学习资料

<https://blog.csdn.net/zzq900503/article/details/80230415>

<http://c.biancheng.net/view/3118.html>

<https://www.cnblogs.com/sparkdev/tag/docker/>

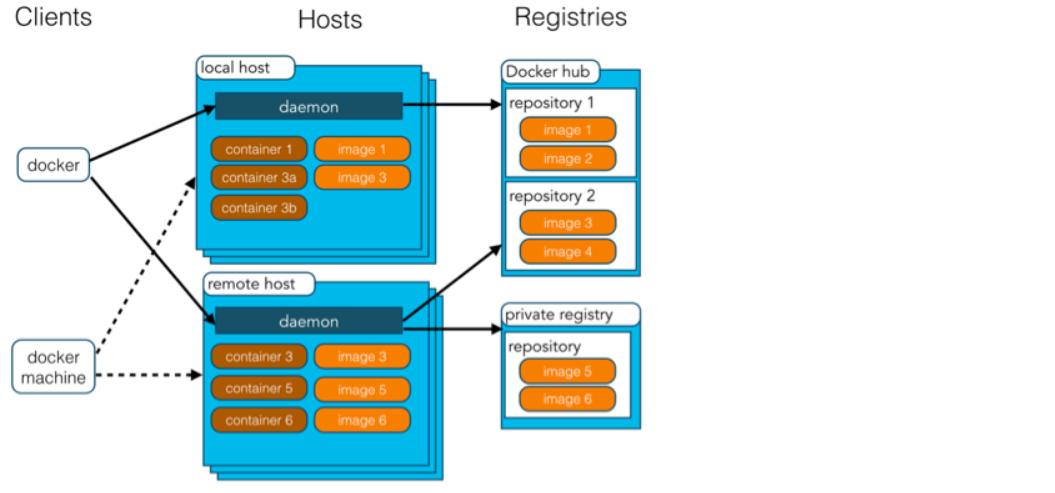
<https://blog.csdn.net/a745233700/article/details/80431045>

## 一 Docker介绍

容器、沙箱

docker如VM，image就是你装vm虚拟机时的iso文件，容器就是正在vm中运行的虚拟机

### 1 Docker架构



|  |
| --- |
| Docker 守护进程(daemon)  Docker Daemon是Docker架构中运行在后台的守护进程，Docker daemon运行在宿主机上，用户不能直接操作daemon，只能通过docker client转达。  Docker 镜像(Images)  Docker 镜像是用于创建 Docker 容器的模板。  Docker 容器(Container)  容器是独立运行的一个或一组应用。  Docker 客户端(Client)  Docker 客户端通过命令行或者其他工具使用 Docker API (https://docs.docker.com/reference/api/docker\_remote\_api) 与 Docker 的守护进程通信。  Docker 主机(Host)  一个物理或者虚拟的机器用于执行 Docker 守护进程和容器。  Docker 仓库(Registry)  Docker 仓库用来保存镜像，可以理解为代码控制中的代码仓库。  Docker Hub(https://hub.docker.com) 提供了庞大的镜像集合供使用。  Docker Machine  Docker Machine是一个简化Docker安装的命令行工具，通过一个简单的命令行即可在相应的平台上安装Docker，比如VirtualBox、 Digital Ocean、Microsoft Azure。 |

### **2 Docker**镜像

镜像，三大核心组件之一，只读的文件系统。运行容器时，需要本地有对应容器的镜像文件。如果没有，则从远程仓库中获取该镜像。

镜像是一个只读的层（layer），由文件系统叠加构成。而镜像又是容器(Container)的构成单元，我们一般会将应用构建成标准的镜像组件，一或多个镜像叠加又构成了容器。

在Docker中将这种文件称为镜像（Image），或者说上Docker容器的文件系统是由多个镜像来构成的。一个镜像，可以放在另一个镜像之上。位于底层的镜像称之为父镜像，而位于最底层的镜像称之为基础镜像（Base Image）。

