# Docker简介

Docker 是一个开源项目，诞生于 2013 年初，最初是 dotCloud 公司内部的一个业余项目。它基于 Google 公司推出的 Go 语言实现。 项目后来加入了 Linux 基金会，遵从了 Apache 2.0 协议，项目代码在 [GitHub](https://github.com/docker/docker) 上进行维护。

Docker 项目的目标是实现轻量级的操作系统虚拟化解决方案。 Docker 的基础是 Linux 容器（LXC）等技术。

（背景），云计算兴起后，服务器硬件扩展非常便利，软件服务部署成为了瓶颈，docker趁势而兴。

# 为什么用 Docker

容器的启动可以在秒级实现，比传统的虚拟机方式要快得多

对系统资源的利用率很高，一台主机上可以同时运行数千个 Docker 容器

docker的出现，让开发/测试/线上的环境部署，成为便利一条龙。

## 更快速的交付和部署

对开发和运维（devop）人员来说，最希望的就是一次创建或配置，可以在任意地方正常运行。

开发者可以使用一个标准的镜像来构建一套开发容器，开发完成之后，运维人员可以直接使用这个容器来部署代码。 Docker 可以快速创建容器，快速迭代应用程序，并让整个过程全程可见，使团队中的其他成员更容易理解应用程序是如何创建和工作的。 Docker 容器很轻很快！容器的启动时间是秒级的，大量地节约开发、测试、部署的时间。

## 更高效的虚拟化

Docker 容器的运行不需要额外的 hypervisor 支持，它是内核级的虚拟化，因此可以实现更高的性能和效率。

## 更轻松的迁移和扩展

Docker 容器几乎可以在任意的平台上运行，包括物理机、虚拟机、公有云、私有云、个人电脑、服务器等。 这种兼容性可以让用户把一个应用程序从一个平台直接迁移到另外一个。

## 更简单的管理

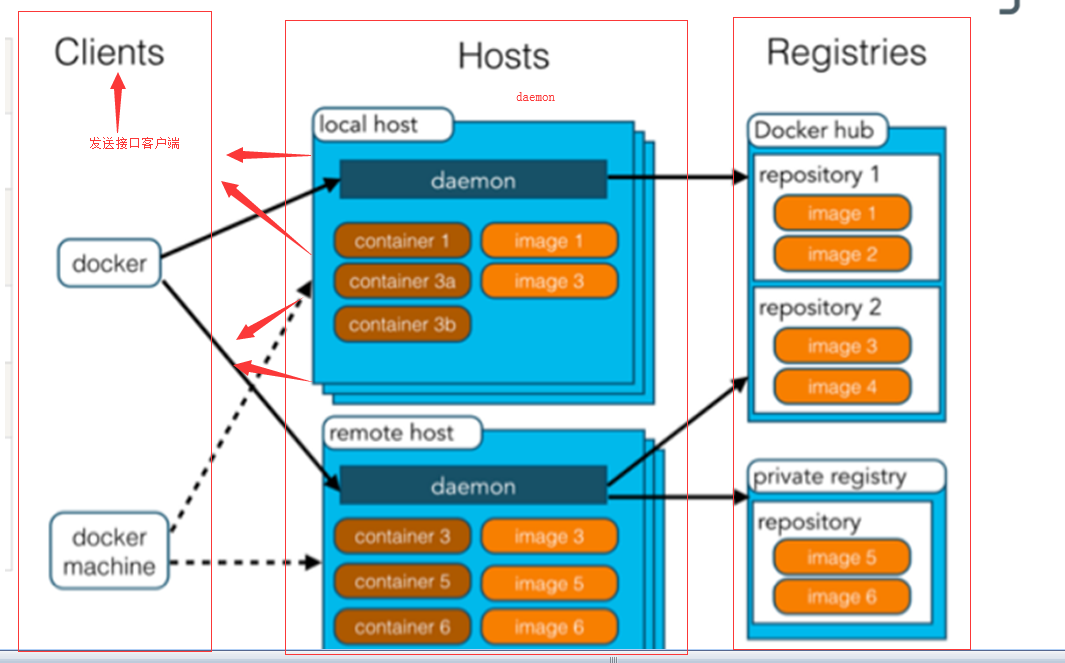
使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量的方式被分发和更新，从而实现自动化并且高效的管理。

