一、Docker简介

Docker是什么?

Docker的英文本意是“搬运工”，在程序员的世界里，Docker搬运的是集装箱（Container），集装箱里装的是任意类型的App，开发者通过Docker可以将App变成一种标准化的、可移植的、自管理的组件，可以在任何主流系统中开发、调试和运行。

说白了,**docker是一种用了新颖方式实现的轻量级虚拟机,类似于VM,但是在原理和应用上和VM的差别还是很大的.并且docker的专业叫法是应用容器(Application Container)。**

为啥要用容器?

**应用容器是个啥样子呢,一个做好的应用容器长得就像一个装好了一组特定应用的虚拟机一样,比如我现在想用mysql,那我就找个装好了mysql的容器就可以了,然后运行起来,我就能使用mysql了。**

**为啥不能直接安装一个mysql?安装一个SQL Server也可以啊,可是有的时候根据每个人电脑的不同,在安装的时候可能会报出各种各样的错误,万一你的机器中毒了,你的电脑挂了,你所有的服务都需要重新安装.但是有了docker,或者说有了容器就不同了,你就相当于有了一个可以运行起来的虚拟机,只要你能运行容器,mysql的配置就省了.而且如果你想换个电脑,直接把容器”端过来”就可以使用容器里面的服务.**

Docker 基于 Go 语言开发，代码托管在Github上，并遵循Apache 2.0 开源协议。Docker 容器可以封装任何有效负载，几乎可以在任何服务器之间进行一致性运行。换句话说，开发者构建的应用只需一次构建即可多平台运行。运营人员只需配置他们的服务，即可运行所有的应用。

**若是利用容器的话,那么开发直接在容器里开发,测试的时候把整个容器给测试,测好了把测试后容器再上线就好了.通过容器,整个开发,测试和生产环境可以保持高度一致。**

**此外容器也VM一样具有一定得隔离性,各个容器之间的数据和内存空间相互隔离,可以保证一定的安全性。**

Hyper-V、KVM和Xen等虚拟机管理程序都“基于虚拟化硬件仿真机制。这意味着，它们对系统要求很高.然而，容器却使用共享的操作系统。这意味着它们在使用系统资源方面比虚拟机管理程序要高效得多。容器不是对硬件进行虚拟化处理，而是驻留在一个Linux实例上。

Docker可以解决虚拟机能够解决的问题，同时也能够解决虚拟机由于资源要求过高而无法解决的问题。

为什么要使用docker？

1 、快速交付应用程序

•开发者使用一个标准的 image 来构建开发容器，开发完成之后，系统管理员就可以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！容器的启动时间是次秒级的，节约开发、测试、部署的时间

2 、更容易部署和扩展

•docker容器可以在几乎所有的环境中运行，物理机、虚拟机、公有云、私有云、个人电脑、服务器等等。

•docker容器兼容很多平台，这样就可以把一个应用程序从一个平台迁移到另外一个。

3 、效率更高

•docker容器不需要 hypervisor ，他是内核级的虚拟化。

4 、快速部署也意味着更简单的管理

•通常只需要小小的改变就可以替代以往巨型和大量的更新工作。

Docker 的常用案例包括：

 自动打包和部署应用

 创建轻量、私有的 PaaS 环境

 自动化测试和持续集成/部署

 部署并扩展 Web 应用、数据库和后端服务器

那么为啥不用VM?

**那么既然容器和VM这么类似为啥不用VM?docker容器相对于VM还是有很多优点的:**

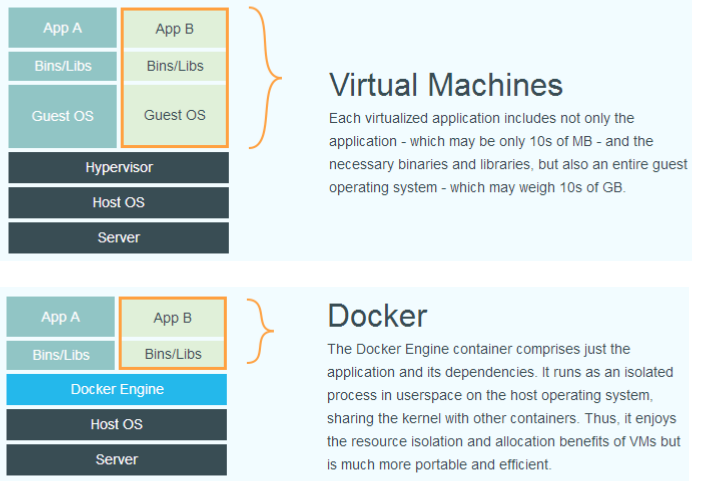
**1.启动速度快,容器通常在一秒内可以启动.而VM要很久.**

**2.资源利用率高,一台普通服务器可以跑上千个容器，而跑VM就。。。。。。**

**3.性能开销小,VM需要额外的CPU和内存来完成OS的功能,这一部分占据了额外的资源.**

为啥相似的功能在性能上会有如此巨大的差距呢?看一下他们的设计图,先看VM的:

下面的图片比较了 Docker 和传统虚拟化方式的不同之处，



可见容器是在操作系统层面上实现虚拟化，直接复用本地主机的操作系统，而传统方式则是在硬件层面实现。

Docker优势和劣势

作为一种新兴的虚拟化方式，Docker 跟传统的虚拟化方式相比具有众多的优势。

首先，Docker 容器的启动可以在秒级实现，这相比传统的虚拟机方式要快得多。

其次，Docker 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。

容器除了运行其中应用外，基本不消耗额外的系统资源，使得应用的性能很高，同时系统的开销尽量小。传统虚拟机方式运行 10 个不同的应用就要起 10 个虚拟机，而 Docker 只需要启动 10 个隔离的应用即可。

具体说来，Docker 在如下几个方面具有较大的优势。

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

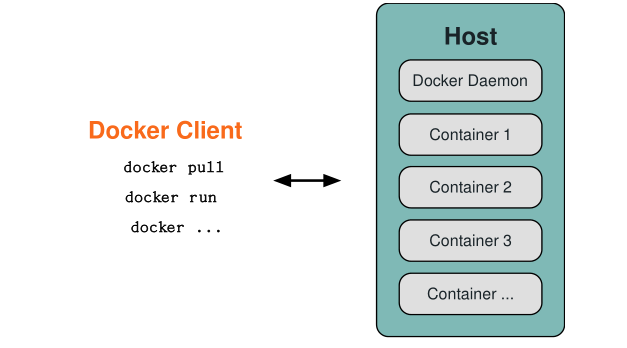
使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量的方式被分发和更新，从而实现自动化并且高效的管理。

