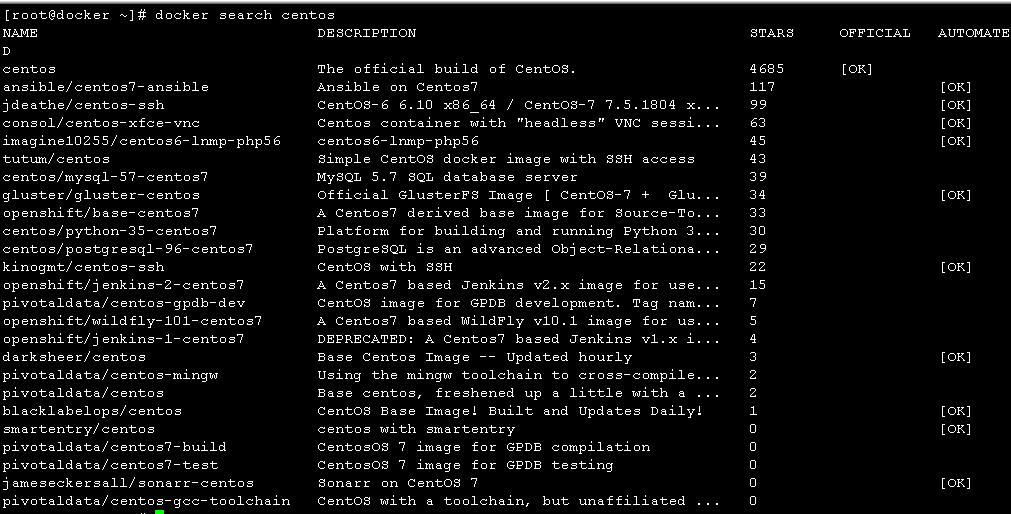


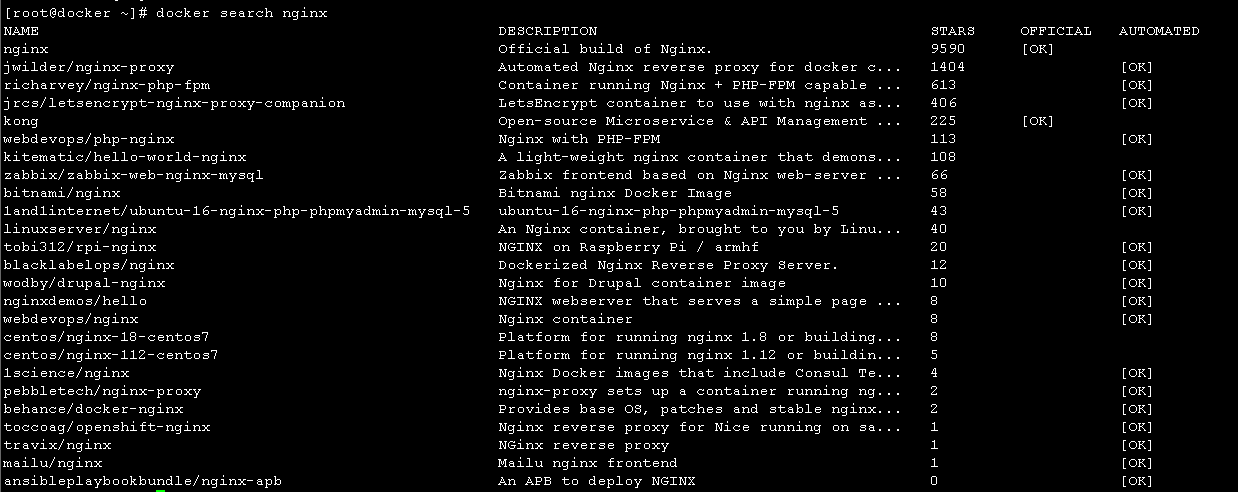
* **镜像管理--检索可用的镜像：**

搜索与关键字相关的镜像：

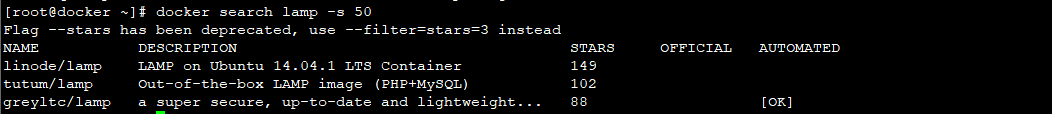
格式：名称 描述 星级（受欢迎级别） 是否是官方创建 是否主动创建（是否允许用户查看镜像的来源和内容）