## 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 | 弱于 |
| 系统支持量 | 单机支持上千个容器 | 一般几十个 |

# Docker基本概念

## Docker架构



host --- 主机载体 == docker安装的地方

继承类比：

Class2 extents Class1 ---------------------- Object o = new Class2

--------------------------------------此时，o对象的结构中，有Class1的成员结构

image2 extents image1 ----------------------Container c = new image2

-------------------------------------此时，c容器中，有image1的文件

## Docker 镜像

Docker 镜像就是一个只读的模板。

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 Apache 或用户需要的其它应用程序。

镜像可以用来创建 Docker 容器。

Docker 提供了一个很简单的机制来创建镜像或者更新现有的镜像，用户甚至可以直接从其他人那里下载一个已经做好的镜像来直接使用。

## Docker 容器

Docker 利用容器来运行应用。

容器是从镜像创建的运行实例。它可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台。

可以把容器看做是一个简易版的 Linux 环境（包括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。

## Docker 仓库

仓库是集中存放镜像文件的场所。有时候会把仓库和仓库注册服务器（Registry）混为一谈，并不严格区分。实际上，仓库注册服务器上往往存放着多个仓库，每个仓库中又包含了多个镜像，每个镜像有不同的标签（tag）。

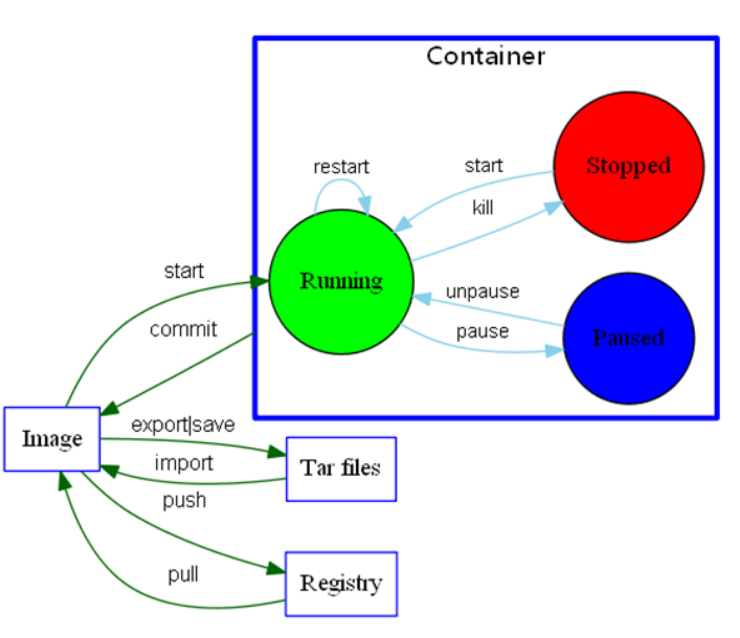
仓库分为公开仓库（Public）和私有仓库（Private）两种形式。

最大的公开仓库是 [Docker Hub](https://hub.docker.com/)，存放了数量庞大的镜像供用户下载。

当然，用户也可以在本地网络内创建一个私有仓库。

当用户创建了自己的镜像之后就可以使用 push 命令将它上传到公有或者私有仓库，这样下次在另外一台机器上使用这个镜像时候，只需要从仓库上 pull 下来就可以了。

## 容器、镜像的运行关系



# 安装 Docker

Docker 支持 CentOS6 及以后的版本。

## 卸载

1.查询安装过的包

yum list installed | grep docker

docker-engine.x86\_64                 17.03.0.ce-1.el7.centos         @dockerrepo

 2.删除安装的软件包

yum -y remove docker-engine.x86\_64

3.删除镜像/容器等

rm -rf /var/lib/docker

## CentOS6

对于 CentOS6，可以使用 [EPEL](https://fedoraproject.org/wiki/EPEL) 库安装 Docker，命令如下

$ sudo yum install http://mirrors.yun-idc.com/epel/6/i386/epel-release-6-8.noarch.rpm

$ sudo yum install docker-io

## CentOS7

CentOS7 系统 CentOS-Extras 库中已带 Docker，可以直接安装：

$ sudo yum install docker ##不是最新版本



#最新版安装

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

sudo yum-config-manager --add-repo https://[mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo](http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo)