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 |  |

二、Docker 的体系结构

docker使用C/S 架构，docker daemon 作为 server 端接受 client 的请求，并处理（创建、运行、分发容器），他们可以运行在一个机器上，也通过 socket或者 RESTful API 通信



Docker daemon 一般在宿主主机后台运行。

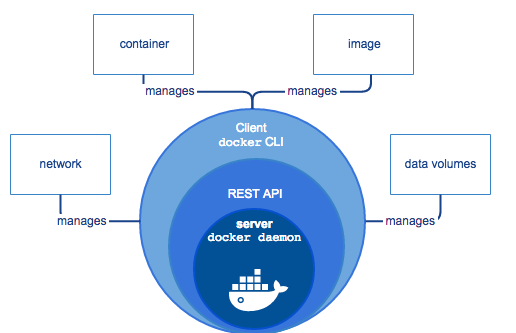
Docker client以系统命令的形式存在，用户用docker命令来跟docker daemon 交互。

Docker 守护进程（Docker daemon）

如上图所示，Docker 守护进程运行在一台主机上。用户并不直接和守护进程进行交互，而是通过 Docker 客户端间接和其通信。

Docker 客户端（Docker client）

Docker 客户端，实际上是docker的二进制程序，是用户与 Docker 交互方式。它接收用户指令并且与背后的 Docker 守护进程通信。



Docker 内部：

要理解 Docker 内部构建，需要理解以下三种部件：

Docker 镜像 - Docker images

Docker 仓库 - Docker registeries

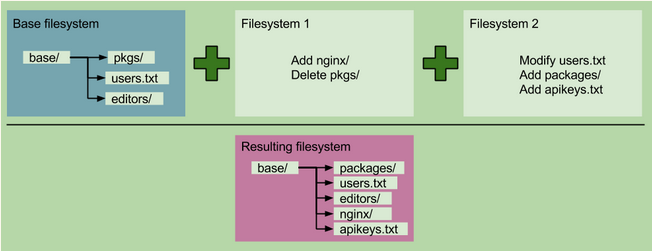
Docker 容器 - Docker containers

#### Docker 镜像

Docker 镜像是 Docker 容器运行时的只读模板，镜像可以用来创建 Docker 容器。每一个镜像由一系列的层 (layers) 组成。Docker 使用UnionFS（联合文件系统）来将这些层联合到单独的镜像中。UnionFS允许独立文件系统中的文件和文件夹(称之为分支)被透明覆盖，形成一个单独连贯的文件系统。正因为有了这些层的存在，Docker 是如此的轻量。当你改变了一个 Docker 镜像，比如升级到某个程序到新的版本，一个新的层会被创建。因此，不用替换整个原先的镜像或者重新建立(在使用虚拟机的时候你可能会这么做)，只是一个新的层被添加或升级了。现在你不用重新发布整个镜像，只需要升级，层使得分发 Docker 镜像变得简单和快速。

每个docker都有很多层次构成，docker使用 union file systems 将这些不同的层结合到一个image 中去。

例如：centos镜像中安装nginx，就成了nginx镜像”，其实在此时Docker镜像的层级概念就体现出来了。底层一个centos操作系统镜像，上面叠加一个ngnx层，就完成了一个nginx镜像的构建。层级概念就不难理解，此时我们一般centos操作系统镜像称为nginx镜像层的父镜像。



#### Docker 仓库

Docker 仓库用来保存镜像，可以理解为代码控制中的代码仓库。同样的，Docker 仓库也有公有和私有的概念。公有的 Docker 仓库名字是 Docker Hub。Docker Hub 提供了庞大的镜像集合供使用。这些镜像可以是自己创建，或者在别人的镜像基础上创建。

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，存放了数量庞大的镜像供用户下载。国内的公开仓库包括 Docker Pool等，可以提供大陆用户更稳定快速的访问。

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。

\*注：Docker 仓库的概念跟Git类似，注册服务器可以理解为 GitHub 这样的托管服务。

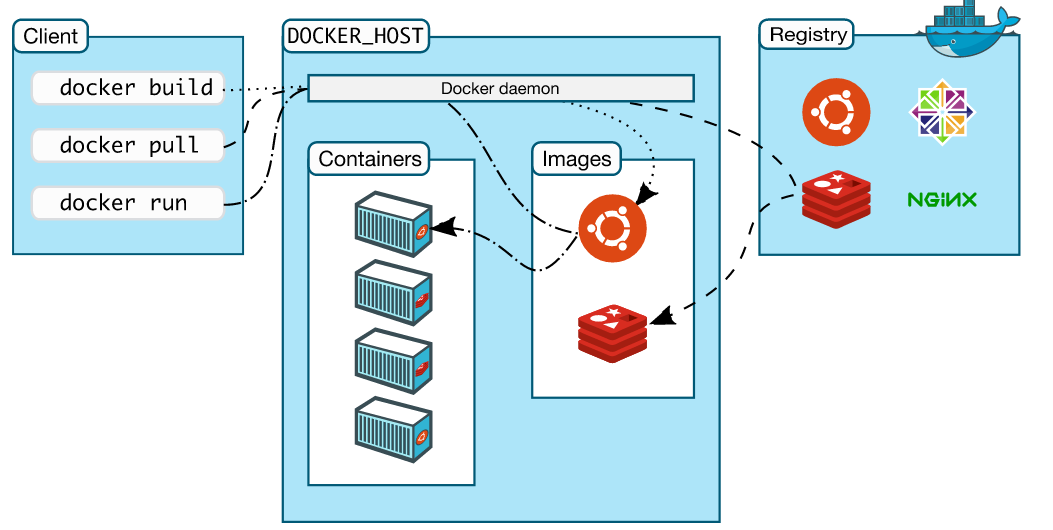
#### Docker 容器

Docker 利用容器来运行应用，一个Docker容器包含了所有的某个应用运行所需要的环境。每一个 Docker 容器都是从 Docker 镜像创建的。Docker 容器可以运行、开始、停止、移动和删除。每一个 Docker 容器都是独立和安全的应用平台。

