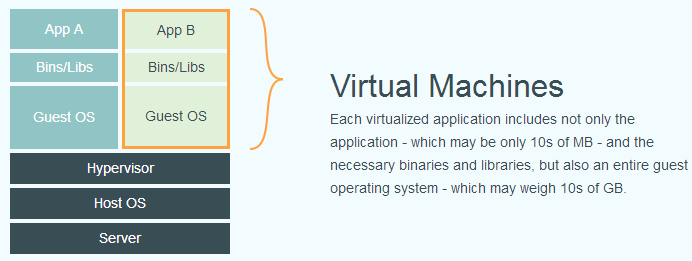
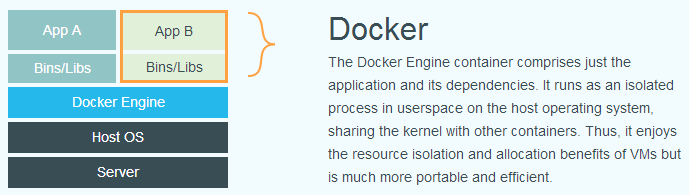
Docker简介及使用指南

Linux Container容器是一种内存虚拟化技术，可以提供轻量级虚拟化，以便隔离进程和资源，而且不需要提供指令解释机制以及全虚拟化的其他复杂性，相当于C++中的NameSpa-

ce。容器可以有效的将单个操作系统管理的资源划分成孤立的组中，以便更好的孤立在组之间平衡有冲突的资源使用需求。LXC(Linux容器技术)，在资源管理方面依赖于Linux内核的cgroups子系统，其实基于进程组的资源管理框架，可以为特定的进程组限定可以使用的资源，Docker的基础是Linux容器(LXC)等技术。

下图比较Docker和传统虚拟化方式的不同方式，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：





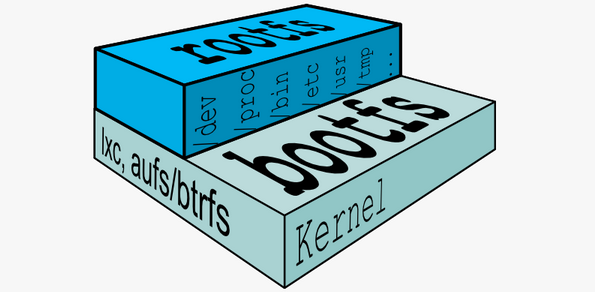
传统虚拟机不仅包括应用，而且需要整个客户端操作系统（GB）。但是Docker仅包括其应用和其依赖，Docker以隔离的进程的方式运行在主机操作系统的用户空间中，dockers之间共享Kernel，其实现了资源隔离和VM分配的好处，但更简单和有效。

# 1、Docker相关技术

## 1.1 AUFS

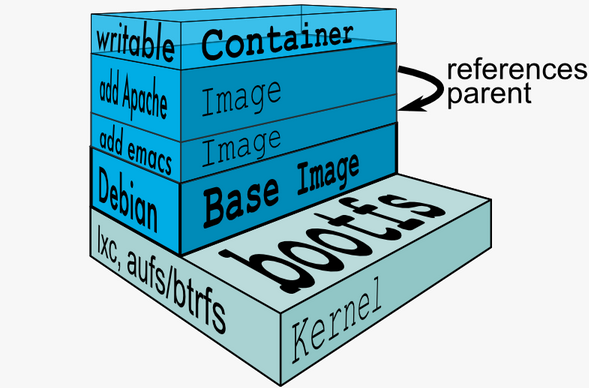
AUFS（AnotherUnionFS）是一种Union FS，简单来说就是支持将不同目录挂载到同一个虚拟文件系统下的文件系统。AUFS支持为每一个成员目录设定readonly,readwrite和writeout-able权限，同时AUFS里面有一个类似分层的概念，对readonly权限的branch可以逻辑上进行修改（增量，不是readonly部分的）。通常AUFS的用途是实现不借助LVM/RAID等将多个disk挂载到同一个目录下，另一个更常用的就是将readonly的branch和writable branch联合在一起。