****



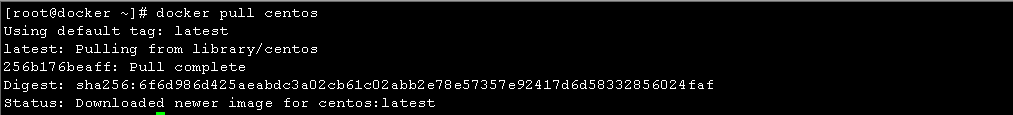


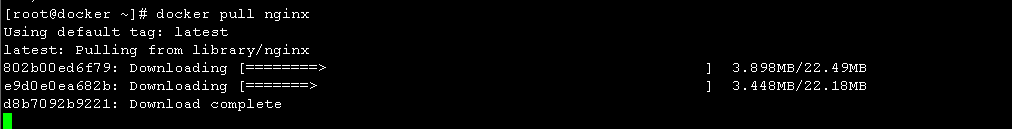
根据关键字和星级进行搜索：

****

* **镜像管理--下载镜像：**

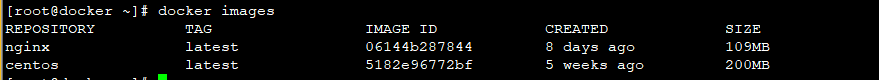
下载镜像的过程中会发现，镜像文件是由若干层（Layer）组成，称之为AUFS（联合文件系统），是实现增量保存和更新的基础，用户也可以在其他注册服务器仓库下载镜像，需要指定仓库名称前指定完整的仓库注册服务器地址；



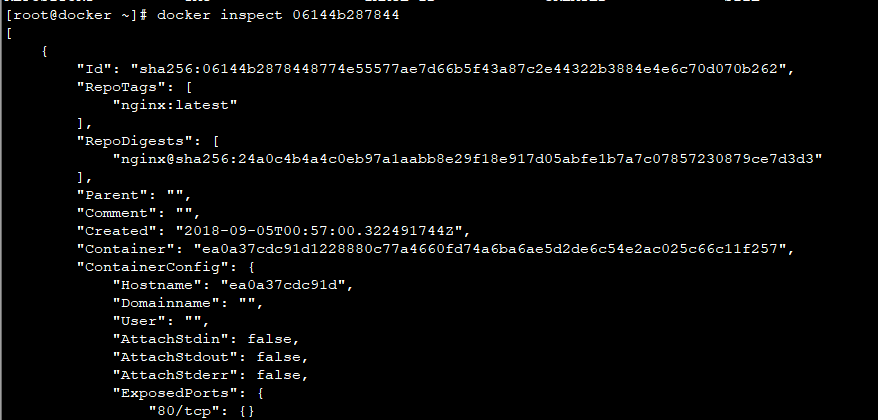


查看已经下载的镜像：

格式：镜像来源的仓库 标签 id号码 创建时间 大小

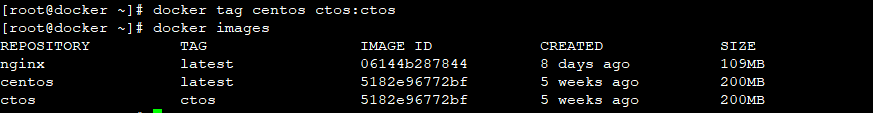


查看镜像的详细信息：



* **镜像管理--为镜像打标签；**

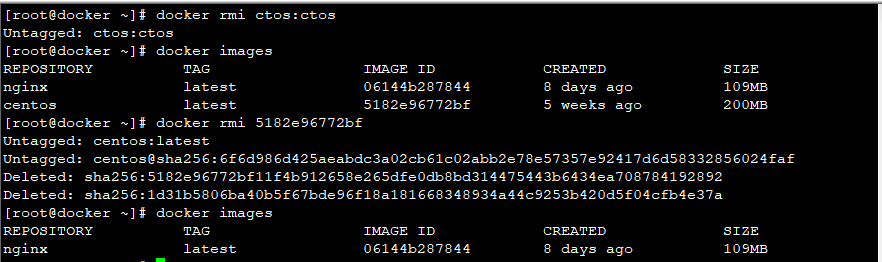
**TAG字段为标签：**每个镜像都富有的标记，类似于别名；



* **镜像管理--删除镜像：**

**格式：**docker rmi 仓库名称：标签 ##当一个源镜像有多个标签，删除的只是当前标签，不会影响源镜像文件

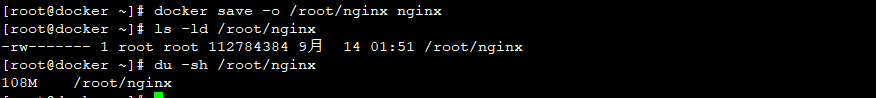
docker rmi 镜像id号码 ##保证当前镜像没有被容器所依赖，删除时会先删除当前镜像的所有的标签，然后将镜像文件删除

