

# Docker

---

- 1.docker是什么
- 2.docker相关组件说明
- 3.docker的安装和启动
- 4.docker相关的容器的创建
- 5.docker的常用的命令
- 6.docker部署项目（系统）

## 1 docker的介绍

---

Docker的思想来自于集装箱，集装箱解决了什么问题？在一艘大船上，可以把货物规整的摆放起来。并且各种各样的货物被集装箱标准化了，集装箱和集装箱之间不会互相影响。这样就可以不用单独使用其他的运输工具。大家都用一个标准，搬运集装箱了。

- 1.不同的应用程序可能会有不同的应用环境，有些软件安装之后会有端口之间的冲突，这时候，可以使用虚拟机来实现隔离，但是使用虚拟机的成本太高，而且消耗硬件。
- 2.不同的软件的环境都不一样，比如：你用的是乌班图，里面有个数据库，现在要迁移到centos中，但是此时需要从新在centos安装数据库，如果版本不一致，或者不支持，就会出现问题。比较麻烦。有了docker之后就不用这么麻烦了，直接将开发环境 搬运到不同的环境即可。
- 3.在服务器负载方面，如果你单独开一个虚拟机，那么虚拟机会占用空闲内存的，docker部署的话，这些内存就会利用起来。

### 1.1虚拟化

#### 1.1.1什么是虚拟化

在计算机中，虚拟化（英语：Virtualization）是一种资源管理技术，是将计算机的各种实体资源，如服务器、网络、内存及存储等，予以抽象、转换后呈现出来，打破实体结构间的不可切割的障碍，使用户可以比原本的组态更好的方式来应用这些资源。这些资源的新虚拟部份是不受现有资源的架设方式，地域或物理组态所限制。一般所指的虚拟化资源包括计算能力和资料存储。

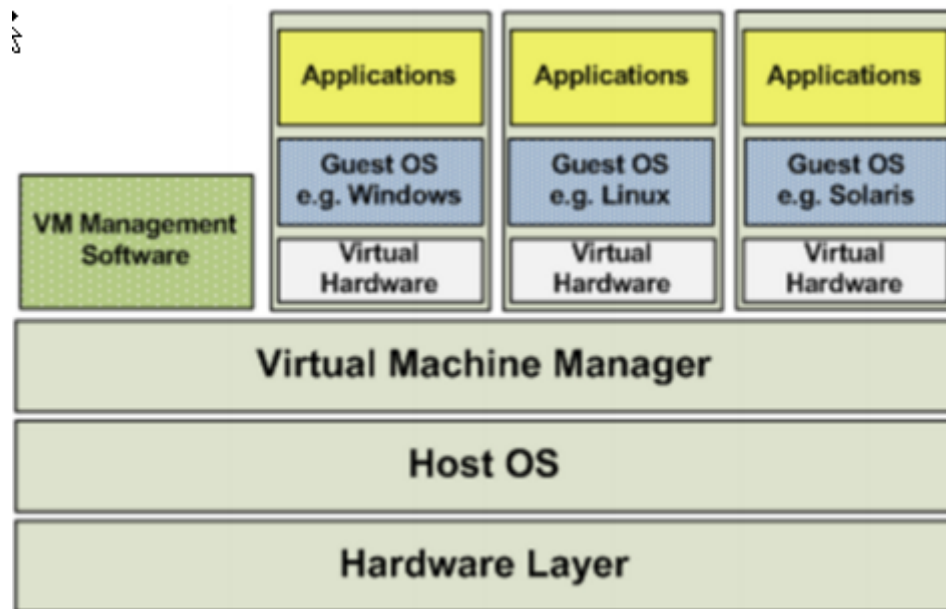
在实际的生产环境中，虚拟化技术主要用来解决高性能的物理硬件产能过剩和老的旧的硬件产能过低的重组重用，透明化底层物理硬件，从而最大化的利用物理硬件 对资源充分利用

虚拟化技术种类很多，例如：软件虚拟化、硬件虚拟化、内存虚拟化、网络虚拟化(vip)、桌面虚拟化、服务虚拟化、虚拟机等等。

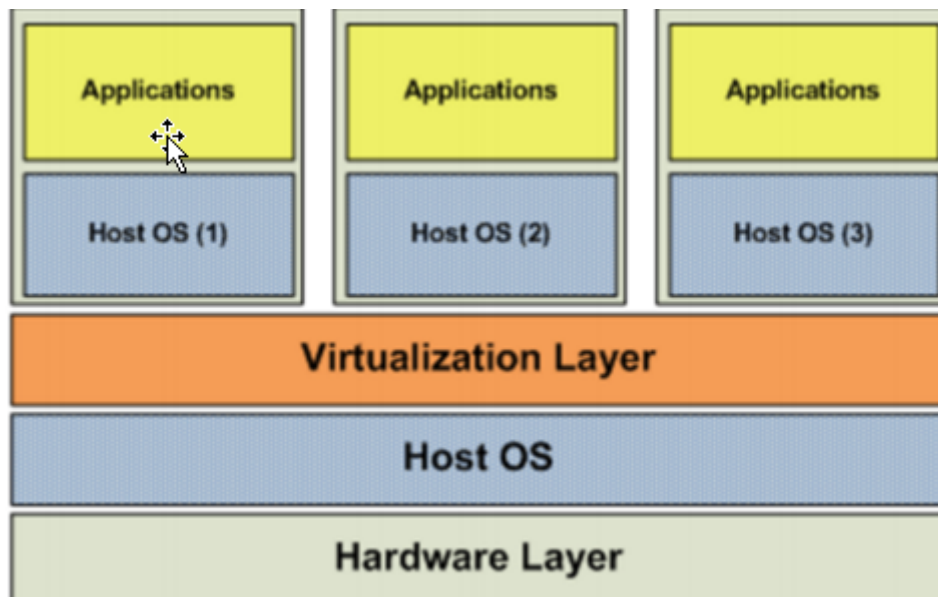
#### 1.1.2虚拟化种类

##### （1）全虚拟化架构

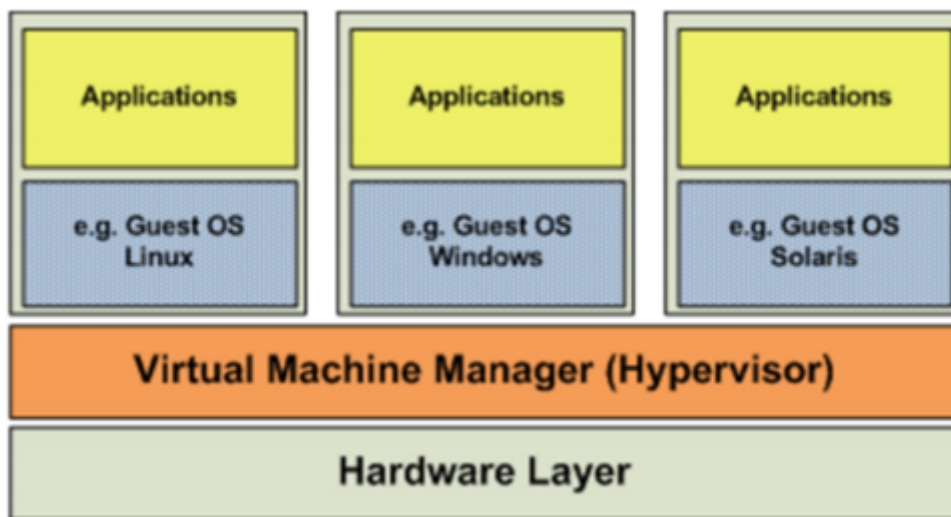
虚拟机的监视器（hypervisor）是类似于用户的应用程序运行在主机的OS之上，如VMware的workstation，这种虚拟化产品提供了虚拟的硬件。



(2) OS层虚拟化架构



(3) 硬件层虚拟化



硬件层的虚拟化具有高性能和隔离性，因为hypervisor直接在硬件上运行，有利于控制VM的OS访问硬件资源，使用这种解决方案的产品有VMware ESXi 和 Xen server Hypervisor是一种运行在物理服务器和操作系统之间的中间软件层,可允许多个操作系统和应用共享一套基础物理硬件，因此也可以看作是虚拟环境中的“元”操作系统，它可以协调访问服务器上的所有物理设备和虚拟机，也叫虚拟机监视器 (Virtual Machine Monitor, VMM) 。

Hypervisor是所有虚拟化技术的核心。当服务器启动并执行Hypervisor时，它会给每一台虚拟机分配适量的内存、CPU、网络和磁盘，并加载所有虚拟机的客户操作系统。宿主机Hypervisor是所有虚拟化技术的核心，软硬件架构和管理更高效、更灵活，硬件的效能能够更好地发挥出来。常见的产品有：VMware、KVM、Xen等等。Openstack。