启动Linux运行需要两个FS: rootfs+rootfs

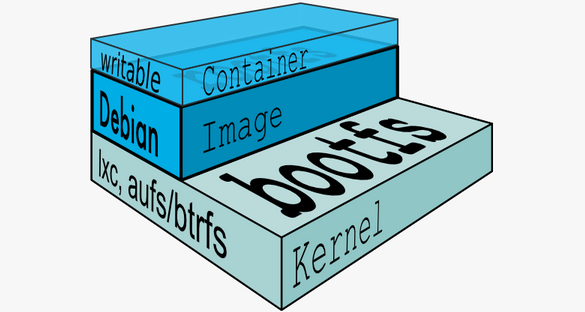


bootfs主要包含bootloader和kernel，bootloader主要引导加载kernel，当boot成功将kernel加载到内存后，bootfs就变成unmount了。rootfs包含的就是典型的Linux系统中的/dev,/proc,/bin,/etc等标准目录和文件。

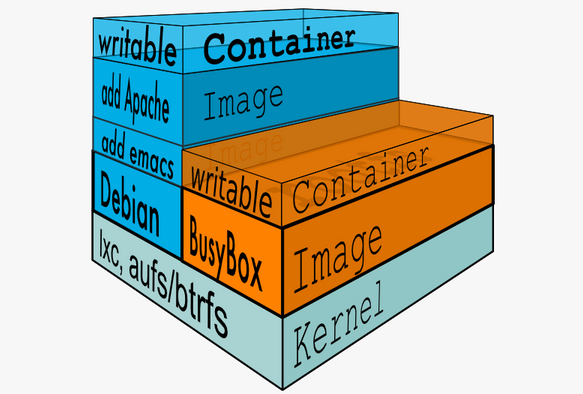
在典型的Linux启动后，首先将rootfs设置为readonly，进行一系列检查，然后将其切换为readwrite供用户使用。在Docker中，初始化时就是将rootfs以readonly方式加载并检查，然后接下来利用union mount的方式将一个readwrite文件系统挂载在readonly的rootfs之上，并且允许再次将下层的FS设定为readonly并且向上叠加，这样一组readonly和一个writable的结构构成一个container运行时态，每一个FS被称为一个FS层，如下图：



得益于AUFS的特性，每一个对readonly层文件/目录修改都只会存在于上层的writable层中，这样多个container可以共享readonly的FS层。所以Docker将readonly的FS层称为image，对于container而言整个rootfs都是read-write，但事实上所有的修改都写入最上层的writable层中，image不保存用户状态，只用于模板、新建和复制使用：



上层image依赖下层的image，因此Docker中把下层image称做父image，没有父image的image称为base image。因此想要从一个image启动一个container，Docker会先加载这个image的父image以及base image，用户的进程运行在writable layer中。所有parent image中的数据信息以及ID、网络和lxc管理的资源限制等具体container配置，构成一个Docker概念上的container，如下图：



## 1.2 Cgroups

Cgroups是Control groups的缩写，是Linux内核提供的一种可以限制、记录和隔离进程组（process groups）所使用的物理资源（如:cpu,memory,IO等）的机制。Cgroups适用多种应用场景，从单进程的资源控制，到实现操作系统层次的虚拟化（OS Level Virtualization），Cgroup提供了以下功能：

1. 限制进程组可以使用的资源数量，比如Memory子系统可以为进程组设定一个Memory使用上限，一旦进程组使用的内存达到限额再申请内存，就会发生OOM异常
2. 进程组的优先级控制，比如，可以使用CPU子系统为某个进程组分配特定的cpu share。
3. 记录进程组使用的资源数量（Accounting），比如使用cpuacct子系统记录某个进程组使用的cpu时间
4. 进程组隔离，比如使用ns子系统可以使不同的进程组使用不同的namespace，已达到隔离的目的，不同的进程组有各自的进程、网络、文件系统挂载空间
5. 进程组控制，比如使用freezer子系统可以将进程组挂起和恢复