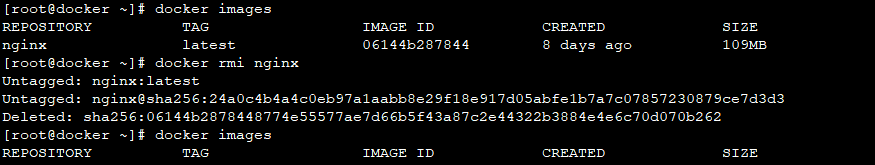


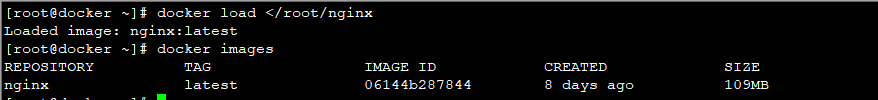
* **镜像管理--存出与载入镜像：**

**存出镜像：**将镜像保存成为本地文件；

**载入镜像：**将本地文件拷贝到其他主机后，将文件导入到其他主机的镜像中；

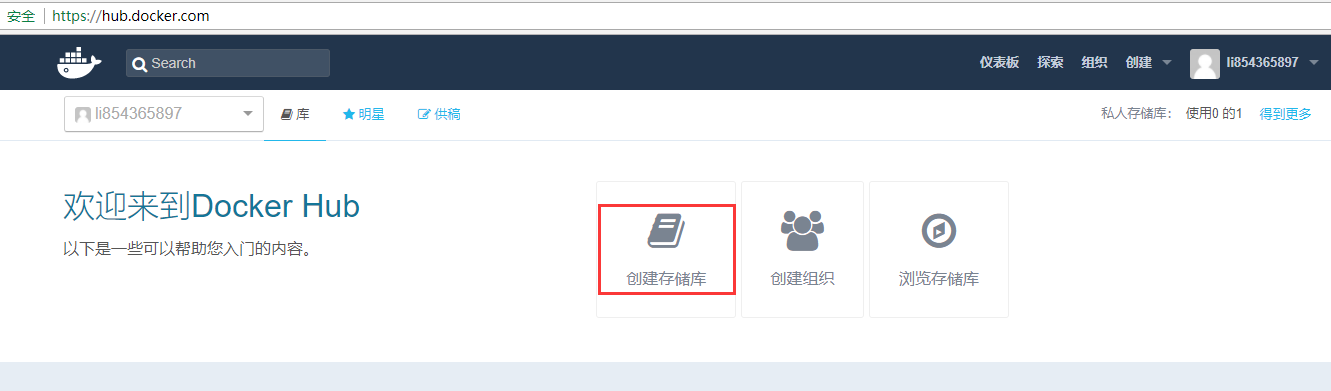


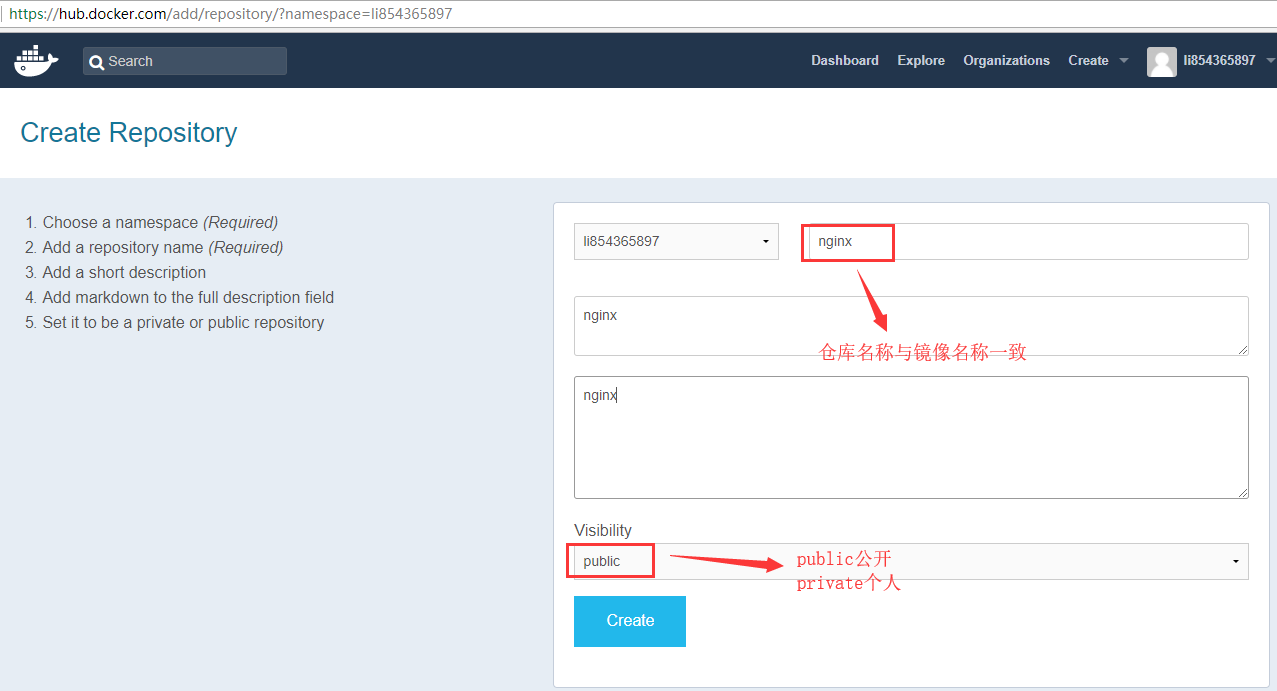


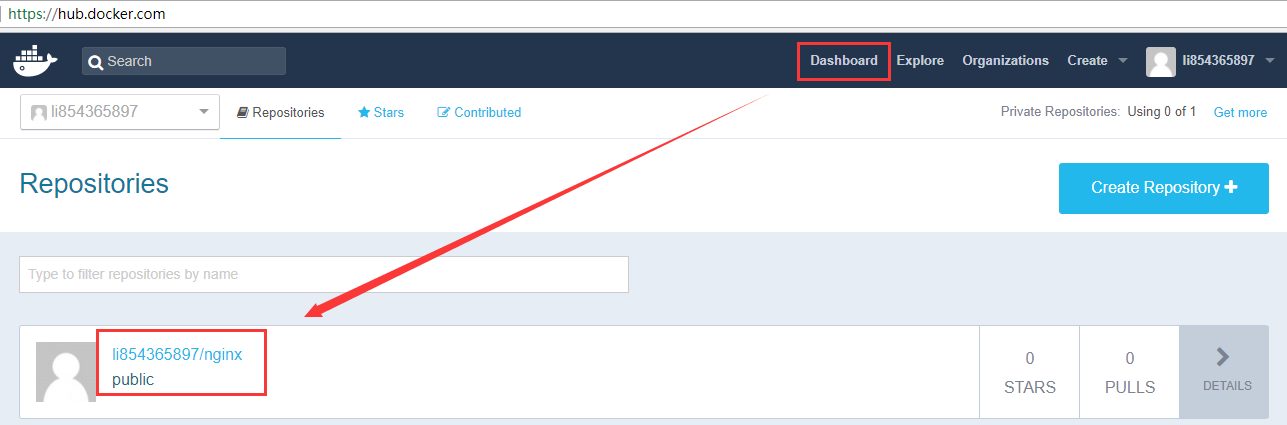


* **镜像管理--上传镜像：**

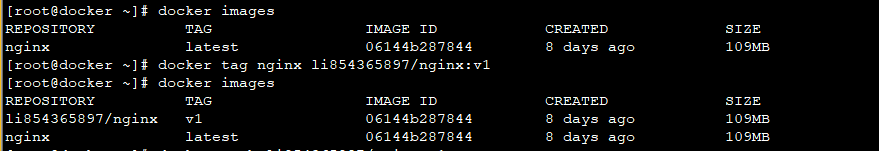
创建docker hub上的存储仓库：

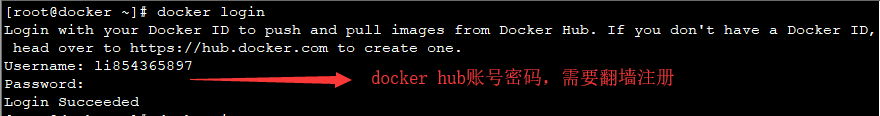


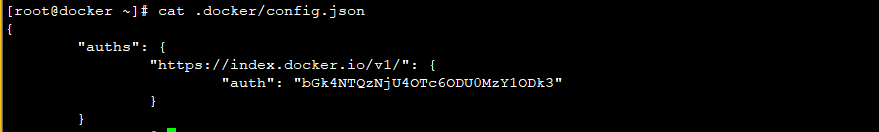