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。

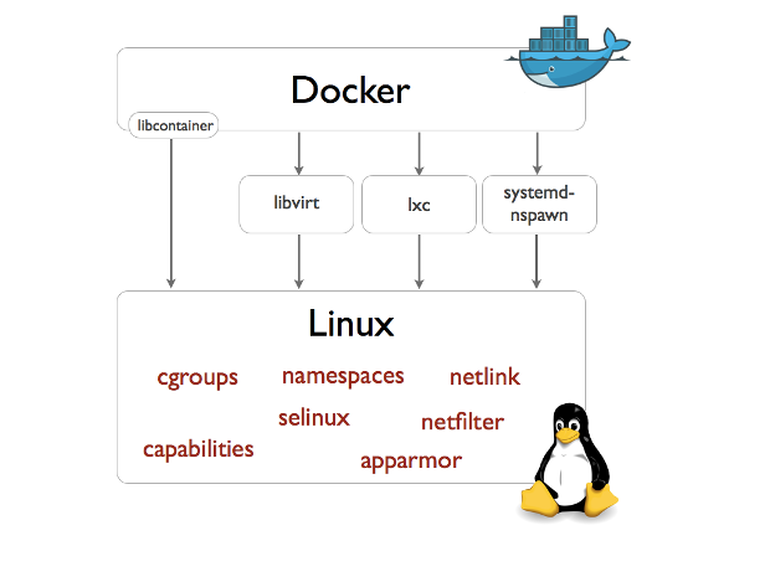
\*注：镜像是只读的，容器在启动的时候创建一层可写层作为最上层。



与虚拟机相比，容器有一个很大的差异，它们被设计用来运行"单进程"，无法很好地模拟一个完整的环境。Docker设计者极力推崇“一个容器一个进程的方式”，如果你要选择在一个容器中运行多个进程，那唯一情况是：出于调试目的。

容器是设计来运行一个应用的，而非一台机器。你可能会把容器当虚拟机用，但你将失去很多的灵活性，因为Docker提供了用于分离应用与数据的工具，使得你可以快捷地更新运行中的代码/系统，而不影响数据。

Docker 从 0.9 版本开始使用 libcontainer 替代 lxc，libcontainer 和 Linux 系统的交互图如下：



Docker 底层技术

docker底层的 2 个核心技术分别是 Namespaces 和 Control groups

Namespaces用来隔离各个容器

1)pid namespace

不同用户的进程就是通过pid namespace 隔离开的，且不同 namespace 中可以有相同pid。所有的LXC进程在docker中的父进程为docker进程，每个lxc进程具有不同的 namespace 。

2) net namespace

有了pid namespace, 每个 namespace 中的pid能够相互隔离，但是网络端口还是共享 host 的端口。网络隔离是通过 net namespace 实现的，每个 net namespace 有独立的 network devices, IP addresses, IP routing tables, /proc/net 目录。这样每个 container 的网络就能隔离开来。docker默认采用veth的方式将 container 中的虚拟网卡同 host 上的一个docker bridge: docker0 连接在一起。

3) ipc namespace

container 中进程交互还是采用linux常见的进程间交互方法 (interprocess communication - IPC),包括常见的信号量、消息队列和共享内存。container 的进程间交互实际上还是host 上具有相同pid namespace 中的进程间交互。

4) mnt namespace

类似chroot，将一个进程放到一个特定的目录执行。mnt namespace 允许不同 namespace 的进程看到的文件结构不同，这样每个 namespace 中的进程所看到的文件目录就被隔离开了。在container里头，看到的文件系统，就是一个完整的linux系统，有/etc、/lib 等，通过chroot实现。

5) uts namespace

UTS("UNIX Time-sharing System") namespace 允许每个 container 拥有独立的 hostname 和 domain name, 使其在网络上可以被视作一个独立的节点而非 Host 上的一个进程。

6) user namespace

每个 container 可以有不同的 user 和 group id, 也就是说可以在 container 内部用 container 内部的用户执行程序而非 Host 上的用户。

有了以上 6 种 namespace 从进程、网络、IPC、文件系统、UTS和用户角度的隔离，一个 container 就可以对外展现出一个独立计算机的能力，并且不同 container 从 OS 层面实现了隔离。然而不同 namespace 之间资源还是相互竞争的，仍然需要类似ulimit来管理每个 container 所能使用的资源 - -cgroup。

cgroups（Control groups）实现了对资源的配额和度量。

三、Docker 安装

docker官网：[https://docs.docker.com](https://docs.docker.com/)

**Docker值得关注的特性：**

* 文件系统隔离：每个进程容器运行在一个完全独立的根文件系统里。
* 资源隔离：系统资源，像CPU和内存等可以分配到不同的容器中，使用cgroup。
* 网络隔离：每个进程容器运行在自己的网络空间，虚拟接口和IP地址。
* 日志记录：Docker将会收集和记录每个进程容器的标准流（stdout/stderr/stdin），用于实时检索或批量检索。
* 变更管理：容器文件系统的变更可以提交到新的映像中，并可重复使用以创建更多的容器。无需使用模板或手动配置。
* 交互式shell：Docker可以分配一个虚拟终端并关联到任何容器的标准输入上，

CentOS 系列安装 Docker，Docker 支持 CentOS6 及以后的版本。

CentOS6：

在RedHat/CentOS环境下安装Docker。官方文档要求Linux kernel至少3.8以上，且docker只能运行在64位的系统中。由于RHEL6和CentOS6的内核版本为2.6，因此必须要先升级内核。

**升级内核（记住一定要升级，要不然会出现很多莫名奇怪的问题，建议用yum安装）**

    1、yum安装带aufs模块的3.10内核

|  |  |
| --- | --- |
|  | #cd /etc/yum.repos.d  #wget http://www.hop5.in/yum/el6/hop5.repo  #yum install kernel-ml-aufs kernel-ml-aufs-devel |

2、修改grub的主配置文件/etc/grub.conf，设置default=0，表示第一个title下的内容为默认启动的kernel（一般新安装的内核在第一个位置），重启系统，这时候你的内核就成功升级了。

|  |  |
| --- | --- |
|  | [root@localhost ~]# uname -r  3.10.5-3.el6.x86\_64 |

 查看内核是否支持aufs：

|  |  |
| --- | --- |
|  | [root@localhost ~]# grep aufs /proc/filesystems  nodev    aufs |

对于 CentOS6，可以使用EPEL库安装 Docker，命令如下

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

#yum install docker-io

启动docker服务

#service docker start

CentOS7：

Prerequisites（先决条件）

Docker requires a 64-bit installation regardless of your CentOS version. Also, your kernel must be 3.10 at minimum, which CentOS 7 runs.