当从一个镜像启动容器时，Docker会从该镜像的最顶层加载一个读／写文件系统，而我们的应用就会运行于这个读/写层中。

### 3 Docker容器

类似linux系统环境，运行和隔离应用。容器从镜像启动的时候，docker会在镜像的最上一层创建一个可写层，镜像本身是只读的，保持不变。

容器是镜像的一个运行实例，不同的是它带有额外的可写层。

可认为docker容器就是独立运行的一个或一组应用，以及它们所运行的必需环境。

### 4 Docker仓库

仓库是集中存放镜像的地方。每个服务器上可以有多个仓库。

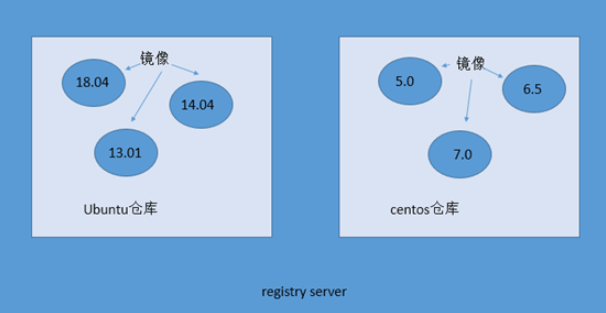
仓库又分为公有仓库（DockerHub、dockerpool）和私有仓库

公有库：

DockerHub：docker官方维护的一个公共仓库https://hub.docker.com，其中包括了15000多个的镜像，大部分都可以通过dockerhub直接下载镜像。也可通过docker search和docker pull命令来下载。

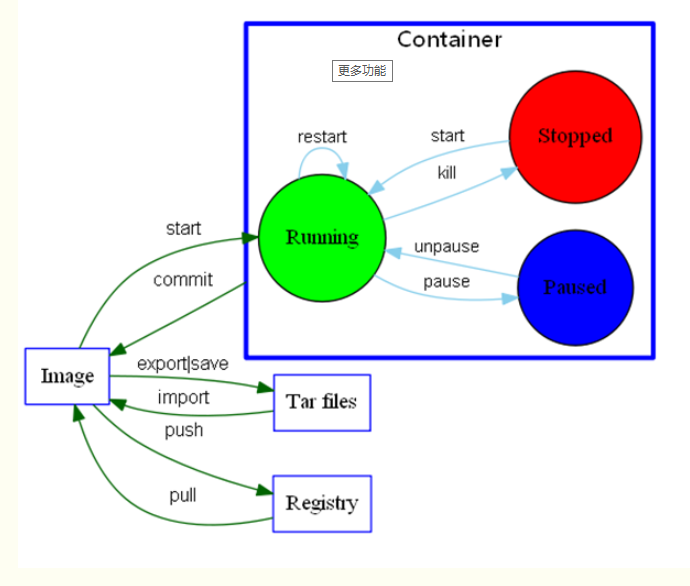
DockerPool：国内专业的docker技术社区，http://www.dockerpool.com也提供官方镜像的下载。

每个仓库中存放着一类镜像，镜像、仓库和registry的关系如下。



Docker三大核心组件的关系

容器、镜像、仓库的运行关系图



## 二 Docker安装

### 1 Ubuntu 16.04 安装 Docker

1.选择国内的云服务商，这里选择阿里云为例

curl -sSL http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/docker-engine/internet | sh -

2.安装所需要的包

sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual

3.添加使用 HTTPS 传输的软件包以及 CA 证书

sudo apt-get update

sudo apt-get install apt-transport-https ca-certificates

4.添加GPG密钥

sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D

5.添加软件源

echo "deb https://apt.dockerproject.org/repo ubuntu-xenial main" | sudo tee /etc/apt/sources.list.d/docker.list

6.添加成功后更新软件包缓存

sudo apt-get update

7.安装docker

sudo apt-get install docker-engine

8.设置开机自启动 docker，并启动docker-cer

sudo systemctl enable docker

sudo systemctl start docker

### 2 Ubuntu 18.04 安装 Docker-ce

1.更换国内软件源，推荐中国科技大学的源，稳定速度快（可选）

sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak

sudo sed -i 's/archive.ubuntu.com/mirrors.ustc.edu.cn/g' /etc/apt/sources.list

sudo apt update

2.安装需要的包

sudo apt install apt-transport-https ca-certificates software-properties-common curl