## 1.2什么是Docker



### 1.2.1容器技术

docker用于部署系统 解决环境问题的这么一个容器技术。

在计算机的世界中，容器拥有一段漫长且传奇的历史。容器与管理程序虚拟化 (hypervisor virtualization, HV) 有所不同，管理程序虚拟化通过中间层将一台或者多台独立的机器虚拟运行与物理硬件之上，而容器则是直接运行在操作系统内核之上的用户空间。因此，容器虚拟化也被称为“操作系统级虚拟化”，容器技术可以让多个独立的用户空间运行在同一台宿主机上。

由于“客居”于操作系统，容器只能运行与底层宿主机相同或者相似的操作系统，这看起来并不是非常灵活。例如：可以在Ubuntu服务中运行RedhatEnterpriseLinux，但无法再Ubuntu服务器上运行MicrosoftWindows。

相对于彻底隔离的管理程序虚拟化，容器被认为是不安全的。而反对这一观点的人则认为，由于虚拟容器所虚拟的是一个完整的操作系统，这无疑增大了攻击范围，而且还要考虑管理程序层潜在的暴露风险。

尽管有诸多局限性，容器还是被广泛部署于各种各样的应用场合。在超大规模的多租户服务部署、轻量级沙盒以及对安全要求不太高的隔离环境中，容器技术非常流行。最常见的一个例子就是“权限隔离监牢”（chrootjail），它创建一个隔离的目录环境来运行进程。如果权限隔离监牢正在运行的进程被入侵者攻破，入侵者便会发现自己“身陷囹圄”，因为权限不足被困在容器所创建的目录中，无法对宿主机进一步破坏。

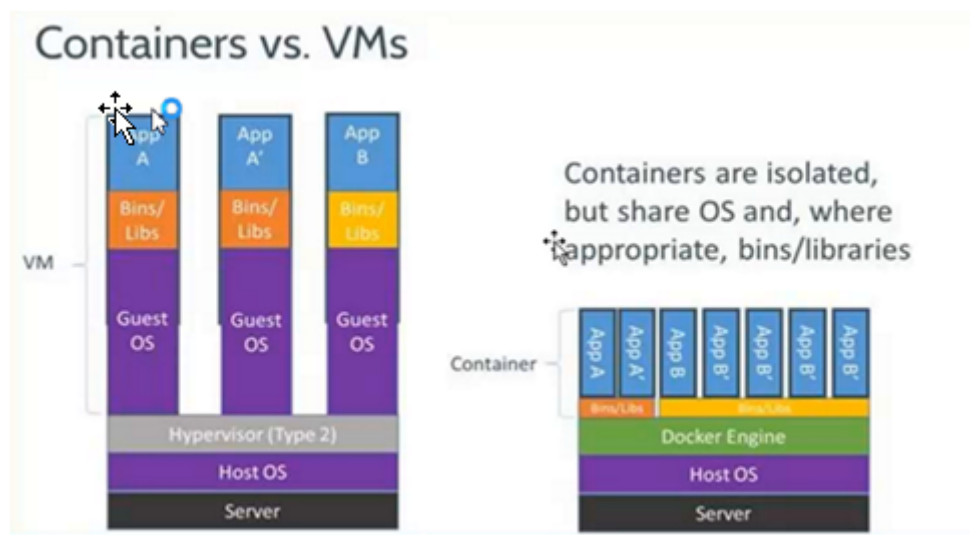
最新的容器技术引入了OpenVZ、Solaris Zones以及Linux容器（LXC）。使用这些新技术，容器不在仅仅是一个单纯的运行环境。在自己的权限类内，容器更像是一个完整的宿主机。对Docker来说，它得益于现代Linux特性，如控件组（controlgroup）、命名空间（namespace）技术，容器和宿主机之间的隔离更加彻底，容器有独立的网络和存储栈，还拥有自己的资源管理能力，使得同一台宿主机中的多个容器可以友好的共存。

容器被认为是精益技术，因为容器需要的开销有限。和传统虚拟化以及半虚拟化相比，容器不需要模拟层（emulationlayer）和管理层（hypervisorlayer），而是使用操作系统的系统调用接口。这降低了运行单个容器所需的开销，也使得宿主机中可以运行更多的容器。

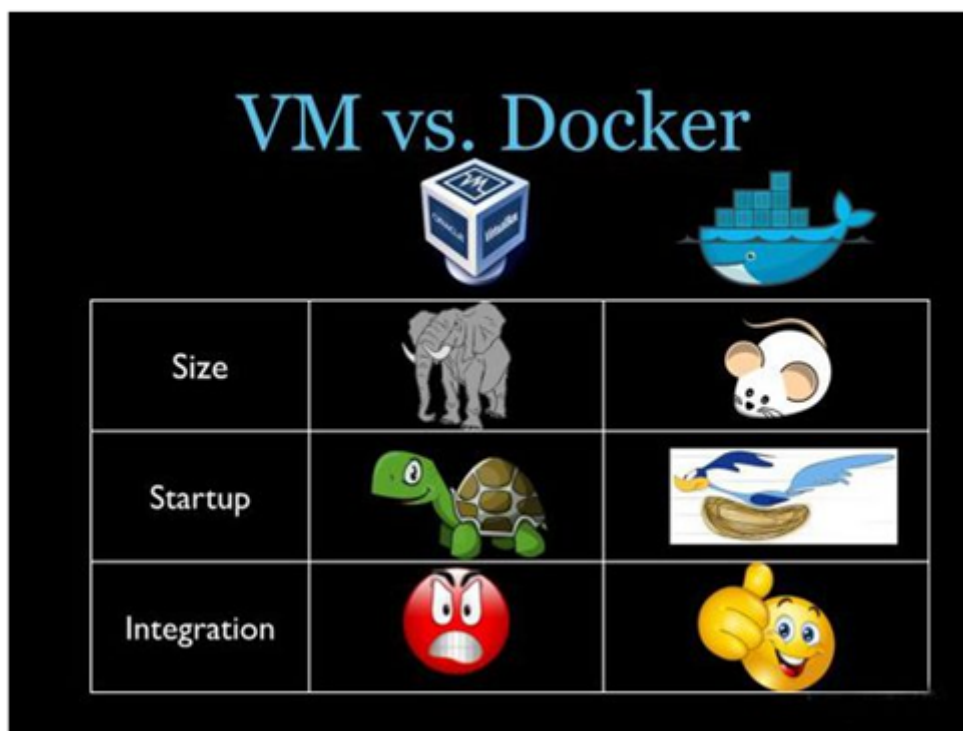
尽管有着光辉的历史，容器仍未得到广泛的认可。一个很重要的原因就是容器技术的复杂性：容器本身就比较复杂，不易安装，管理和自动化也很困难。而Docker就是为了改变这一切而生的。

## 1.2.2容器与虚拟机比较

### （1）本质上的区别



### （2）使用上的区别



虚拟机已死，容器才是未来。

### 1.2.3 Docker特点

(1) 上手快。

用户只需要几分钟，就可以把自己的程序“Docker化”。Docker依赖于“写时复制”（copy-on-write）模型，使修改应用程序也非常迅速，可以说达到“随心所欲，代码即改”的境界。

随后，就可以创建容器来运行应用程序了。大多数Docker容器只需要不到1秒中即可启动。由于去除了管理程序的开销，Docker容器拥有很高的性能，同时同一台宿主主机中也可以运行更多的容器，使用户尽可能的充分利用系统资源。

(2) 职责的逻辑分类

使用Docker，开发人员只需要关心容器中运行的应用程序，而运维人员只需要关心如何管理容器。Docker设计的目的是要加强开发人员写代码的开发环境与应用程序要部署的生产环境一致性。从而降低那种“开发时一切正常，肯定是运维的问题（测试环境都是正常的，上线后出了问题就归结为肯定是运维的问题）”

(3) 快速高效的开发生命周期

Docker的目标之一就是缩短代码从开发、测试到部署、上线运行的周期，让你的应用程序具备可移植性，易于构建，并易于协作。（通俗一点说，Docker就像一个盒子，里面可以装很多物件，如果需要这些物件的可以直接将该大盒子拿走，而不需要从该盒子中一件件的取。）