To check your current kernel version, open a terminal and use uname -r to display your kernel version:



Docker 软件包已经包括在默认的 CentOS-Extras 软件源里。因此想要安装docker，只需要运行下面的 yum 命令：

[root@localhost~]#yum install docker

启动 Docker 服务：

安装完成后，使用下面的命令来启动docker服务，并将其设置为开机启动：

[root@localhost~]#service docker start

[root@localhost~]#chkconfig docker on

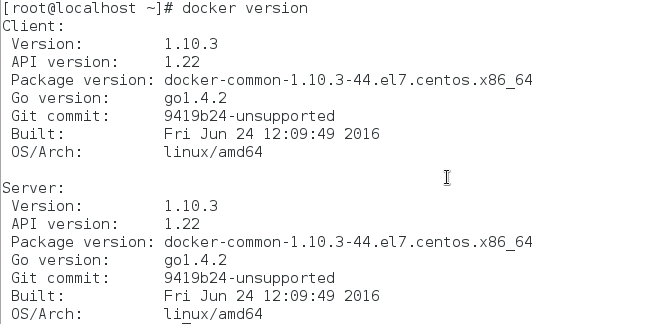
（此处采用了旧式的sysv语法，如采用CentOS 7中支持的新式systemd语法，如下：

[root@localhost~]#systemctl start docker.service

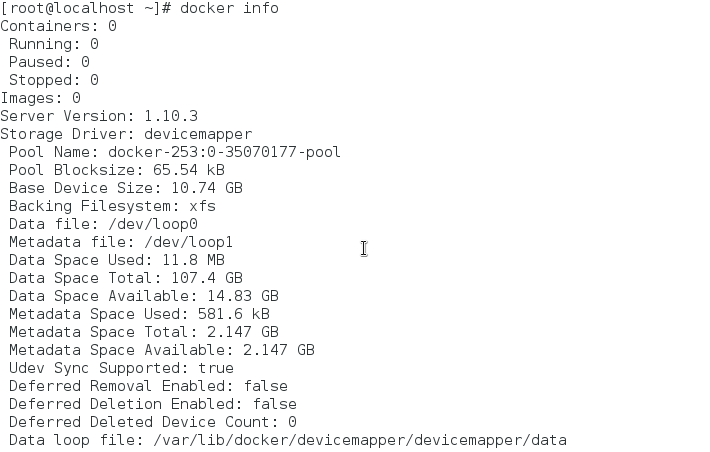
[root@localhost~]#systemctl enable docker.service

查看docker版本

#docker version



查看docker版本：docker info



以上是使用centos7软件源提供的docker安装程序

附：也可以按照官方文档安装

1. Log into your machine as a user with sudo or root privileges.
2. Make sure your existing yum packages are up-to-date.

3．Add the yum repo

$ sudo tee /etc/yum.repos.d/docker.repo<<-'EOF'

[dockerrepo]

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

enabled=1

gpgcheck=1

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

EOF

注：如果我们既想把输出保存到文件中，又想在屏幕上看到输出内容，就可以使用tee命令了。tee命令读取标准输入，把这些内容同时输出到标准输出和（多个）文件中

4．Install the Docker package

$ sudo yum install docker-engine

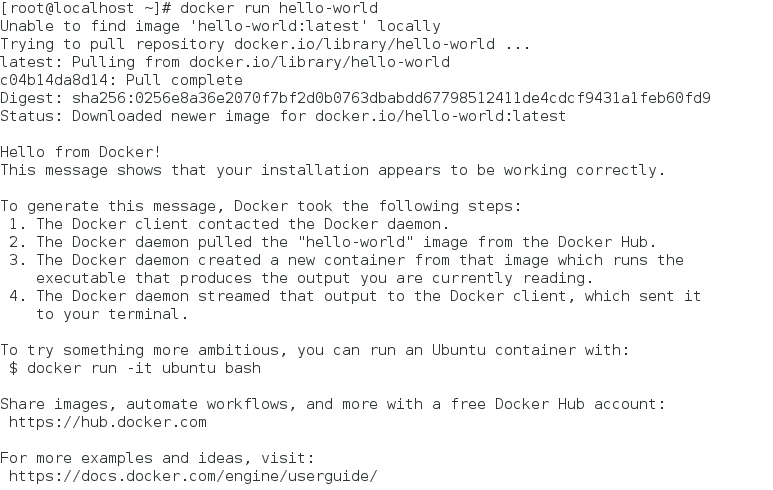
5．Start the Docker daemon.

$ sudo service dockerstart

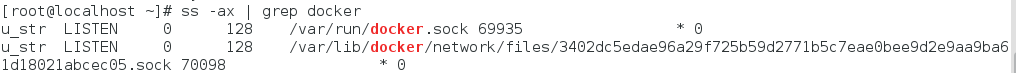
6．Verify docker is installed correctly by running a test image in a container.

验证docker安装正确

$ sudo docker run hello-world



docker默认使用的是unix socket



附：直接输入docker命令来查看所有的Options和Commands，查看某一个command的详细使用方法：dockerCOMMAND--help

四、Docker image详细介绍

在之前的介绍中，我们知道docker images 是docker的三大组件之一。

docker把下载的 images 存储到docker主机上，如果一个 image 不在主机上，docker会从一个镜像仓库下载，默认的仓库是 DOCKER HUB 公共仓库。