sudo yum install docker-ce

## 查看docker版本

docker version

docker info

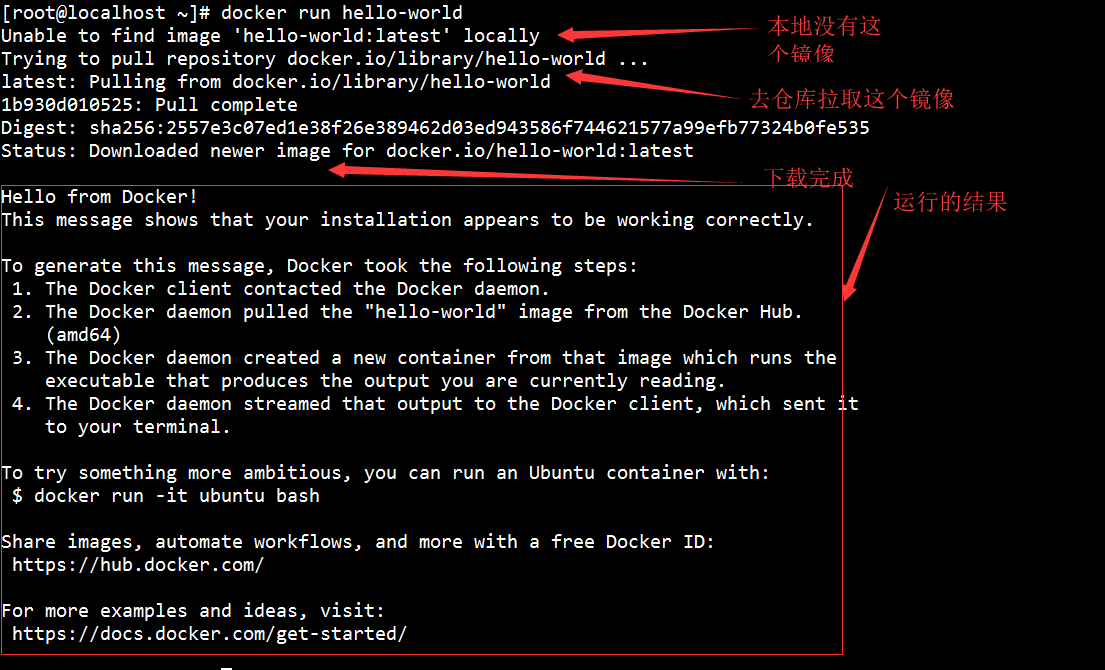
## 启动docker

sudo service docker start

## 设置随系统启动

sudo chkconfig docker on

## Docker初体验

docker run hello-world ##进入docker世界 

# Docker基本操作

## 容器操作

### docker [run|start|stop|restart|kill|rm|pause|unpause]

* [run](http://www.runoob.com/docker/docker-run-command.html)/[create](http://www.runoob.com/docker/docker-create-command.html)[镜像名]： 创建一个新的容器并运行一个命令
* [start/stop/restart](http://www.runoob.com/docker/docker-start-stop-restart-command.html)[容器名]：启动/停止/重启一个容器
* [kill](http://www.runoob.com/docker/docker-kill-command.html) [容器名]： 直接杀掉容器，不给进程响应时间
* [rm](http://www.runoob.com/docker/docker-rm-command.html)[容器名]：删除已经停止的容器
* [pause/unpause](http://www.runoob.com/docker/docker-pause-unpause-command.html)[容器名]：暂停/恢复容器中的进程

### docker [ps|inspect|exec|logs|export|import]

* [ps](http://www.runoob.com/docker/docker-ps-command.html)：查看容器列表（默认查看正在运行的容器，-a查看所有容器）
* [inspect](http://www.runoob.com/docker/docker-inspect-command.html)[容器名]：查看容器配置元数据
* exec -it [容器名] /bin/bash：进入容器环境中交互操作
* logs --since="2019-02-01" -f --tail=10 [容器名]:查看容器日志
* **cp** path1 **[容器名]:path 容器与主机之间的数据拷贝**
* **export -o test.tar** [容器名] **/ docker export** [容器名]**>test.tar : 文件系统作为一个tar归档文件**
* **import test.tar [镜像名:版本号]:导入归档文件，成为一个镜像**

## 镜像操作

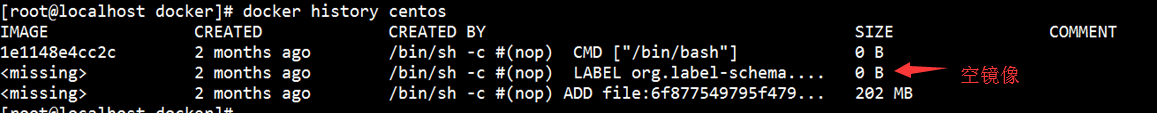
### docker images|rmi|tag|build|history|save|load]