3.添加 GPG 密钥，并添加 Docker-ce 软件源，这里还是以中国科技大学的 Docker-ce 源为例

curl -fsSL https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu \

$(lsb\_release -cs) stable"

4.添加成功后更新软件包缓存

sudo apt update

5.安装 Docker-ce

sudo apt install docker-ce

6.设置开机自启动并启动 Docker-ce（安装成功后默认已设置并启动，可忽略）

sudo systemctl enable docker

sudo systemctl start docker

7.测试运行

sudo docker run hello-world

8.添加当前用户到 docker 用户组，可以不用 sudo 运行 docker（可选）

sudo groupadd docker

sudo usermod -aG docker $USER

9.测试添加用户组（可选）

docker run hello-world

## 三 Docker镜像加速

Ubuntu16.04+、Debian8+、CentOS7

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）（最简单的方案）：

{"registry-mirrors":["https://registry.docker-cn.com"]}

之后重新启动服务：

$ sudo systemctl daemon-reload

$ sudo systemctl restart docker

|  |
| --- |
| 国内较快的镜像原地址  #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 |

## 四 Docker镜像使用

镜像学习地址：<https://www.cnblogs.com/sparkdev/p/8901728.html#t1>

### 1 获取docker镜像

1 通过docker image pull 命令获取镜像

本地镜像是运行容器的前提，所以在运行容器前我们需要使用 docker image pull 命令从网络上的镜像仓库把镜像拉取到本地。该命令的格式为：

docker image pull [OPTIONS] NAME[:TAG|@DIGEST]

只指定了镜像的名称，默认会选择拉取 latest 标签标记的镜像。比如我们要拉取最新的 ubuntu 镜像：

$ docker image pull Ubuntu

该命令实际上是从在配置文件中配置的仓库地址中拉取最近的ubuntu镜像。（默认是：DockerHub0，下载较慢，可以配置国内的下载地址。）

镜像下载到本地后就可运行容器了，比如：

$ docker run --rm ubuntu echo hello docker

2 通过创建Dockerfile文件获取镜像。

待续。。。。。

### 2 查看镜像信息

docke image ls

docker images

|  |
| --- |
| 输出的信息中包含的内容有：  REPOSITORY：说明镜像来自哪个仓库，比如 ubuntu 或 registry.docker-cn.com/library/ubuntu。  TAG：镜像的标签信息，比如 14.04 或 latest。  IMAGE ID：标识镜像的 ID 号。  CREATED：创建镜像的时间。  SIZE：镜像大小。 |

为本地镜像添加tag

docker image tag

$ docker image tag ubuntu:14.04 oldubuntu

添加本地镜像添加新的tag

### 3搜索镜像

除了直接在 Docker Hub 的官方网站上搜索镜像资源，还可以通 docker search 命令以命令行的方式进行搜索，比如搜索 mysql 镜像：

$ docker search mysql

选择通过docker search 命令搜索的结果时需要谨慎，一般在docker Hub中查询详细的信息。

### 4删除镜像

Docker image rm +(tag/ID)

两者略有不同

使用tag删除镜像

在通过命令docker image rm Ubuntu:10.0 删除镜像时，如果该镜像的ID对应一个镜像tag，则删除最终的镜像文件。如果该镜像ID对应多个tag，则通过tag删除时，并不同删除镜像文件。因为其他tag还在引用这个镜像文件。当删除最后一个ID对应的tag镜像时，该镜像文件才被彻底删除。

使用ID删除镜像

Docker image rm imgID

通过ID删除镜像时，在镜像ID 只对应一个tag时，可以删除镜像。

如果该ID被多个tag引用，则提示删除错误。

### 5 创建镜像

待续。。。。。

### 6 导出和导入镜像

待续。。。。。

### 7 上传镜像

待续。。。。。

## 五 Docker容器使用

Docker容器学习地址：<https://www.cnblogs.com/whych/p/9446032.html>

### 1 创建容器

1 docker run -i -t REPOSITORY:TAG (新建并启动容器:)

-t:让docker分配一个伪终端并绑定到容器的标准输入上

-i: 让容器的标准输入保持打开

若要在后台以守护态（daemonized）形式运行，可加参数-d

Eg:

Docker run –ti Ubuntu:10.04

Docker run 的执行过程

|  |
| --- |
| 在执行docker run来创建并启动容器时，后台运行的标准包括：  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可写层  从宿主机配置的网桥接口中桥接一个虚拟接口到容器  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 |

2 docker create –it Ubuntu

使用docker create命令创建的容器处于停止状态，可以使用docker start命令启动它。

### 2开启/停止/重启容器

docker start/stop/restart <containerID>

### 3进入容器

docker attach [container\_id]

连接一个正在运行的container实例（即实例须为start状态，可以多个 窗口同时attach 一个container实例），但当某个窗口因命令阻塞时，其它窗口也无法执行了。

在使用-d参数时，容器启动后会进入后台，用户无法看到容器中的信息。

exec可直接在容器内运行的命令

docker exec container\_ID

### 4退出容器

Exit

Ctrl+P+Q

### 5 查看容器

docker ps –a

查看所有容器

docker ps

查看正在运行的容器。

### 6 删除容器

docker rm <container...> #：删除一个或多个container(删除停止的容器。)

docker rm `docker ps -a -q` #：删除所有的container

docker ps -a -q | xargs docker rm #：同上, 删除所有的container

docker -rm

-f 强制中止并运行的容器

-l 删除容器的连接，但保留容器

-v 删除容器挂载的数据卷

### 7修改容器

docker commit <container> [repo:tag]