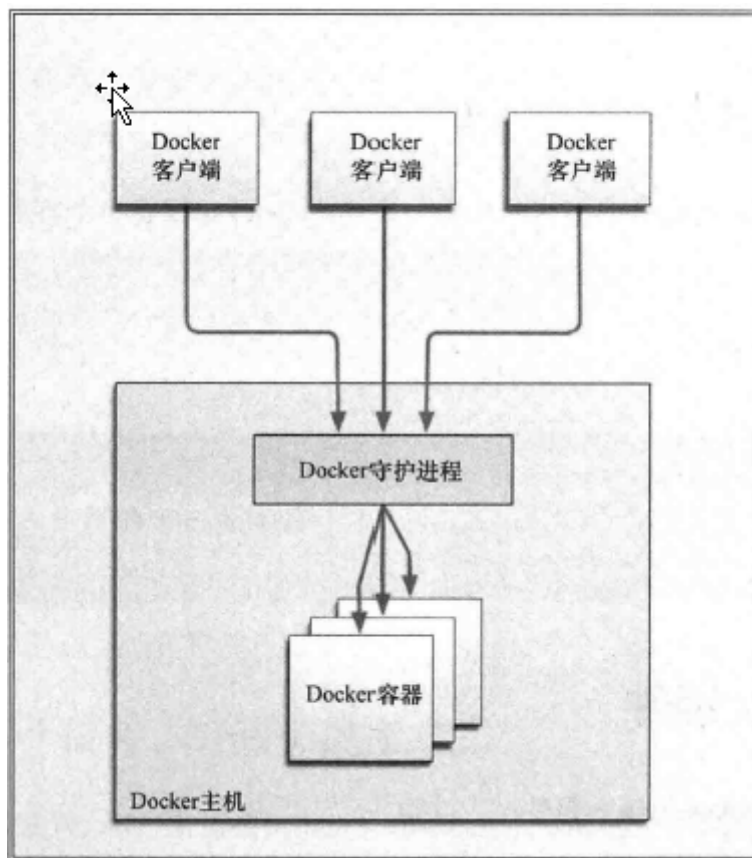
(4) 鼓励使用面向服务的架构

Docker还鼓励面向服务的体系结构和微服务架构。Docker推荐单个容器只运行一个应用程序或进程，这样就形成了一个分布式的应用程序模型，在这种模型下，应用程序或者服务都可以表示为一系列内部互联的容器，从而使分布式部署应用程序，扩展或调试应用程序都变得非常简单，同时也提高了程序的内存省性。（当然，可以在一个容器中运行多个应用程序）

## 1.3 Docker组件

### 1.3.1 Docker客户端和服务端

Docker是一个客户端-服务器（C/S）架构程序。Docker客户端只需要向Docker服务器或者守护进程发出请求，服务器或者守护进程将完成所有工作并返回结果。Docker提供了一个命令行工具Docker以及一整套RESTful API。你可以在同一台宿主机上运行Docker守护daemon进程和客户端，也可以从本地的Docker客户端连接到运行在另一台宿主机上的远程Docker守护进程。



### 1.3.2 Docker镜像

镜像是构建Docker的基石。用户基于镜像来运行自己的容器。镜像也是Docker生命周期中的“构建”部分。镜像是基于联合文件系统的一种层式结构，由一系列指令一步一步构建出来。例如：镜像可以是mysql tomcat redis

添加一个文件；

执行一个命令；

打开一个窗口。

也可以将镜像当作容器的“源代码”。镜像体积很小，非常“便携”，易于分享、存储和更新。

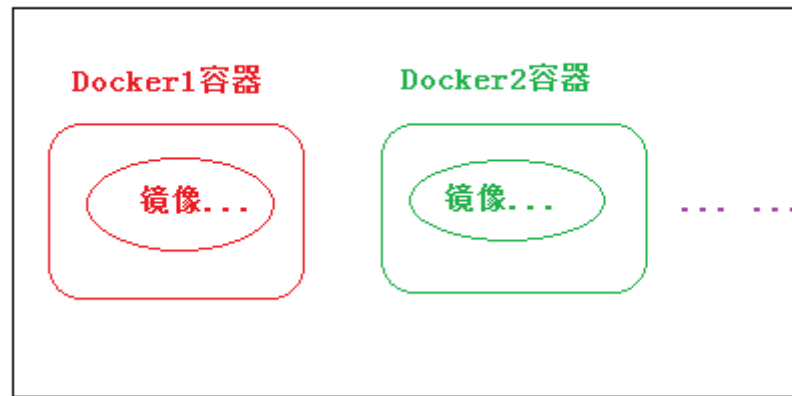
### 1.3.3 Registry（注册中心）

Docker用Registry来保存用户构建的镜像。Registry分为公共和私有两种。Docker公司运营公共的Registry叫做Docker Hub。用户可以在Docker Hub注册账号，分享并保存自己的镜像（说明：在Docker Hub下载镜像巨慢，可以自己构建私有的Registry）。

### 1.3.4 Docker容器

Docker可以帮助你构建和部署容器，你只需要把自己的应用程序或者服务打包放进容器即可。容器是基于镜像启动起来的，容器中可以运行一个或多个进程。我们可以认为，镜像是Docker生命周期中的构建或者打包阶段，而容器则是启动或者执行阶段。容器基于镜像启动，一旦容器启动完成后，我们就可以登录到容器中安装自己需要的软件或者服务。

## Linux OS



所以Docker容器就是：

- 一个镜像格式；
- 一些列标准操作；
- 一个执行环境。

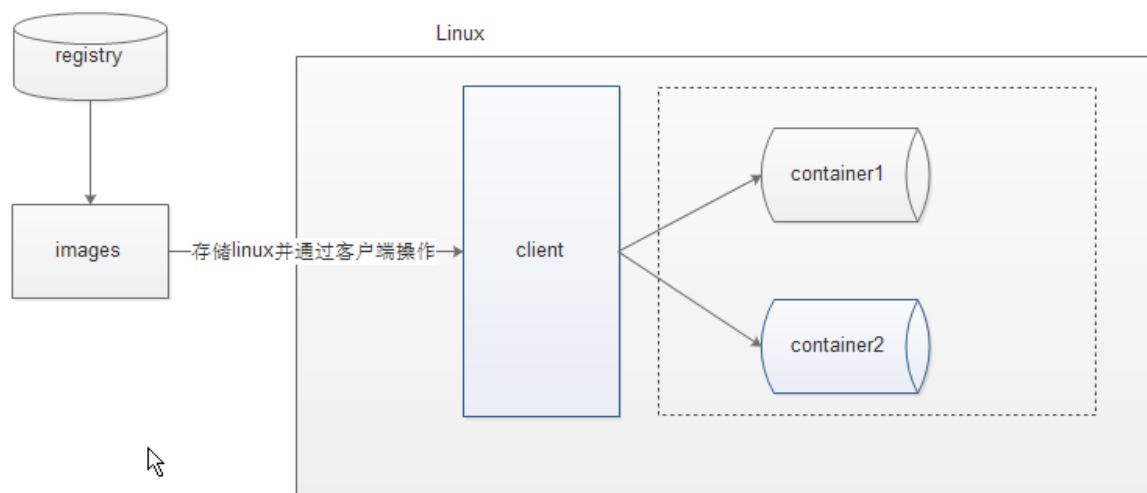
Docker借鉴了标准集装箱的概念。标准集装箱将货物运往世界各地，Docker将这个模型运用到自己的设计中，唯一不同的是：集装箱运输货物，而Docker运输软件。

和集装箱一样，Docker在执行上述操作时，并不关心容器中到底装了什么，它不管是web服务器，还是数据库，或者是应用程序服务器什么的。所有的容器都按照相同的方式将内容“装载”进去。

Docker也不关心你要把容器运到何方：我们可以在自己的笔记本中构建容器，上传到Registry，然后下载到一个物理的或者虚拟的服务器来测试，在把容器部署到具体的主机中。像标准集装箱一样，Docker容器方便替换，可以叠加，易于分发，并且尽量通用。

使用Docker，我们可以快速的构建一个应用程序服务器、一个消息总线、一套实用工具、一个持续集成（CI）测试环境或者任意一种应用程序、服务或工具。我们可以在本地构建一个完整的测试环境，也可以为生产或开发快速复制一套复杂的应用程序栈。

### 1.3.5 理解图



registry:中央注册中心

images:就是下载镜像文件

client:就是操作docker的客户端（命令）



container:就是docker容器 需要运行在docker服务中

## 2.Docker安装与启动

---

### 2.1安装环境说明

Docker官方建议在Ubuntu中安装，因为Docker是基于Ubuntu发布的，而且一般Docker出现的问题Ubuntu是最先更新或者打补丁的。在很多版本的CentOS中是不支持更新最新的一些补丁包的。

由于我们学习的环境都使用的是CentOS，因此这里我们将Docker安装到CentOS上。注意：这里建议安装在CentOS7.x以上的版本，在CentOS6.x的版本中，安装前需要安装其他很多的环境而且Docker很多补丁不支持更新。

### 2.2安装CentOS7

先在VMware中安装CENTOS系统 我们安装的是centos7

### 2.3安装Docker

使用yum命令在线安装

```
yum -y install docker
```

### 2.4安装后查看Docker版本

docker -v

### 2.5启动与停止Docker

systemctl命令是系统服务管理器指令，它是 service 和 chkconfig 两个命令组合。

- ☐ 启动docker：systemctl start docker
- ☐ 停止docker：systemctl stop docker
- ☐ 重启docker：systemctl restart docker
- ☐ 查看docker状态：systemctl status docker
- ☐ 开机启动：systemctl enable docker
  