* [images](http://www.runoob.com/docker/docker-images-command.html)：列出本地镜像列表
* [rmi](http://www.runoob.com/docker/docker-rmi-command.html) [镜像名：版本]：删除镜像
* tag [镜像名：版本] [仓库]/[镜像名：版本]：标记本地镜像，将其归入某一仓库
* [build](http://www.runoob.com/docker/docker-build-command.html) -t [镜像名：版本] [path]：Dockerfile 创建镜像
* [history](http://www.runoob.com/docker/docker-history-command.html) [镜像名：版本]: 查看指定镜像的创建历史
* **save -o xxx.tar** [镜像名：版本] **/  save** [镜像名：版本]**>xxx.tar** : **将镜像保存成 tar 归档文件**
* **load --input  xx.tar / docker load<xxx.tar : 从归档文件加载镜像**

# 镜像与容器原理及用法探究

## history命令查看镜像层

例：docker history hello-world

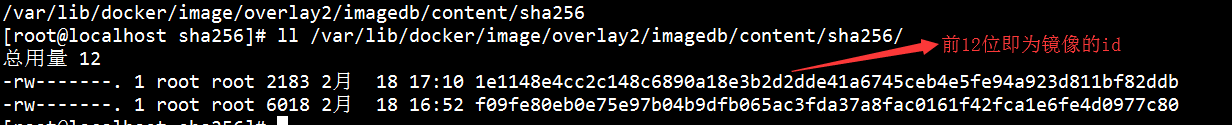


显示镜像hello-world分三层，其中两个空层

## 查看镜像文件

镜像存放在imagedb里

一般在image/overlay2/imagedb/content/sha256下



打开一个镜像文件查看其内容：

cat f09fe80eb0e75e97b04b9dfb065ac3fda37a8fac0161f42fca1e6fe4d0977c80

{

"architecture": "amd64",

"config": {

"Hostname": "",

"Domainname": "",

"User": "",

"AttachStdin": false,

"AttachStdout": false,

"AttachStderr": false,

"Tty": false,

"OpenStdin": false,

"StdinOnce": false,

"Env": ["PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"],

"Cmd": ["/hello"],

"ArgsEscaped": true,

"Image": "sha256:a6d1aaad8ca65655449a26146699fe9d61240071f6992975be7e720f1cd42440",

"Volumes": null,

"WorkingDir": "",

"Entrypoint": null,

"OnBuild": null,

"Labels": null

},

"container": "8e2caa5a514bb6d8b4f2a2553e9067498d261a0fd83a96aeaaf303943dff6ff9",

"container\_config": {

"Hostname": "8e2caa5a514b",

"Domainname": "",

"User": "",

"AttachStdin": false,

"AttachStdout": false,

"AttachStderr": false,

"Tty": false,

"OpenStdin": false,

"StdinOnce": false,

"Env": ["PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"],

"Cmd": ["/bin/sh", "-c", "#(nop) ", "CMD [\"/hello\"]"],

"ArgsEscaped": true,

"Image": "sha256:a6d1aaad8ca65655449a26146699fe9d61240071f6992975be7e720f1cd42440",

"Volumes": null,

"WorkingDir": "",

"Entrypoint": null,

"OnBuild": null,

"Labels": {}

},

"created": "2019-01-01T01:29:27.650294696Z",

"docker\_version": "18.06.1-ce",

"history": [{

"created": "2019-01-01T01:29:27.416803627Z",

"created\_by": "/bin/sh -c #(nop) COPY file:f77490f70ce51da25bd21bfc30cb5e1a24b2b65eb37d4af0c327ddc24f0986a6 in / "

}, {

"created": "2019-01-01T01:29:27.650294696Z",

"created\_by": "/bin/sh -c #(nop) CMD [\"/hello\"]",

"empty\_layer": true

}],

"os": "linux",

"rootfs": {

"type": "layers",

"diff\_ids": ["sha256:af0b15c8625bb1938f1d7b17081031f649fd14e6b233688eea3c5483994a66a3"]

}

}

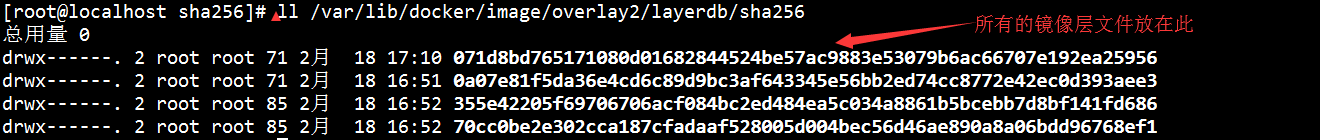
----其中，history数组内，标识了镜像的历史记录（与history命令内容对应）