接下来将介绍更多关于docker images 的内容，包括：

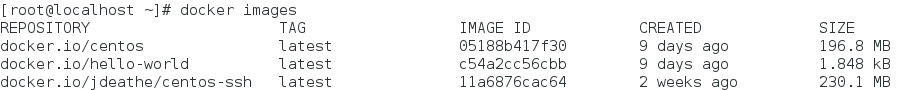
•使用和管理本地主机上的 images

•创建一个基础的 images

•上传 images 到docker hub （公共 images 仓库）

•列出本地主机上已经存在的 images

1）使用docker images 显示本机上的 images



在列出信息中，

•REPOSITORY:来自于哪个仓库，比如 docker.io/centos

•TAG的标记，比如 latest

•IMAGE ID:镜像它的 ID 号

•CREATED:创建时间

•SIZE:镜像的 SIZE

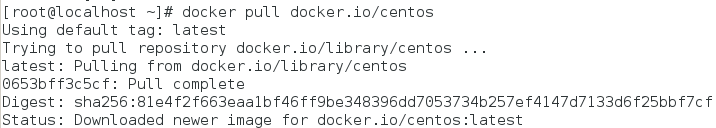
一个仓库可能有一个 images 的都个发行版，比如ubuntu，他们有 10.04 12.04 12.10 13.04 14.04,每个发行版的标记都不同，可以使用 tag 命令来指定 images

注：如果你不指定具体的发行版，比如仅使用ubuntu，那么docker会使用最新的发行版ubuntu:latest

提示：建议最好指定发行版，只有这样你才可以保证你真正使用的 image 是那个

2）获取 images

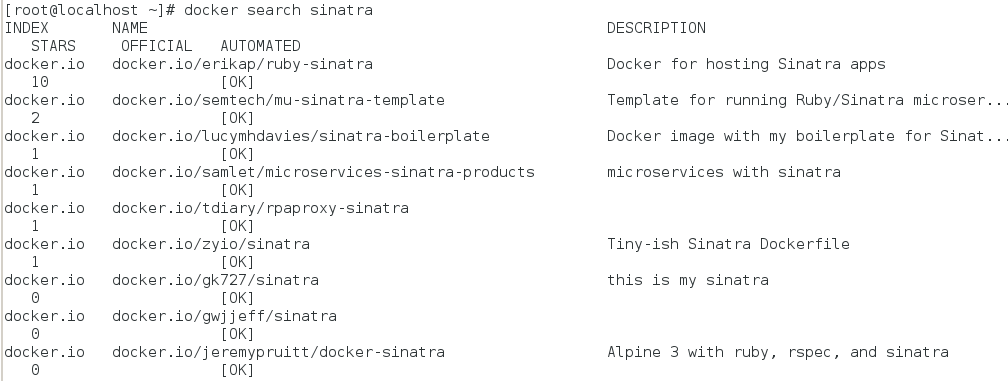
我们如何获取新的 images 呢？当我们启动容器使用的 image 不再本地主机上时，docker会自动下载他们。这很耗时，我们可以使用docker pull 命令来预先下载我们需要的 image 。下面的例子下载一个centos 镜像。



 这样当我们使用这个 image 来启动容器的时候，它就可以马上启动了。

3）查找 images

docker的一个特点是很多人因为各种不同的用途创建了各种不同的 images 。它们都被上传到了docker hub 共有仓库上，我们可以在docker hub 的网站上来查找它们。使用docker search 命令。比如，当我们需要 ruby 和sinatra作为 web 应用程序的开发时，我们使用docker search 来搜索合适的image ，使用关键字sinatra



我们看到返回了很多包含sinatra的 images 。其中包括 image 名字、描述、星级（表示该 image 的受欢迎程度）、是否官方创建、是否自动创建。官方的 images 是stackbrew项目组创建和维护的，automated 资源允许你验证 image 的来源和内容。

到目前为止，我们看到了 2种 images 资源。比如ubuntu，被称为基础或则根镜像。这些基础镜像是docker公司创建、验证、支持、提供。他们往往使用一个单词作为他们的名字。还有一种类型，比如我们选择的 training/sinatra镜像。它是由docker的用户创建并维护的，你可以通过指定 image 名字的前缀来指定他们，比如 training 。

3）下载 images

现在我们指定了一个 image ， training/sinatra，我们可以使用docker pull 命令来下载它



下载过程中，会输出获取镜像的每一层信息。

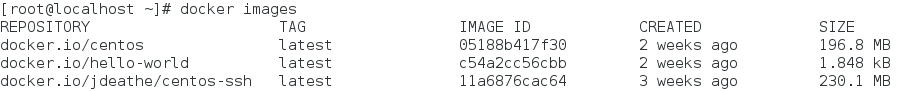
该命令实际上相当于#docker pull registry.hub.docker.com/ubuntu:12.04 命令，即从注册服

务器 registry.hub.docker.com 中的ubuntu仓库来下载标记为 12.04 的镜像。

有时候官方仓库注册服务器下载较慢，可以从其他仓库下载。从其它仓库下载时需要指定完整的仓库注册服务器地址。例如

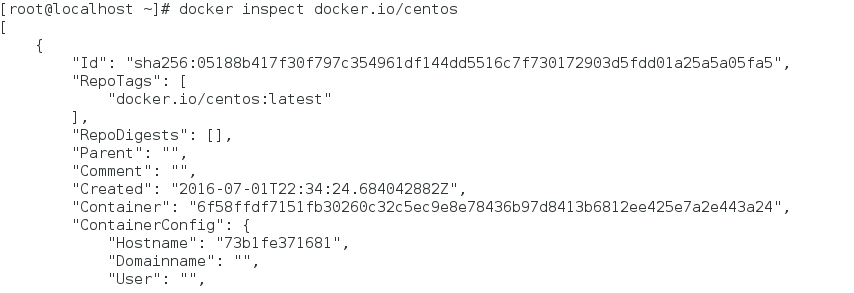
#docker pull dl.dockerpool.com:5000/ubuntu:12.04

查看镜像文件docker images



查看完整信息

当镜像下载成功后，你可以看到 12 位的 hash 值像 05188b417f30，这是下载完整的镜像的精简 ID，这些短的镜像 ID 是完整镜像 ID 的前 12 个字符--可以使用docker inspect 或者docker images --no-trunc来获得完整的镜像 ID