- ☐ 查看docker概要信息：docker info
- ☐ 查看docker帮助文档：docker --help

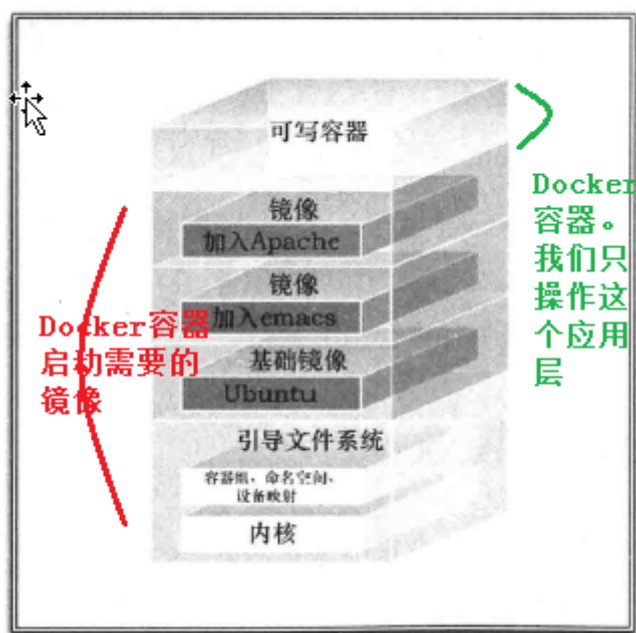
## 3.Docker镜像操作

---



## 3.1什么是Docker镜像

Docker镜像是由文件系统叠加而成（是一种文件的存储形式）。最底端是一个文件引导系统，即 bootfs，这很像典型的Linux/Unix的引导文件系统。Docker用户几乎永远不会和引导系统有什么交互。实际上，当一个容器启动后，它将会被移动到内存中，而引导文件系统则会被卸载，以留出更多的内存供磁盘镜像使用。Docker容器启动是需要的一些文件，而这些文件就可以称为Docker镜像。



## 3.2列出镜像

列出docker下的当前docker服务所在的系统里面所有镜像： docker images

```
[root@localhost ~]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mydockerfileimage	latest	71ef3e622362	6 weeks ago	200 MB
mysqlimage	latest	40aed1c9df94	6 weeks ago	372 MB
mycentosmuch	latest	bc90c60b0938	6 weeks ago	200 MB
tomcat2	1.0	81133c6f0747	6 weeks ago	200 MB
centos-much	latest	1b3e6b384808	6 weeks ago	200 MB
centos	1.0.0	1b3e6b384808	6 weeks ago	200 MB
docker.io/centos	latest	5182e96772bf	2 months ago	200 MB
myimagenginx	latest	eac79a640ab0	3 months ago	109 MB
mynginximage	latest	b75cbd5459b6	3 months ago	109 MB
docker.io/tomcat	7-jre7	1313cee27cdf	3 months ago	357 MB
docker.io/nginx	latest	3c5a05123222	3 months ago	109 MB
docker.io/redis	latest	71a81cb279e3	3 months ago	83.4 MB
docker.io/mysql	5	66bc0f66b7af	3 months ago	372 MB
docker.io/mysql	5.7	66bc0f66b7af	3 months ago	372 MB
docker.io/centos	7	49f7960eb7e4	4 months ago	200 MB

```
[root@localhost ~]#
```

- REPOSITORY: 镜像所在的仓库名称
- TAG: 镜像标签
- IMAGE ID: 镜像ID
- CREATED: 镜像的创建日期（不是获取该镜像的日期）
- SIZE: 镜像大小
- 这些镜像都是存储在Docker宿主机的/var/lib/docker目录下

为了区分同一个仓库下的不同镜像，Docker提供了一种称为标签（Tag）的功能。每个镜像在列出来时都带有一个标签，例如12.10、12.04等等。每个标签对组成特定镜像的一些镜像层进行标记（比如，标签12.04就是对所有Ubuntu12.04镜像层的标记）。这种机制使得同一个仓库中可以存储多个镜像。

我们在运行同一个仓库中的不同镜像时，可以通过在仓库名后面加上一个冒号和标签名来指定该仓库中的某一具体的镜像，例如

```
docker run --name custom_container_name -i -t docker.io/ubuntu:12.04 /bin/bash
```

表明从镜像Ubuntu:12.04启动一个容器，而这个镜像的操作系统就是Ubuntu:12.04。在构建容器时指定仓库的标签也是一个好习惯。

## 3.3搜索镜像

如果你需要从网络中查找需要的镜像，可以通过以下命令搜索: `docker search 镜像名称`.例如:

```
[root@localhost ~]# docker search tomcat
```

INDEX	NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
docker.io	docker.io/tomcat	Apache Tomcat is an open source implementa...	2091	[OK]	
docker.io	docker.io/tomee	Apache TomEE is an all-Apache Java EE cert...	58	[OK]	
docker.io	docker.io/dordoka/tomcat	Ubuntu 14.04, Oracle JDK 8 and Tomcat 8 ba...	49		[OK]
docker.io	docker.io/davidcaste/alpine-tomcat	Apache Tomcat 7/8 using Oracle Java 7/8 wi...	31		[OK]
docker.io	docker.io/bitnami/tomcat	Bitnami Tomcat Docker Image	25		[OK]
docker.io	docker.io/consol/tomcat-7.0	Tomcat 7.0.57, 8080, "admin/admin"	16		[OK]
docker.io	docker.io/cloudesire/tomcat	Tomcat server, 6/7/8	15		[OK]
docker.io	docker.io/tutum/tomcat	Base docker image to run a Tomcat applicat...	11		
docker.io	docker.io/meirwa/spring-boot-tomcat-mysql-app	a sample spring-boot app using tomcat and ...	10		[OK]
docker.io	docker.io/aallam/tomcat-mysql	Debian, Oracle JDK, Tomcat & MySQL	8		[OK]
docker.io	docker.io/jeanblanchard/tomcat	Minimal Docker image with Apache Tomcat	8		
docker.io	docker.io/maluuba/tomcat7-java8	Tomcat7 with java8.	3		
docker.io	docker.io/rightctrl/tomcat	CentOS , Oracle Java, tomcat application s...	3		[OK]
docker.io	docker.io/amd64/tomcat	Apache Tomcat is an open source implementa...	2		
docker.io	docker.io/arm64v8/tomcat	Apache Tomcat is an open source implementa...	2		
docker.io	docker.io/99taxi/tomcat7	Tomcat7	1		[OK]
docker.io	docker.io/camptocamp/tomcat-logback	Docker image for tomcat with logback integ...	1		[OK]
docker.io	docker.io/fabric8/tomcat-8	Fabric8 Tomcat 8 Image	1		[OK]
docker.io	docker.io/jelastic/tomcat	An image of the Tomcat Java application se...	1		
docker.io	docker.io/primetoninc/tomcat	Apache tomcat 8.5, 8.0, 7.0	1		[OK]
docker.io	docker.io/cfje/tomcat-resource	Tomcat Concourse Resource	0		
docker.io	docker.io/oobsoi/tomcat8	Testing CI Jobs with different names.	0		
docker.io	docker.io/picoded/tomcat7	tomcat7 with jre8 and MANAGER USER / MANAG...	0		[OK]
docker.io	docker.io/s390x/tomcat	Apache Tomcat is an open source implementa...	0		
docker.io	docker.io/swisstopo/service-print-tomcat	backend tomcat for service-print "the true...	0		

```
[root@localhost ~]#
```

- NAME: 仓库名称
- DESCRIPTION: 镜像描述
- STARS: 用户评价，反应一个镜像的受欢迎程度
- OFFICIAL: 是否官方
- AUTOMATED: 自动构建，表示该镜像由Docker Hub自动构建流程创建的

## 3.4拉取镜像

### 3.4.1从Docker Hub拉取

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

Docker镜像首页，包括官方镜像和其它公开镜像。Docker Hub上最受欢迎的10大镜像（通过Docker registry API获取不了镜像被pull的个数，只能通过镜像的stars数量，来衡量镜像的流行度。毫无疑问，拥有最高stars数量的库都是官方库）。由于国情的原因，国内下载 Docker HUB 官方的相关镜像比较慢，可以使用国内（docker.io）的一些镜像加速器，镜像保持和官方一致，关键是速度快，推荐使用。

 <b>nginx</b> official	9.9K STARS	10M+ PULLS
 <b>alpine</b> official	4.4K STARS	10M+ PULLS
 <b>busybox</b> official	1.4K STARS	10M+ PULLS
 <b>redis</b> official	5.9K STARS	10M+ PULLS
 <b>httpd</b> official	2.1K STARS	10M+ PULLS
 <b>mongo</b>	5.1K	10M+

### Mirror与Private Registry的区别:

Private Registry（私有仓库）是开发者或者企业自建的镜像存储库，通常用来保存企业内部的 Docker 镜像，用于内部开发流程和产品的发布、版本控制。

Mirror是一种代理中转服务，我们(比如daocloud)提供的Mirror服务，直接对接Docker Hub的官方 Registry。Docker Hub 上有数以十万计的各类 Docker 镜像。

在使用Private Registry时，需要在Docker Pull 或Dockerfile中直接键入Private Registry 的地址，通常这样会导致与 Private Registry 的绑定，缺乏灵活性。

使用 Mirror 服务，只需要在 Docker 守护进程（Daemon）的配置文件中加入 Mirror 参数，即可在全局范围内透明的访问官方的 Docker Hub，避免了对 Dockerfile 镜像引用来源的修改。

### 拉取镜像