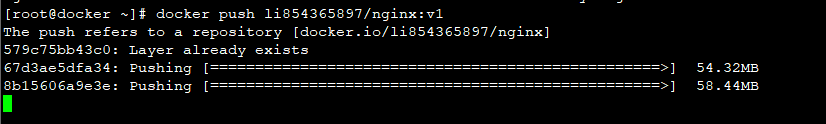


上传镜像到docker hub公共仓库：

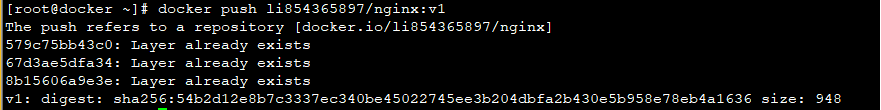




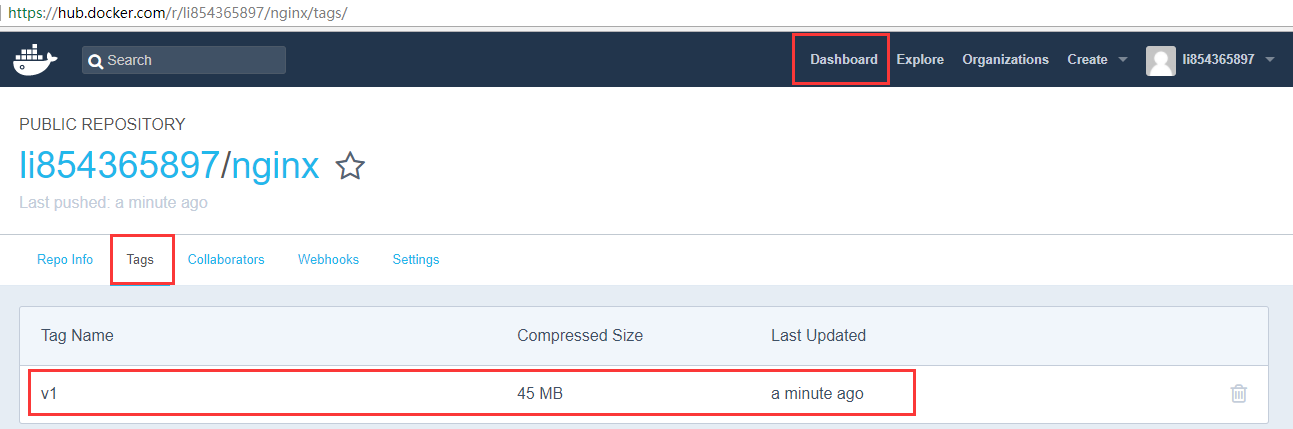




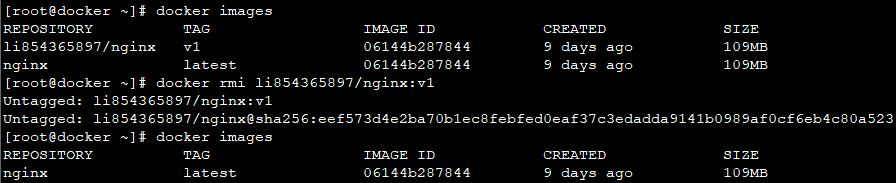
网络失败，再来一次；

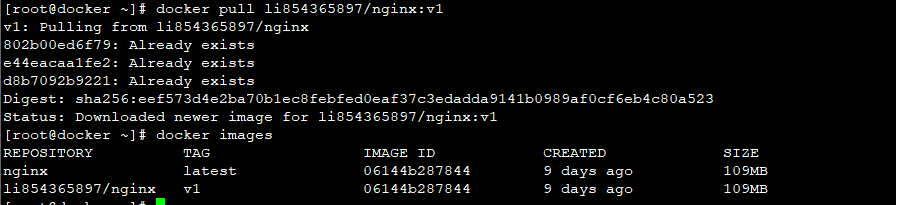


验证docker hub公共仓库的镜像存储位置：



删除dcker服务器本地的镜像，测试下载刚刚上传到docker hub的镜像：





* **容器管理--创建容器：**

**语法：**docker create [选项] [--name 容器名称] [...] 镜像 运行的程序

**选项：**-i ##让容器的输入保持打开

-t ##为docker分配一个伪终端

-d ##使容器在后台daemon一直运行

[--name 容器名称] ##指定容器名称，如若不指定则与镜像名称一致