或



4）创建我们自己的 images

别人的镜像虽然好，但不一定适合我们。我们可以对他们做一些改变，有 2 个方法：

1.第一个方法：使用docker commit 来扩展一个 image

先使用 image 启动容器，更新后提交结果到新的 image 。



注意：记住容器的 ID ，稍后我们还会用到

在容器中添加mariadb-server应用。



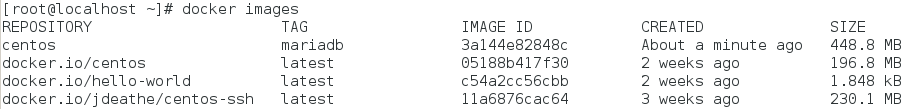
当结束后，我们使用 exit 来退出，现在我们的容器已经被我们改变了

使用dockercommint命令来提交相应的副本。



其中， -m 来指定提交的说明信息，跟我们使用的版本控制工具一样； -a 可以指定更新的用户信息；之后是用来创建镜像的容器的 ID；最后指定目标镜像的仓库名和 tag 信息。创建成功后会返回这个镜像的 ID信息。

使用docker images 来查看新创建的镜像。



之后，可以使用新的镜像来启动容器

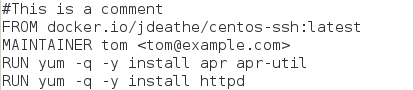


2.第二个办法：从dockerfile来创建 image

使用docker commit 来扩展一个 image 比较简单，但它不容易在一个团队中分享它。我们使用docker build 来创建一个新的 image 。为此，我们需要创建一个dockerfile，包含一些如何创建我们的image 的指令。现在，我们来创建一个目录和一个dockerfile



内容如下：



Dockerfile基本的语法是

使用#来注释

FROM指令告诉 Docker 使用哪个镜像作为基础（docker使用哪个 image 源）

MAINTAINER是维护者的信息

RUN开头的指令会在创建中运行，比如安装一个软件包，在这里使用 yum来安装了一些软件

编写完成Dockerfile后可以使用docker build 来生成镜像。

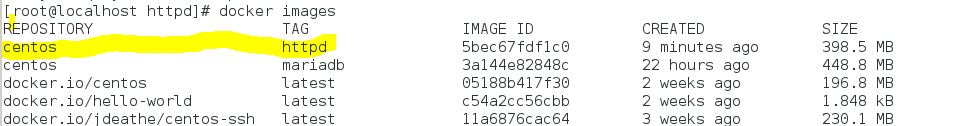


其中 -t 标记来添加 tag，指定新的镜像的用户信息。“.”是Dockerfile所在的路径（当前目录），也可以替换为一个具体的Dockerfile的路径。

可以看到 build 进程在执行操作。它要做的第一件事情就是上传这个Dockerfile内容，因为所有的操作都要依据Dockerfile来进行。然后，Dockfile中的指令被一条一条的执行。每一步都创建了一个新的容器，在容器中执行指令并提交修改（就跟之前介绍过的docker commit 一样）。当所有的指令都执行完毕之后，返回了最终的镜像 id。所有的中间步骤所产生的容器都被删除和清理了。

\*注意一个镜像不能超过 127 层

查看生成的镜像



从我们新建的 images 开启容器



还可以用docker tag 命令来修改镜像的标签。



3.从本地文件系统导入

要从本地文件系统导入一个镜像，可以使用openvz（容器虚拟化的先锋技术）的模板来创建：openvz的模板下载地址为<http://openvz.org/Download/template/precreated>。

先下载了一个centos-6-x86\_64.tar.gz的镜像，之后使用以下命令导入：



5）上传镜像

用户可以通过docker push 命令，把自己创建的镜像上传到仓库中来共享。例如，用户在 Docker Hub 上，完成注册后，可以推送自己的镜像到仓库中。

这里有两种访问可以创建和注册一个 Docker Hub 账户：

1.通过网站，（https://hub.docker.com）

2.通过命令行

你可以通过使用命令行输入 docker login 命令来创建一个 Docker Hub 账号

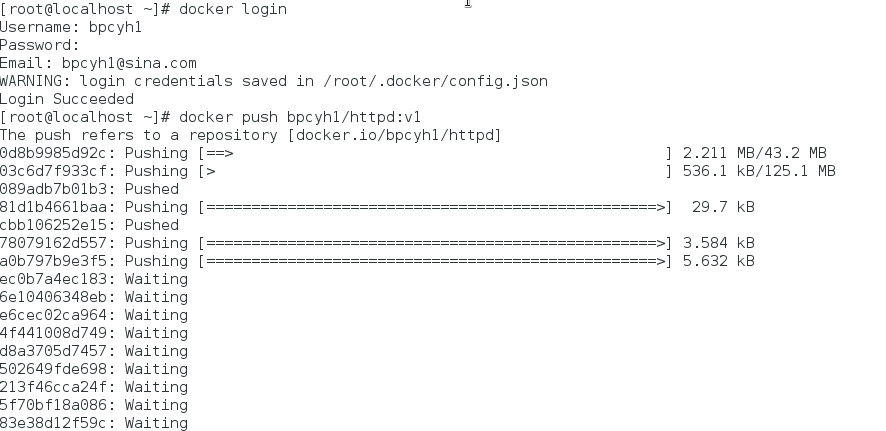
#docker login

邮箱确认

一旦你填写完毕表格，请查看你的电子邮件，通过点击欢迎信息中的链接来激活您的账户。

基本思路：

首先注册docker的账户，然后使用docker login登录。  
使用docker push可以将自己的镜像上传上去了



如果有其他的仓库,例如:

# docker push docker.sina.com.cn:5000/commit

6）用dcokerrmi移除本地 images

dockerrmi镜像名或镜像id

注意：在删除 images 之前要先用dockerrm删掉依赖于这个 images 的容器

7）存出和载入镜像

当需要把一台机器上的镜像迁移到另一台机器的时候，需要存出镜像与载入镜像。

存出镜像

如果要导出镜像到本地文件，可以使用docker save 命令。



载入镜像

可以使用docker load 从导出的本地文件中再导入到本地镜像库，例如



或



这将导入镜像以及其相关的元数据信息（包括标签等）

五、Docker 容器

容器是 Docker 又一核心概念,简单的说，容器是独立运行的一个或一组应用，以及它们的运行态环境。对应的，虚拟机可以理解为模拟运行的一整套操作系统（提供了运行态环境和其他系统环境）和跑在上面的应用。