将一个container固化为一个新的image，后面的repo:tag可选。

### 8 导出容器

待续。。。。

### 9 查看版本信息

docker –v #简单查看版本  
docker version #查看版本

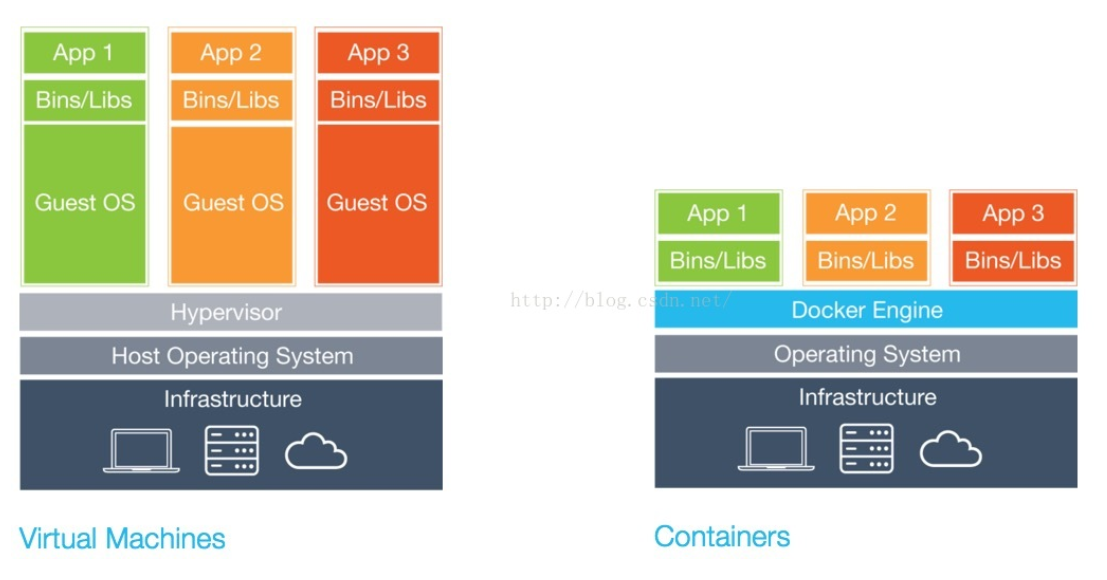
docker info #查看信息

## 六 Docker仓库使用

## 七 Docker卸载

$ sudo apt-get remove docker docker-engine docker.io containerd runc

## 八 Docker与虚拟机的关系和区别

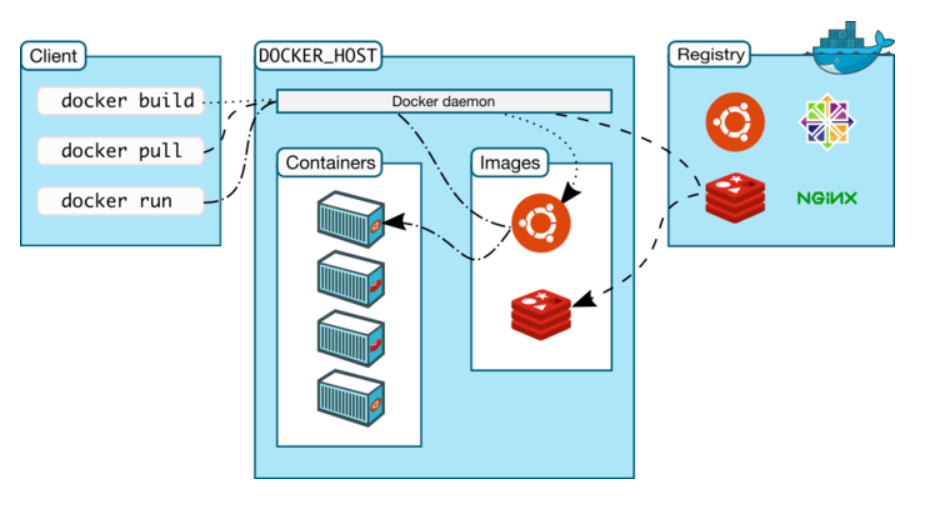


每个虚拟机不仅包括程序、程序需要的包，而且还包含了一个操作系统，但是Docker容器是共享一个操作系统内核（kernel）的，这些容器通过命名空间相互独立。

虚拟机实现资源隔离和环境隔离的方法是利用独立的OS，并利用Hypervisor（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。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。）虚拟化CPU、内存、IO设备等实现的。

对比虚拟机实现资源和环境隔离的方案，Docker就显得简练很多。Docker Engine可以简单看成对Linux的NameSpace、Cgroup、镜像管理文件系统操作的封装。Docker并没有和虚拟机一样利用一个完全独立的Guest OS实现环境隔离，它利用的是目前Linux内核本身支持的容器方式实现资源和环境隔离。简单的说，Docker利用namespace实现系统环境的隔离；利用Cgroup实现资源限制；利用镜像实现根目录环境的隔离。

## 九 Docker运行流程



正如上图所示，当我们想运行一个容器的时候，docker会：

1、拉取镜像，若本地已经存在该镜像，则不用到网上去拉取

2、创建新的容器

3、分配union文件系统并且挂着一个可读写的层，任何修改容器的操作都会被记录在这个读写层上，你可以保存这些修改成新的镜像，也可以选择不保存，那么下次运行改镜像的时候所有修改操作都会被消除

4、分配网络\桥接接口，创建一个允许容器与本地主机通信的网络接口

5、设置ip地址，从池中寻找一个可用的ip地址附加到容器上，换句话说，localhost并不能访问到容器

6、运行你指定的程序

7、捕获并且提供应用输出，包括输入、输出、报错信息

## 十 Docker中安装常用软件

<https://www.jianshu.com/p/17c5811cd850>

## =========================

## 问题整理

### 1 Ubuntu下docker使用非root权限运行docker

默认情况下，docker 命令会使用 Unix socket 与 Docker 引擎通讯。而只有 root 用户和 docker 组的用户才可以访问 Docker 引擎的 Unix socket。出于安全考虑，一般 Linux 系统上不会直接使用 root 用户。因此，更好地做法是将需要使用 docker 的用户加入 docker 用户组。

1.添加 docker group ：

$ sudo groupadd docker

2.将用户加入该 group 内

$ sudo usermod -aG docker $USER

# 或者使用下面命令

$ sudo gpasswd -a ${USER} docker

3.重启服务

$ sudo service docker restart

# 或者

$ sudo /etc/init.d/docker restart

4.切换当前会话到新 group 或者重启 X 会话

注意:这一步是必须的，否则因为 groups 命令获取到的是缓存的组信息，刚添加的组信息未能生效，所以 docker images 执行时同样有错。

$ newgrp - docker