[-p 宿主机本地端口:容器端口] ##端口映射

[-P] ##自动将容器端口映射到宿主机本地的49000到49900

[-v 宿主机本地目录:容器目录] ##将宿主机目录与容器目录进行挂载

[--volumes-from 源数据卷容器位置] ##指定源数据卷容器位置



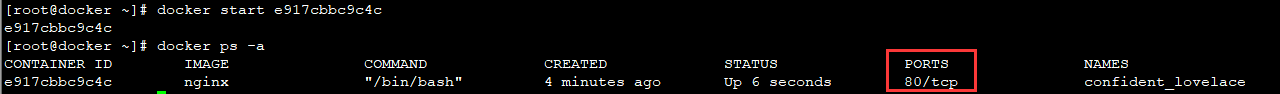
**查看容器的运行状态：**

**格式：**ID 加载的镜像 运行的程序 创建时间 当前所处的状态 端口映射



* **容器管理--启动容器：**

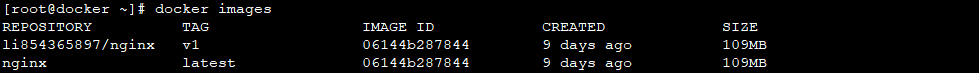
**语法：**docker start 容器的id或者名称

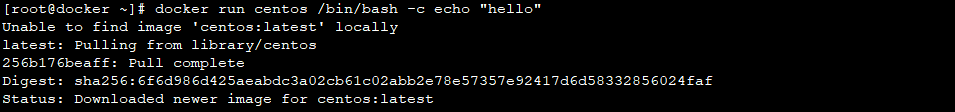


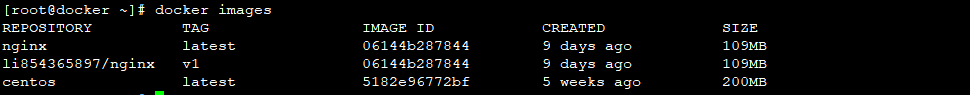
**创建并启动容器：**等于docker pull 命令+docker create 命令+docker start 命令