本章将具体介绍如何来管理一个容器，包括创建、启动和停止等。

启动容器有两种方式，一种是基于镜像新建一个容器并启动，另外一个是将在终止状态（stopped）的容器重新启动。

新建并启动

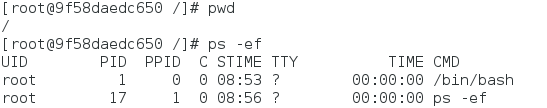
所需要的命令主要为docker run

下面的命令则启动一个 bash 终端，允许用户进行交互。

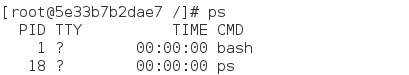


-t 选项让 Docker 分配一个伪终端（pseudo-tty）并绑定到容器的标准输入上， -i则让容器的标准输入保持打开(即交互式)，可以使用—name给容器起个形象的名称。

在交互模式下，用户可以通过所创建的终端来输入命令，例如



容器的核心为所执行的应用程序，所需要的资源都是应用程序运行所必需的。除此之外，并没有其它的资源。可以在伪终端中利用ps或 top 来查看进程信息。



可见，容器中仅运行了指定的 bash 应用。这种特点使得 Docker 对资源的利用率极高，是货真价实的轻量级虚拟化。

如果这个时候我们正常退出，logout 或者 exit 或者Ctrl+d或者Ctrl+c，dockerps–a 查看容器处于 Exit 状态如果需要正常退出可以使用 CTRL –p + CTRL -q ----就像先按 CTRL -p 然后 CTRL –q 退出伪终端

下面的命令输出一个“Hello World”，之后终止容器。



这跟在本地直接执行 /bin/echo 'hello world' 几乎感觉不出任何区别。

当利用docker run 来创建容器时，Docker 在后台运行的标准操作包括：

1.检查本地是否存在指定的镜像，不存在就从公有仓库下载

2.利用镜像创建并启动一个容器

3.分配一个文件系统，并在只读的镜像层外面挂载一层可读写层

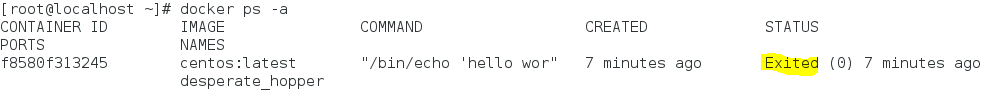
4.从宿主主机配置的网桥接口中桥接一个虚拟接口到容器中去

5.从地址池配置一个ip地址给容器

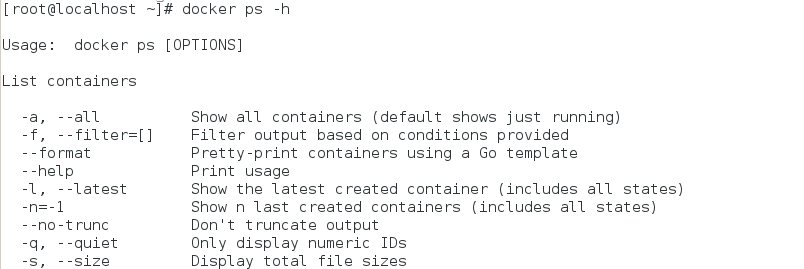
6.执行用户指定的应用程序

7.执行完毕后容器被终止

查看容器dockerps



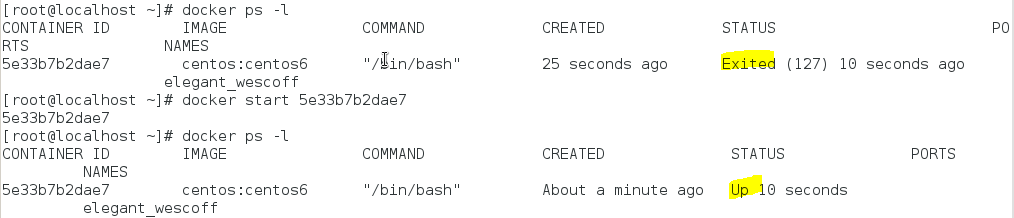
附：养成查看帮助的习惯



可以利用docker start 命令，直接将一个已经终止的容器启动运行。

# docker start [contraiID]

容器处于 Exited 状态，可以直接启动



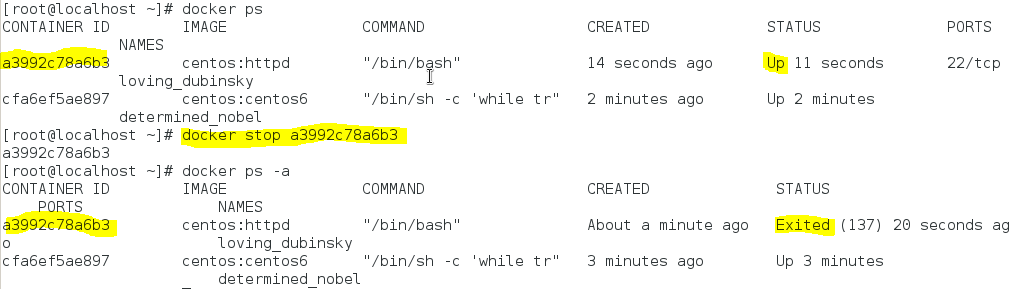
终止容器

# docker stop [容器 ID]

# docker kill [容器 ID]

可以使用docker stop 来终止一个运行中的容器。此外，当Docker容器中指定的应用终结时，容器也自动终止。例如对于前面所讲中启动了一个终端的容器，用户通过 exit 命令或Ctrl+d来退出终端时，所创建的容器立刻终止