```
docker pull centos:7
```

目前国内访问docker hub速度上有点尴尬，使用docker Mirror势在必行。现有国内提供docker镜像加速服务的商家有不少，下面重点ustc镜像。

### 3.4.2 ustc的镜像

ustc是老旧的linux镜像服务提供者了，还在遥远的ubuntu 5.04版本的时候就在用。ustc的docker镜像加速器速度很快。ustc docker mirror的优势之一就是不需要注册，是真正的公共服务。

<https://lug.ustc.edu.cn/wiki/mirrors/help/docker>

步骤：

(1) 编辑该文件：vi /etc/docker/daemon.json

PS: 如果该文件不存在就手动创建；另外有可能如果没有vim 命令则使用vi命令即可。

(2) 在该文件中输入如下内容：

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

(3) 注意：一定要重启docker服务，如果重启docker后无法加速，可以重新启动OS ,然后通过docker pull命令下载镜像。

```
service docker restart
```

```
[root@localhost ~]# service docker restart
Redirecting to /bin/systemctl restart docker.service
[root@localhost ~]#
```

## 3.5 删除镜像

删除镜像方式1：根据仓库的名称（镜像的名称）来删除 还可以使用image\_id来进行删除。

1、 docker rmi \$IMAGE\_ID: 删除指定镜像

删除镜像方式2：

2、 docker rmi `docker images -q`: 删除所有镜像

```
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mydockerfileimage    latest             71ef3e622362       6 weeks ago        200 MB
mysqlimage           latest             40a8d1c9df94       6 weeks ago        372 MB
mycentosmuch         latest             bc90c60b0938       6 weeks ago        200 MB
tomcat2              1.0               81133c6f0747       6 weeks ago        200 MB
centos-much          latest             1b3e6b384808       6 weeks ago        200 MB
centos                1.0.0             1b3e6b384808       6 weeks ago        200 MB
docker.io/centos      latest             5182e96772bf       2 months ago       200 MB
myimagenginx         latest             eac79a640ab0       3 months ago       109 MB
mynginximage         latest             b75cbd5459b6       3 months ago       109 MB
docker.io/tomcat      7-jre7            1313cee27cdf       3 months ago       357 MB
docker.io/nginx       latest             3c5a05123222       3 months ago       109 MB
docker.io/redis       latest             71a81cb279e3       3 months ago       83.4 MB
docker.io/mysql       5                 66bc0f66b7af       3 months ago       372 MB
docker.io/mysql       5.7              66bc0f66b7af       3 months ago       372 MB
docker.io/centos      7                 49f7960eb7e4       4 months ago       200 MB
[root@localhost ~]# docker rmi 71ef3e622362
```

## 4.Docker容器操作

### 4.1查看容器

- 查看正在运行容器：

```
docker ps
```

- 查看所有的容器（启动过的历史容器）

```
docker ps -a
```

```
[root@localhost ~]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
86d3ade528a9	mydockerfileimage	"/bin/bash"	6 weeks ago	Exited (137)	4 weeks ago	mydockerfilecentos
dc91e3d658d4	mycentosmuch	"/bin/bash"	6 weeks ago	Exited (137)	4 weeks ago	mycentosmuchmuch
2bd8e928a63f	mysql:5	"docker-entrypoint..."	6 weeks ago	Exited (137)	4 weeks ago	mysqlceshi
0839798ac5c3	centos:7	"/bin/bash"	6 weeks ago	Exited (137)	4 weeks ago	mycentos004
eb0b82e2edd2	centos:7	"/bin/bash"	6 weeks ago	Exited (137)	4 weeks ago	mycentos003
557eb754580c	centos:7	"/bin/bash"	6 weeks ago	Exited (0)	6 weeks ago	mycentos001
06a925989f4	tomcat:1.0	"/bin/bash"	6 weeks ago	Exited (0)	6 weeks ago	competent_ardinghell
0316ed6b86fb	centos-much	"/bin/bash"	6 weeks ago	Exited (0)	6 weeks ago	naughty_ramanujan
ef2673d6ab02	nginx	"nginx -g 'daemon ...'"	3 months ago	Exited (0)	3 months ago	mynginx1
c3cd619ae9c1	mysql:5.7	"docker-entrypoint..."	3 months ago	Exited (0)	3 months ago	mysql1
26d74856bb7d	centos:7	"/bin/bash"	3 months ago	Exited (137)	3 months ago	mycentos10
c868c2b4ecf0	centos:7	"/bin/bash"	3 months ago	Exited (137)	3 months ago	mycentos8
14b2c8d82deb	centos:7	"/bin/bash"	3 months ago	Exited (0)	3 months ago	mycentos7
f299f0889d7	mynginximage	"nginx -g 'daemon ...'"	3 months ago	Exited (0)	3 months ago	othernginx
894e32a267a2	nginx	"nginx -g 'daemon ...'"	3 months ago	Exited (0)	3 months ago	mynginx
4771a22869cf	tomcat:7-jre7	"catalina.sh run"	3 months ago	Exited (137)	3 months ago	mytomcat7
a31025ff39a9	mysql:5.7	"docker-entrypoint..."	3 months ago	Exited (137)	3 months ago	pinyougou_mysql
f87ad3e820fe	mysql:5	"docker-entrypoint..."	3 months ago	Exited (1)	3 months ago	mymysql5
c3f366a6e7de	centos:7	"/bin/bash"	3 months ago	Exited (137)	6 weeks ago	mycentos6
916266af2771	centos:7	"/bin/bash"	3 months ago	Exited (137)	3 months ago	mycentos5
d755d0b094f6	centos:7	"/bin/bash"	3 months ago	Exited (137)	3 months ago	mycentos4
298b6b6c1af4	centos:7	"/bin/bash"	3 months ago	Exited (0)	3 months ago	mycentos3
af3380c1b1ab	centos:7	"/bin/bash"	3 months ago	Exited (137)	3 months ago	mycentos02
48641d18a8d6	centos:7	"/bin/bash"	3 months ago	Exited (0)	3 months ago	mycentos01

```
[root@localhost ~]#
```

- 查看最后一次运行的容器：

```
docker ps -l
```

```
[root@localhost ~]# docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
86d3ade528a9	mydockerfileimage	"/bin/bash"	6 weeks ago	Exited (137)	4 weeks ago	mydockerfilecentos

```
[root@localhost ~]#
```

- 查看停止的容器

```
docker ps -f status=exited
```

## 4.2 创建与启动容器

l 创建容器常用的参数说明：

l 创建容器命令：docker run

l -i：表示运行容器

l -t：表示容器启动后会进入其命令行。加入这两个参数后，容器创建就能登录进去。即分配一个伪终端。

l --name：为创建的容器命名。

l -v：表示目录映射关系（前者是宿主机目录，后者是映射到宿主机上的目录），可以使用多个 -v 做多个目录或文件映射。注意：最好做目录映射，在宿主机上做修改，然后共享到容器上。

l -d：在run后面加上-d参数,则会创建一个守护式容器在后台运行（这样创建容器后不会自动登录容器，如果只加-i -t两个参数，创建后就会自动进去容器）。

l -p：表示端口映射，前者是宿主机端口，后者是容器内的映射端口。可以使用多个 -p 做多个端口映射

## 4.1.1创建交互式容器

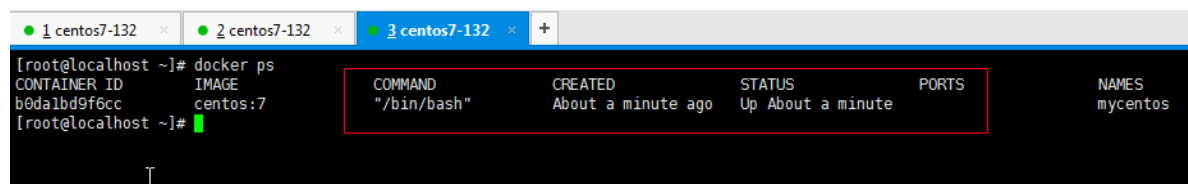
创建一个交互式容器并取名为mycentos