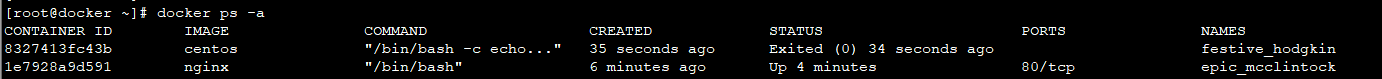
**语法：**docker run 容器的名称 运行的程序 [ -c "命令" ]

**注解：**执行此命令后，docker后台会检测此容器需要的镜像是否存在，如果本地不存在则去公共仓库下载，下载完毕后，根据镜像创建一个容器，分配一个文件系统给容器，在只读的镜像层外边挂载一层可读写层，从宿主机配置的网桥接口中桥接一个虚拟机借口到容器中，分配一个地址池中的ip地址给容器，执行用户所执行的命令，执行完成后将容器终止运行；

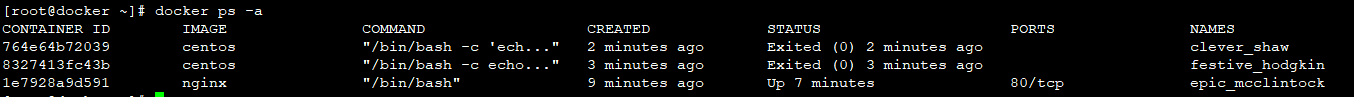


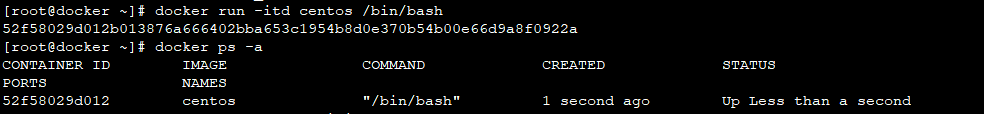










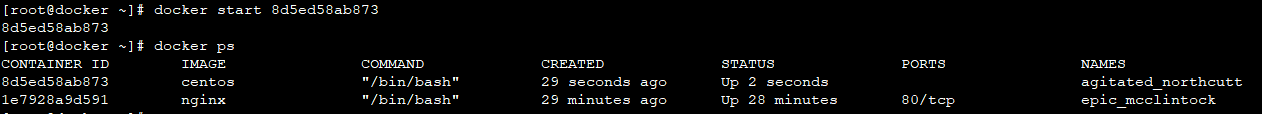


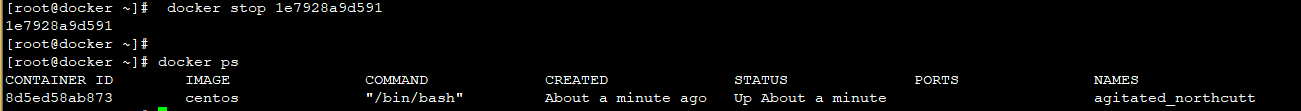


* **容器管理--容器的终止：**

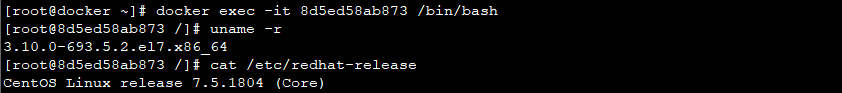


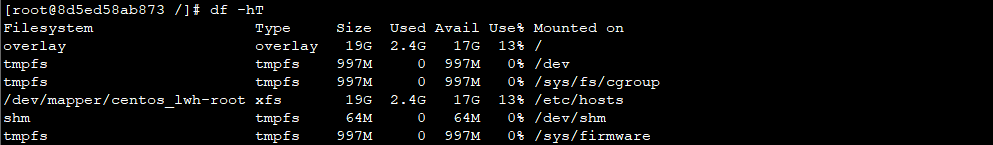




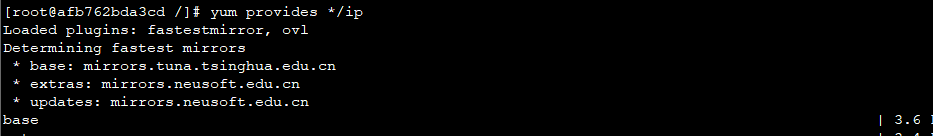


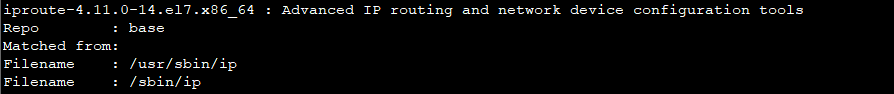
* **容器管理--容器的进入：**