终止状态的容器可以用dockerps -a 命令看到。例如



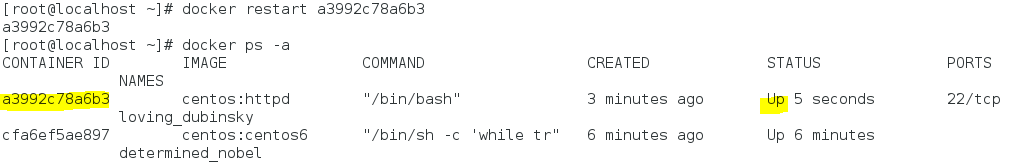
状态由 Up -> Exit

处于终止状态的容器，可以通过docker start 命令来重新启动。

重启容器

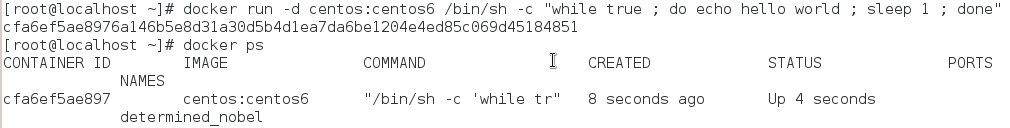
docker restart 命令会将一个运行态的容器终止，然后再重新启动它。

# docker restart [容器 ID]



守护态运行

更多的时候，需要让 Docker 容器在后台以守护态（Daemonized）形式运行。此时，可以通过添加 -d 参数来实现。例如下面的命令会在后台运行容器。



或



容器启动后会返回一个唯一的 id，也可以通过dockerps命令来查看容器信息。

1．docker run -d 运行提个新的容器，我们通过-d 命令让他作为一个后台运行

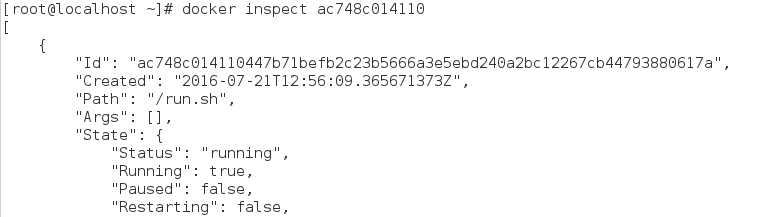
2．centos:centos6 是一个我们想要在内部运行命令的镜像

3．/bin/sh -c 是我们想要在容器内部运行的命令

4．while true; do echo hello weibo; sleep 1; done 这是一个简单的脚本，我们仅仅只是每秒打印一次 hello word 一直到我们结束它

用docker inspect查看容器的信息

命令格式：docker inspect 容器ID或容器名



用docker inspect查看容器的ip地址



用docker inspect查看容器执行的程序

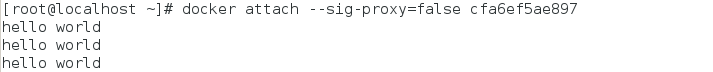


进入容器

在使用 -d 参数时，容器启动后会进入后台。某些时候需要进入容器进行操作，有很多种方法，包括使用docker attach 命令或nsenter命令。

使用docker attach进入容器

docker attach 是Docker自带的命令。下面示例如何使用该命令。



1．docker attach 允许我们进入后台进程.

2．--sig-proxy=false 不使用容器转发信号，允许我们使用 ctrl -c 来退出，执行dockerps查看在后台运行

但是使用 attach 命令有时候并不方便。当多个窗口同时 attach 到同一个容器的时候，所有窗口都会同步显示。当某个窗口因命令阻塞时,其他窗口也无法执行操作了。

也可以执行docker exec进入运行的容器

docker exec -it 容器ID/名称 /bin/bash

以上命令返回一个命令界面，exec代表直接在容器中运行命令



使用nsenter进入容器

安装

nsenter工具在util-linux包2.23版本后包含。如果系统中util-linux包没有该命令，可以按照下面的方法从源码安装

#wget<https://www.kernel.org/pub/linux/utils/util-linux/v2.24/util-linux-2.24.tar.gz>

#tar util-linux-2.24.tar.gz

#cd util-linux-2.24

# ./configure --without-ncurses&& make nsenter

#cpnsenter /usr/local/bin

nsenter可以访问另一个进程的名字空间。nsenter要正常工作需要有 root 权限

庆幸的是centos7使用的是util-linux-2.23，所以就直接使用系统提供的util-linux包了。



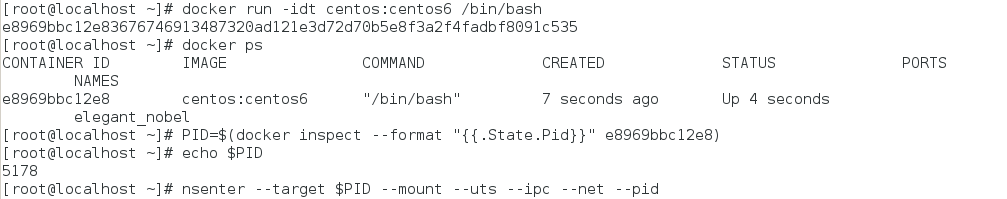
为了连接到容器，你还需要找到容器的第一个进程的PID，可以通过下面的命令获取。

PID=$(docker inspect --format "{{ .State.Pid }}" <container>)

通过这个PID，就可以连接到这个容器：

nsenter --target $PID --mount --uts --ipc --net --pid

下面给出一个完整的例子。



附：更简单的，建议大家下载 .bashrc\_docker，并将内容放到 .bashrc中。

#wget ~ https://github.com/yeasy/docker\_practice/raw/master/\_local/.bashrc\_docker

#echo "[ -f ~/.bashrc\_docker ] &&. ~/.bashrc\_docker" >> ~/.bashrc

#source ~/.bashrc

这个文件中定义了很多方便使用 Docker 的命令，例如docker-pid可以获取某个容器的PID；而docker-enter 可以进入容器或直接在容器内执行命令。

echo $(docker-pid<container>)

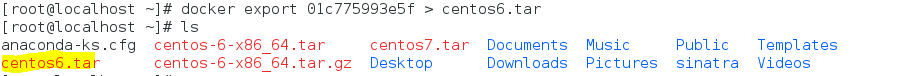
docker-enter <container> ls

容器导入和导出

导出容器

docker export [容器 id] > [导出文件]

如果要导出本地某个容器，可以使用docker export 命令。



这样将导出容器快照到本地文件

导入容器

可以使用docker import 从容器快照文件中再导入为镜像

# cat centos6.tar | docker import – centos6:test

#docker images

此外，也可以通过指定 URL 或者某个目录来导入，例如

#docker import <http://example.com/exampleimage.tgz>example/imagerepo

\*注：用户既可以使用docker load 来导入镜像存储文件到本地镜像库，也可以使用docker import 来导入一个容器快照到本地镜像库。这两者的区别在于容器快照文件将丢弃所有的历史记录和元数据信息（即仅保存容器当时的快照状态），而镜像存储文件将保存完整记录，体积也要大。此外，从容器快照文件导入时可以重新指定标签等元数据信息。

删除容器

可以使用dockerrm来删除一个处于终止状态的容器。

如果要删除一个运行中的容器，可以添加 -f 参数。Docker 会发送SIGKILL信号给容器。

# dockerrm [容器 id/容器 name]

批量删除多个容器

#dockerrm $(dockerps–a –q)