```
docker run -it --name=mycentos centos:7 /bin/bash
```

```
[root@localhost ~]# docker run -it --name=mycentos centos:7 /bin/bash
[root@b0dalbd9f6cc /]#
```

开启另外一个终端来查看状态:

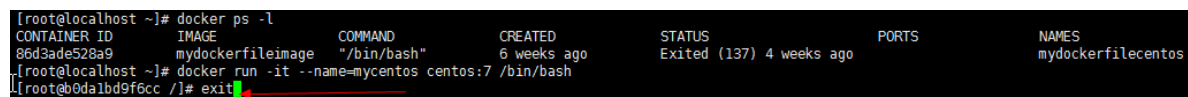
```
docker ps
```



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b0dalbd9f6cc	centos:7	"/bin/bash"	About a minute ago	Up About a minute		mycentos

退出当前容器:

```
exit
```



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
86d3ade528a9	mydockerfileimage	"/bin/bash"	6 weeks ago	Exited (137) 4 weeks ago		mydockerfilecentos

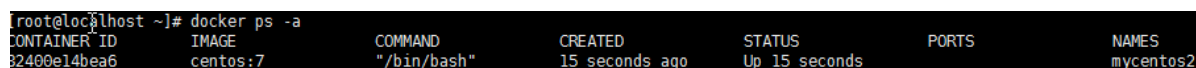
然后用ps -a 命令查看发现该容器也随之停止:

## 4.1.2守护式容器

创建一个守护式容器: 如果对于一个需要长期运行的容器来说, 我们可以创建一个守护式容器

命令如下 (容器名称不能重复):

```
docker run -di --name=mycentos2 centos:7
```



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
32400e14bea6	centos:7	"/bin/bash"	15 seconds ago	Up 15 seconds		mycentos2

登录守护式容器:

docker exec -it container\_name (或者container\_id)/bin/bash (exit退出时, 容器不会停止)

命令如下:

```
docker exec -it mycentos2 /bin/bash
```

```
[root@localhost ~]# docker exec -it mycentos2 /bin/bash
[root@32400e14bea6 /]#
```

## 4.3停止与启动容器

- 停止正在运行的容器: `docker stop $CONTAINER_NAME/ID`

```
docker stop mycentos2
```

```
[root@localhost ~]# docker stop mycentos2
mycentos2
[root@localhost ~]#
```

- 启动已运行过的容器: `docker start $CONTAINER_NAME/ID`

```
docker start mycentos2
```

```
[root@localhost ~]# docker start mycentos2
mycentos2
[root@localhost ~]#
```

## 4.4文件拷贝

如果我们需要将文件拷贝到容器内可以使用cp命令:

```
docker cp 需要拷贝的文件或目录 容器名称:容器目录
```

也可以将文件从容器内拷贝出来

```
docker cp 容器名称:容器目录 需要拷贝的文件或目录
```

```
[root@localhost ~]# ll
总用量 188184
-rw-r--r--. 1 root root      0 10月 19 17:10 a
drwxr-xr-x. 2 root root    37 10月 19 11:42 apps
-rw-r--r--. 1 root root 192699649 10月 19 11:44 jdk-8u152-linux-i586.tar.gz
[root@localhost ~]# docker cp a mycentos2:/usr/local/a
```

解释: cp 当前目录下的a文件到 容器中的/usr/local/下的a文件

```
[root@localhost ~]# docker exec -it mycentos2 /bin/bash
[root@32400e14bea6 /]# cd /usr/local/
[root@32400e14bea6 local]# ll
```

```
root@32400e14bea6 local]# ll
total 0
-rw-r--r--. 1 root root  0 Oct 19 09:10 a
drwxr-xr-x. 2 root root  6 Apr 11 2018 bin
drwxr-xr-x. 2 root root  6 Apr 11 2018 etc
drwxr-xr-x. 2 root root  6 Apr 11 2018 games
drwxr-xr-x. 2 root root  6 Apr 11 2018 include
drwxr-xr-x. 2 root root  6 Apr 11 2018 lib
drwxr-xr-x. 2 root root  6 Apr 11 2018 lib64
drwxr-xr-x. 2 root root  6 Apr 11 2018 libexec
drwxr-xr-x. 2 root root  6 Apr 11 2018/sbin
drwxr-xr-x. 5 root root 49 May 31 18:02 share
drwxr-xr-x. 2 root root  6 Apr 11 2018 src
```



## 4.5 目录挂载（映射）

我们可以在创建容器的时候，将宿主机的目录与容器内的目录进行映射，这样我们就可以通过修改宿主机某个目录的文件从而去影响容器里所对应的目录。

创建容器 添加-v参数 后边为 宿主机目录:容器目录

创建容器 并挂载宿主机目录 到容器中的目录下：

```
docker run -di -v /usr/local/myhtml:/usr/local/myhtml --name=mycentos3 centos:7
```

如果你共享的是多级的目录，可能会出现权限不足的提示。

这是因为CentOS7中的安全模块selinux把权限禁掉了，我们需要添加参数--privileged=true来解决挂载的目录没有权限的问题。

```
docker run -di --privileged=true -v /root/test:/usr/local/test --name=mycentos4 centos:7
```

## 4.6 查看容器IP地址

我们可以通过以下命令查看容器运行的各种数据：

```
docker inspect mycentos2
```

也可以直接执行下面的命令直接输出IP地址：

```
docker inspect --format='{{.NetworkSettings.IPAddress}}' mycentos2
```

## 4.7 删除容器

- 删除指定的容器：这个命令只能删除已经关闭的容器，不能删除正在运行的容器

```
docker rm $CONTAINER_ID/NAME
```

- 删除所有的容器：

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

或者：

```
[root@localhost ~]# docker rm $(docker ps -aq)
```

## 5. 部署应用

## 5.1MySQL部署

### 5.1.1拉取MySQL镜像

```
docker pull mysql
```

查看镜像:

```
[root@localhost ~]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mydockerfileimage	latest	71ef3e622362	6 weeks ago	200 MB
mysqlimage	latest	40aed1c9df94	6 weeks ago	372 MB
mycentosmuch	latest	bc90c60b0938	6 weeks ago	200 MB
tomcat2	1.0	81133c6f0747	6 weeks ago	200 MB
centos-much	latest	1b3e6b384808	6 weeks ago	200 MB
centos	1.0.0	1b3e6b384808	6 weeks ago	200 MB
docker.io/centos	latest	5182e96772bf	2 months ago	200 MB
myimagenginx	latest	eac79a640ab0	3 months ago	109 MB
mynginximage	latest	b75cbd5459b6	3 months ago	109 MB
docker.io/tomcat	7-jre7	1313cee27cdf	3 months ago	357 MB
docker.io/nginx	latest	3c5a05123222	3 months ago	109 MB
docker.io/redis	latest	71a81cb279e3	3 months ago	83.4 MB
docker.io/mysql	5	66bc0f66b7af	3 months ago	372 MB
docker.io/mysql	5.7	66bc0f66b7af	3 months ago	372 MB
docker.io/centos	7	49f7960eb7e4	4 months ago	200 MB

### 5.1.2创建MySQL容器

```
docker run -di --name=pinyougou_mysql -p 33306:3306 -e  
MYSQL_ROOT_PASSWORD=123456 mysql:5.7
```