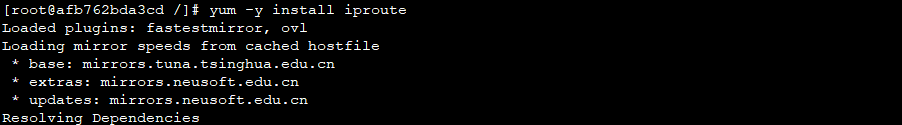


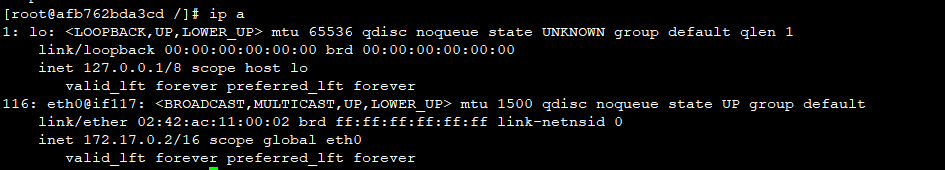


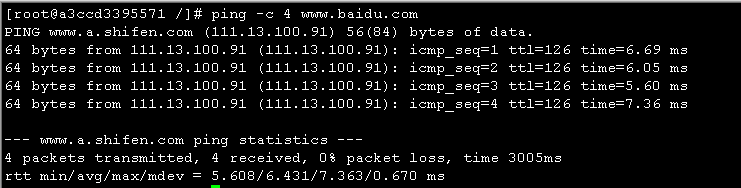


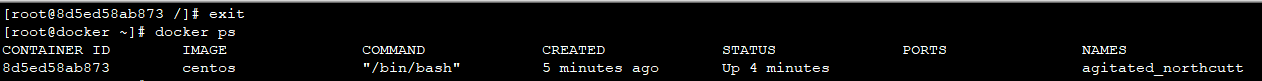






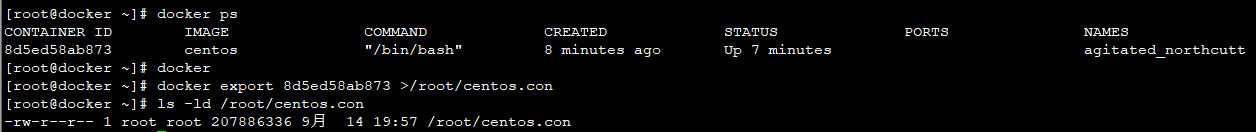




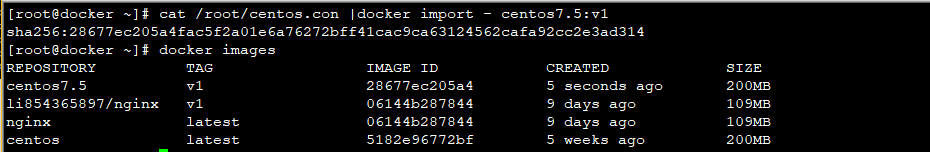


* **容器管理--容器的导出和导入：**

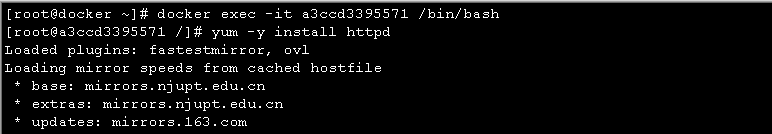
导出容器到系统文件：



导入系统文件到docker镜像中：



* **容器管理--为centos容器安装httpd服务：**

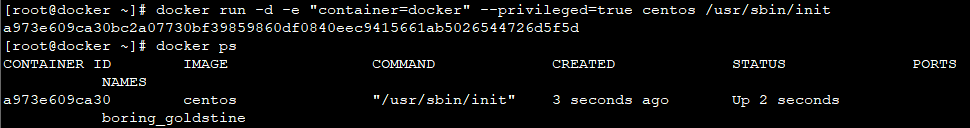


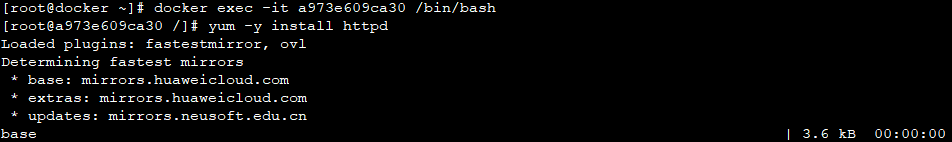


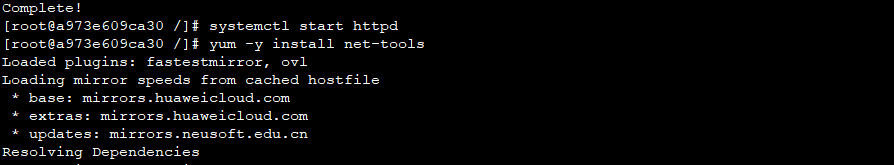


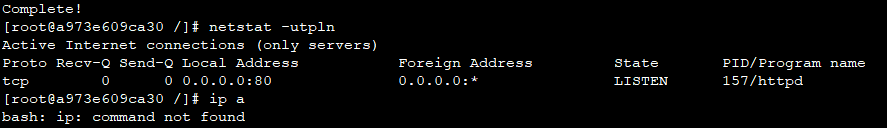


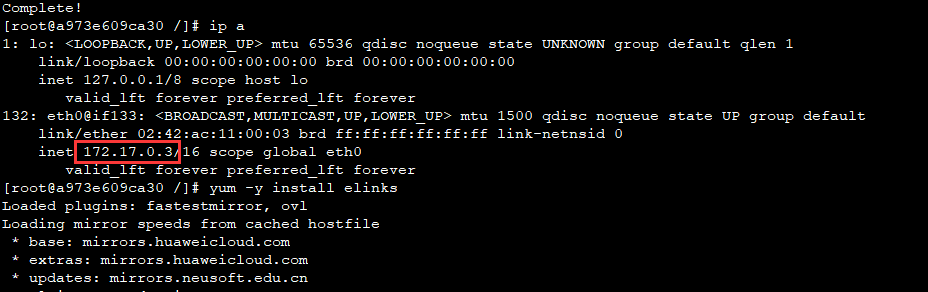