----rootfs的diff\_ids中，对应了依赖使用中镜像层文件（history命令中size大于0的层）

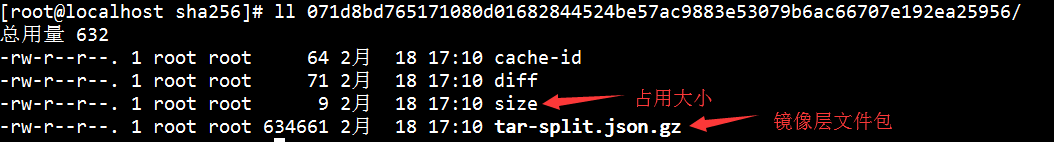
## 查看镜像层文件

层文件在layerdb里

ll /var/lib/docker/image/overlay2/layerdb/sha256

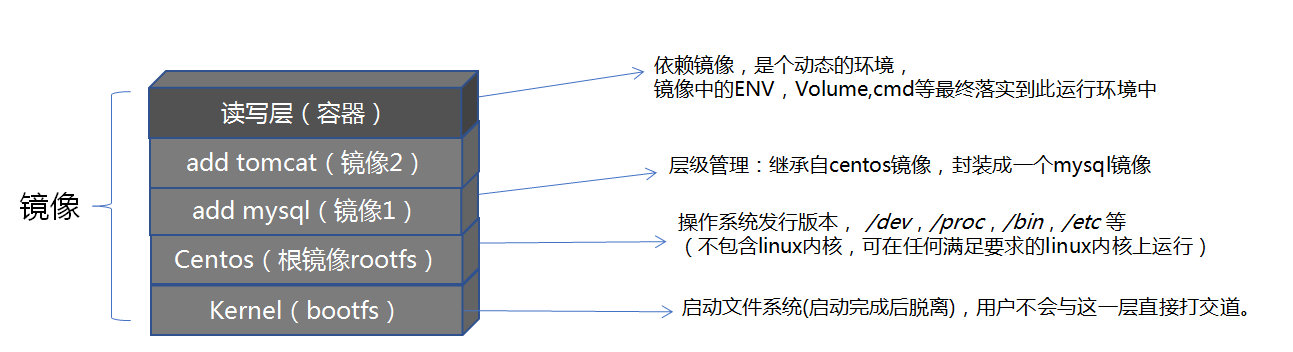


#镜像层文件内结构：



## 镜像与容器总结

一个镜像就是一层层的layer层文件，盖楼而成，上层文件叠于下层文件上，若上层文件有与下层文件重复的，则覆盖掉下层文件重复的部分，如下图：



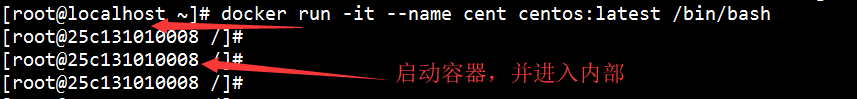
---------初始挂载时读写层为空。

---------当需要修改镜像内的某个文件时，只对处于最上方的读写层进行了变动，不复写下层已有文件系统的内容，已有文件在只读层中的原始版本仍然存在，但会被读写层中的新版本文件所隐藏，当 docker commit 这个修改过的容器文件系统为一个新的镜像时，保存的内容仅为最上层读写文件系统中被更新过的文件。

---------联合挂载是用于将多个镜像层的文件系统挂载到一个挂载点来实现一个统一文件系统视图的途径，是下层存储驱动(aufs、overlay等) 实现分层合并的方式。

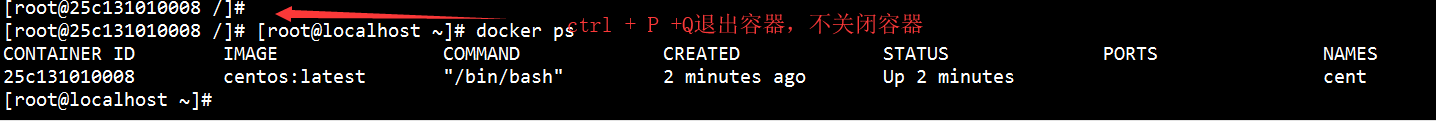
## 容器创建详解

### 交互式创建容器并进入：



docker run -it --name centos centos /bin/bash（前台进程）

------------------------exit退出也关闭容器; Ctrl+P+Q退出不关闭容器



### 后台启动容器：



docker run -d --name nginx nginx

### 进入已运行的容器：

docker exec -it nginx /bin/bash

查看容器的元数据： docker inspect nginx

### 绑定容器端口到主机：

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

### 挂载主机文件目录到容器内：

docker run -dit -v /root/peter\_dir/:/pdir --name cent centos

### 复制主机文件到容器内：

docker cp anaconda-ks.cfg cent:/var