-p 代表端口映射, 格式为 宿主机映射端口:容器运行端口

-e 代表添加环境变量 MYSQL\_ROOT\_PASSWORD是root用户的登陆密码

### 5.1.3进入MySQL容器

- 进入容器中

```
docker exec -it pinyougou_mysql /bin/bash
```

- 登录mysql

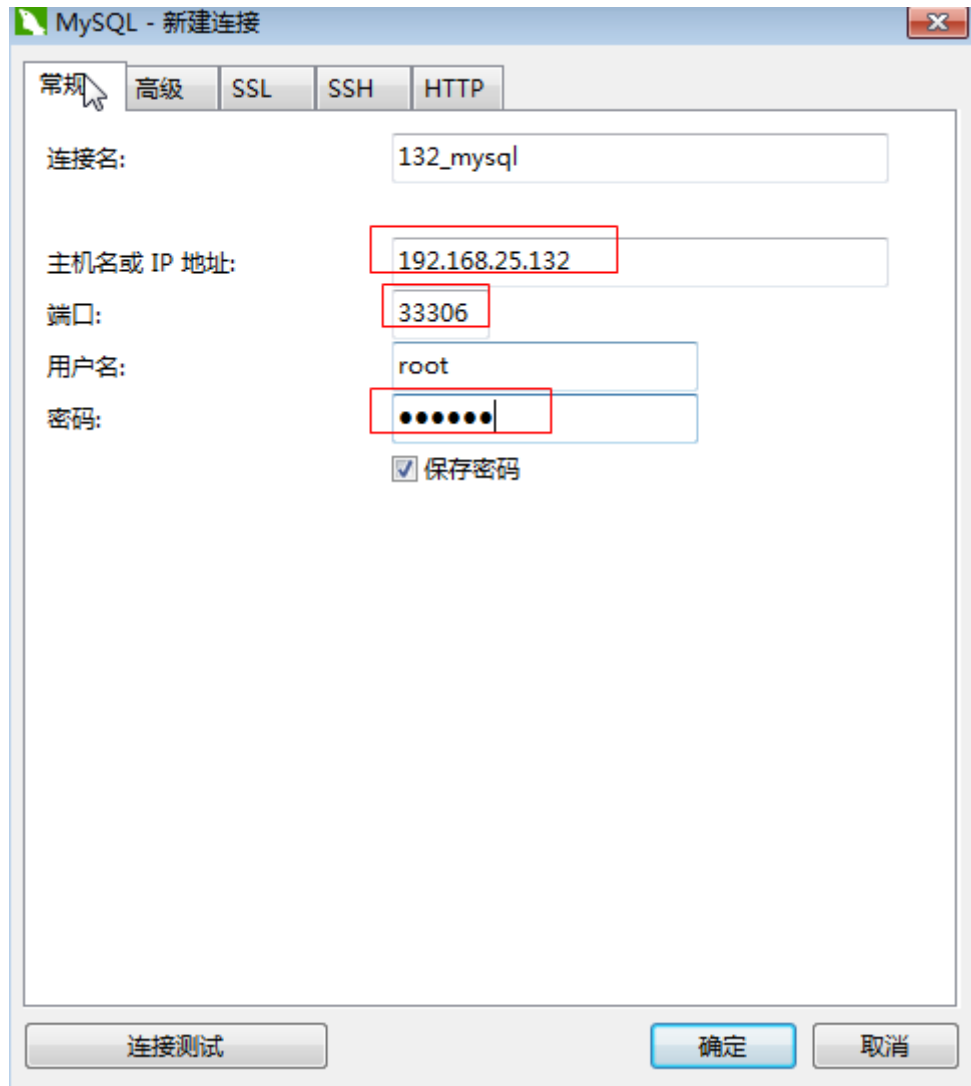
```
mysql -u root -p
```

- 授权允许远程登录

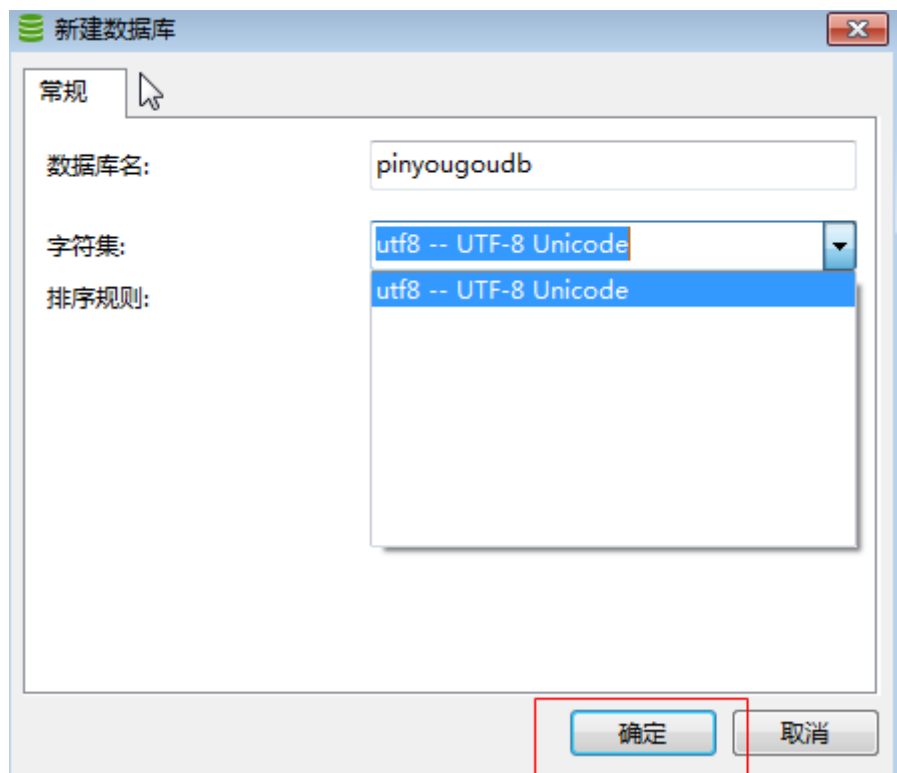
```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '123456' WITH GRANT  
OPTION;
```

## 5.1.4远程登陆MySQL

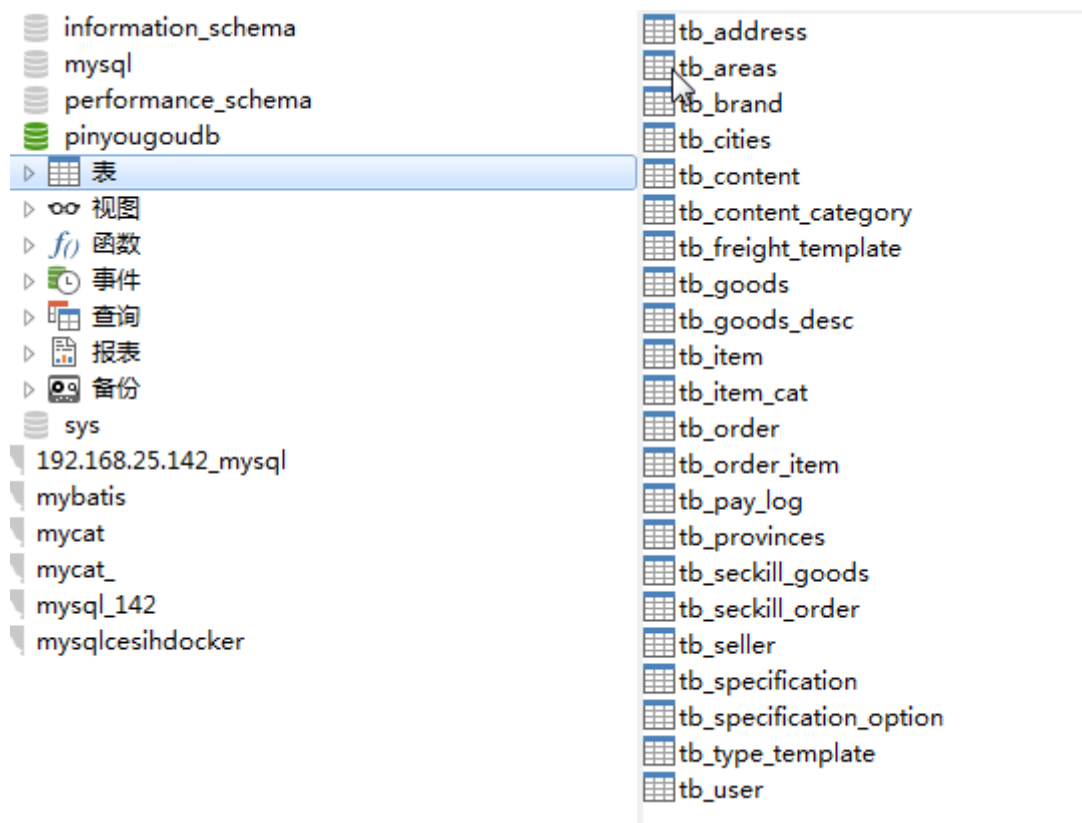
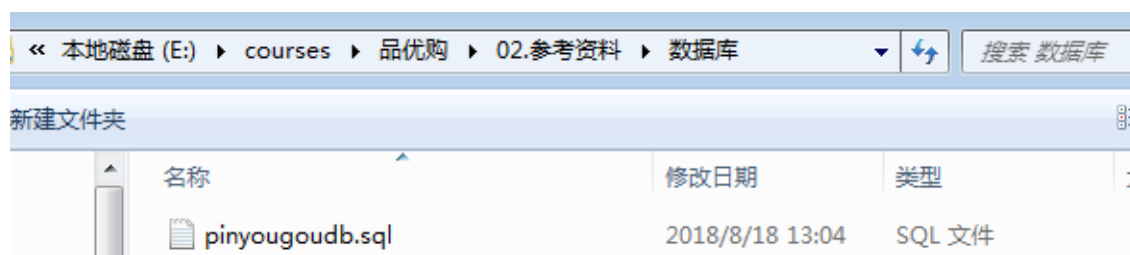
- 我们在我们本机的电脑上去连接虚拟机Centos中的Docker容器，这里192.168.25.132是虚拟机操作系统的IP。