发现httpd服务无法启动（也可以使用/usr/sbin/apachectl命令启动），因为需要在run容器时，需要指定启动程序为/usr/sbin/init；



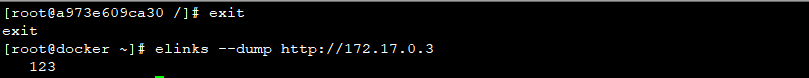




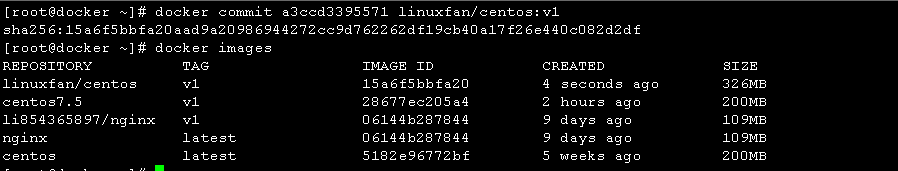




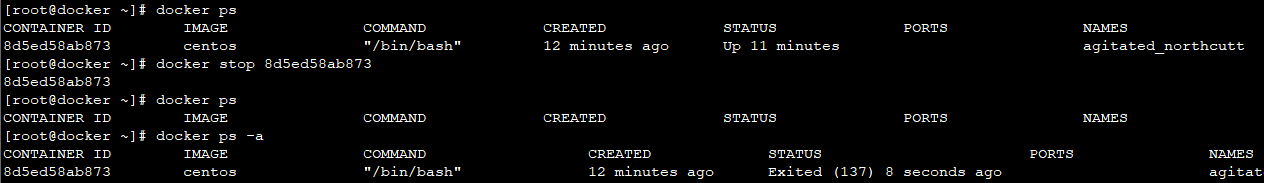


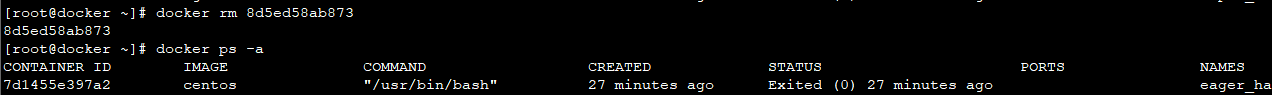


* **容器管理--将容器保存至镜像；**



* **容器管理--容器的删除：**





**五、案例：Docker容器资源控制；**

**Cgroup概述：**control group的简写，是linux内核提供的一种限制所使用物理资源的机制，这些资源主要包括CPU、内存、blkio；

**CPU的控制：**

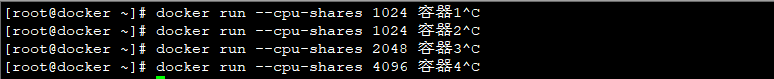
1.限制CPU的使用速率：



2.修改cgroup配置文件实现限制使用速率：



3.根据多个容器设置其占用CPU的比例，如下为1:1:2:4：



4.根据多个容器进行绑定所使用的CPU的核心，如下为八颗CPU，分配给两个容器：



**内存的控制：**

如下：限制容器1使用的内存为512M；



**blkio的控制：**

如果一台服务器上多个容器的部署，会出现同时又多个程序进程读写磁盘数据的情况；