- 在本地客户端执行品优购的数据库脚本 (pinyougoudb.sql)  
创建数据库实例:



导入脚本:



查看容器中的数据是否已经导入成功：

```
[root@localhost ~]# docker exec -it pinyougou_mysql /bin/bash
root@f1333a4c76dc:/# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.7.22 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| pinyougodb |
| sys |
+-----+
5 rows in set (0.00 sec)
```

## 5.2 tomcat部署

### 5.2.1拉取tomcat镜像

```
docker pull tomcat:7-jre8
```

### 5.2.2创建tomcat容器

```
docker run -di --name=pinyougou_tomcat -p 9100:8080 -v
/usr/local/myhtml:/usr/local/tomcat/webapps --privileged=true tomcat:7-jre8
```

### 5.2.3部署web应用

修改cas系统的配置文件，修改数据库连接的url，注意修改jdbc的URL地址要和容器的地址一致。

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
    p:driverClass="com.mysql.jdbc.Driver"
    p:jdbcUrl="jdbc:mysql://172.17.0.6:3306/pinyougodb?
characterEncoding=utf8"
    p:user="root"
    p:password="123456" />
```

测试：地址栏输入：<http://192.168.25.132:9100/cas/login>

## 5.3 Nginx部署

### 5.3.1拉取Nginx镜像

```
docker pull nginx
```

### 5.3.2创建Nginx容器

```
docker run -di --name=pinyougou_nginx -p 80:80 nginx
```

### 5.3.3测试Nginx

浏览器地址栏输入: <http://192.168.25.132>

### 5.3.4配置反向代理

官方的nginx镜像,nginx配置文件nginx.conf 在/etc/nginx/目录下。在容器内编辑配置文件不方便,我们可以先将配置文件从容器内拷贝到宿主机,编辑修改后再拷贝回去。

- 从容器拷贝配置文件到宿主机

```
[root@localhost ~]# docker cp pinyougou_nginx:/etc/nginx/nginx.conf nginx.conf
```

- 修改nginx.conf 添加如下配置:

```
upstream tomcat-cas {  
    server 172.17.0.7:8080;  
}  
server {  
    listen 80;  
    server_name passport.pinyougou.com;  
    location / {  
        proxy_pass http://tomcat-cas;  
        index index.html index.htm;  
    }  
}
```

- 将修改后的配置文件拷贝到容器

```
docker cp nginx.conf pinyougou_nginx:/etc/nginx/nginx.conf
```

- 重新启动容器

```
docker restart pinyougou_nginx
```

- 在本地hosts文件中添加域名和ip地址的映射关系

```
192.168.25.132 passport.pinyougou.com
```

浏览器测试: <http://passport.pinyougou.com/cas/login>

## 5.4 Redis部署

### 5.4.1拉取Redis镜像

```
docker pull redis
```

### 5.4.2创建Redis容器

```
docker run -di --name=pinyougou_redis -p 6379:6379 redis
```

### 5.4.3客户端测试

本地安装一个redis的客户端 连接即可

## 6.备份与迁移

### 6.1容器保存为镜像

我们可以通过以下命令将容器保存为镜像

```
docker commit pinyougou_nginx mynginx
```

pinyougou\_nginx是容器名称

mynginx是新的镜像名称

此镜像的内容就是你当前容器的内容，接下来你可以用此镜像再次运行新的容器

### 6.2镜像备份

```
docker save -o mynginx.tar mynginx
```

-o 输出到的文件

执行后，运行ls命令即可看到打成的tar包.

### 6.3镜像恢复与迁移

首先我们先删除掉mynginx镜像,然后执行命令进行恢复

```
docker load -i mynginx.tar
```

-i 输入的文件

执行后再次查看镜像，可以看到镜像已经恢复

再创建容器。

## 7.dockerfile(了解)

Dockerfile是由一系列命令和参数构成的脚本，这些命令应用于基础镜像并最终创建一个新的镜像。它们简化了从头到尾的流程并极大的简化了部署工作。Dockerfile从FROM命令开始，紧接着跟随者各种方法，命令和参数。其产出为一个新的可以用于创建容器的镜像。



更多语法参考：（官网手册的翻译）

<https://blog.csdn.net/guyue35/article/details/53891862>

## 7.1 dockerfile的demo

- 在root下创建demo目录

```
mkdir demo
cd demo
```

- 在demo 目录下创建 a文件

```
vim a
```

- 在某~/demo下创建Dockerfile文件

```
vim Dockerfile
```

内容为：

```
# my dockerfile ljh
FROM centos
MAINTAINER ljh
WORKDIR /root/workdir
RUN touch te
ADD a b
ENV key1 "hello"
```

语法解释：

# ：代表注释

FROM centos ：代表依据基本的镜像来创建

MAINTAINER ljh ：代表就是作者是谁

WORKDIR /root/workdir ：代表就是创建容器时进入工作的目录是容器中的/root/workdir目录

RUN touch te ：代表就是RUN 运行命令 运行一个创建空文件te

COPY ["HI","."] ：代表从宿主系统中复制HI 文件到容器系统中工作目录中的当前路径下

ADD a b ：代表从宿主主机所在Dockerfile文件的目录下Copy 文件A 到容器中的b文件 b文件的目录为工作目录下。

ENV key1 "hello" ：定义linux中的环境变量。如下：

定义一个：key value

定义多个：key=value key2=value2

- 创建自定义镜像

```
docker build -t mycentos .
```

语法解释：

docker build ：表示通过Dockerfile文件来创建镜像

-t mycentos 表示 给与镜像的名称和版本 为：mycentos:lasted (lasted可以不写)

. 表示从当前目录下进行加载Dockerfile文件

```

[root@localhost demo]# ll
总用量 8
-rw-r--r--. 1 root root 370 10月 19 23:08 a
-rw-r--r--. 1 root root 107 10月 19 23:14 Dockerfile
[root@localhost demo]# docker build -t mycustomcentos .
Sending build context to Docker daemon 3.072 kB
Step 1/6 : FROM centos
----> 5182e96772bf
Step 2/6 : MAINTAINER ljh
----> Using cache
----> f96fbe42cd44
Step 3/6 : WORKDIR /root/workdir
----> Using cache
----> 011338d95a94
Step 4/6 : RUN touch te
----> Using cache
----> 0f00eb768809
Step 5/6 : ADD a b
----> 4eld77fdc682
Removing intermediate container 0ca2897b5877
Step 6/6 : ENV key1 "hello"
----> Running in 6a9bfa6a031d
----> 615f90dad61d
Removing intermediate container 6a9bfa6a031d
Successfully built 615f90dad61d
[root@localhost demo]#

```

- 查看是否打包镜像成功

```

[root@localhost demo]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mycustomcentos      latest             615f90dad61d       40 seconds ago     200 MB

```

- 测试创建容器:

```
docker run -di --name=mycentosheh mycustomcentos
```

- 连接容器

```
[root@localhost demo]# docker exec -it mycentosheh /bin/bash
```

注意：进入目录即为：/root/workdir

```

[root@eff5cd1a0e30 workdir]# pwd
/root/workdir
[root@eff5cd1a0e30 workdir]#

```

在目录下有 b文件和 te文件

```
[root@eff5cd1a0e30 workdir]# ll
total 4
-rw-r--r--. 1 root root 370 Oct 19 15:08 b
-rw-r--r--. 1 root root 0 Sep 3 08:47 te
[root@eff5cd1a0e30 workdir]#
```

输入：echo \$key1 查看环境变量的值结果

```
[root@eff5cd1a0e30 workdir]# echo $key1
hello
[root@eff5cd1a0e30 workdir]#
```

## 7.2 springboot微服务部署

在微服的世界中，使用springboot来开发的微服务架构，使用dockerfile 来部署应用。

- 在本地开发完成微服系统 打包，将其copy到linux系统中

```
[root@localhost demo]# pwd
/root/apps/demo
[root@localhost demo]# ll
总用量 14276
-rw-r--r--. 1 root root 14612937 10月 19 11:42 demo-0.0.1-SNAPSHOT.jar
-rw-r--r--. 1 root root 135 10月 19 23:43 Dockerfile
[root@localhost demo]#
```

- 创建dockefile文件上图：

```
#my dockerfile ljh
FROM java:8
MAINTAINER ljh
ADD demo-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
EXPOSE 8080
```

ENTRYPOINT ["java","-jar","/app.jar"] 表示 执行命令： 整个命令都只能有一个ENTRYPOINT

```
java -jar app.jar
```

EXPOSE 8080 发布端口为：8080

- 构建镜像

```
docker build -t demoappimage .
```

- 创建容器

```
docker run -di --name=myapp1 -p 8080:8080
```

- 浏览器中访问系统：

`http://192.168.25.132:8080/hello`

- 效果