Cgroups是LXC实现虚拟化所使用的资源管理手段，Docker构建在LXC之上，结构图如下：



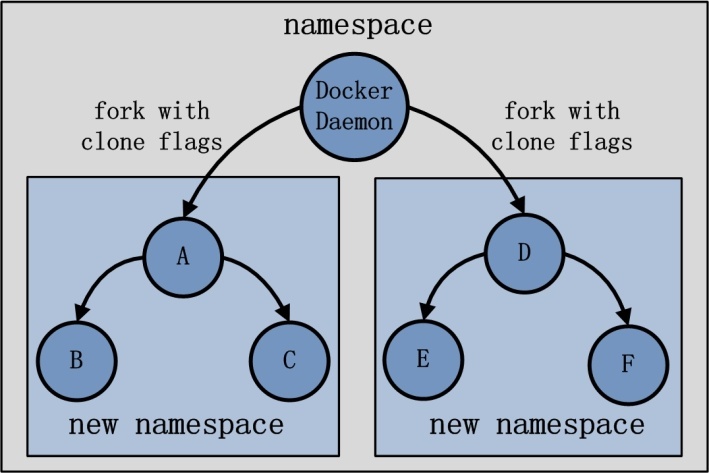
Cgroups组织结构图（缺少子系统的支撑组件）

## 1.3 NameSpace

Linux Namespace机制提供一种资源隔离方案，PID/IPD/NetWork等系统资源不再是全局性的，而是属于某个特定的Namespace，每个namespace下的资源对于其他Namespace下的资源都是透明，不可以见，如下图：



在Linux内核中提供多个Namespace，包括mnt、uts、network、sysvipc等将container的进程、网络、消息、文件系统、UTS和用户空间隔离开。



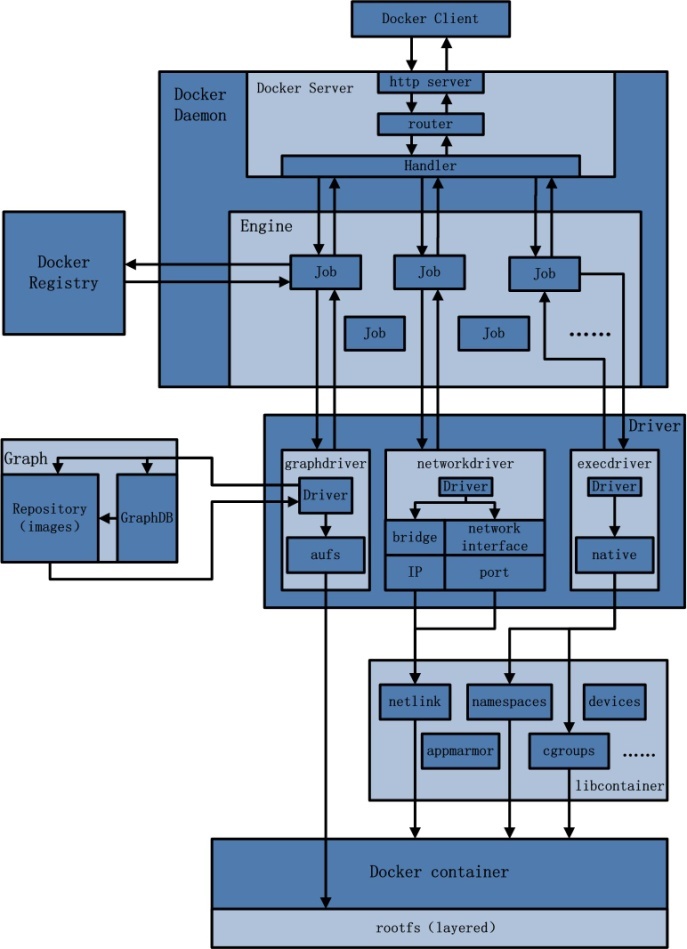
当用户创建并启动容器时，Docker Daemon会fork出容器中的第一个进程A，Docker Daemon执行fork时，会传入参数使子进程不再与父进程共享相同的命名空间，而是由Linux为其创建新的命名空间，从而保证子进程和父进程使用隔离的环境。如果子进程A再次fork出进程B和C，而fork没有传入相应的namespace参数标志，那么A和BC会共享一个命名空间。

与Namespace不同的是，cgroup的使用并不是在创建容器内进程时完成的，而是在容器内进程只后再使用cgroup，使得容器处于资源控制的状态。当容器内进程被创建完毕，Docker Daemon可以获知容器内进程的PID信息，随后将PID放置在Cgroup文件系统的指定位置，做相应的资源限制。

注：namespace主要负责命名空间的限制，而cgroup主要负责资源使用的限制。父进程通过fork创建子进程时，使用namespace技术，实现子进程和其他进程的命名空间隔离。子进程创建完毕之后，使用cgroup技术处理子进程，实现进程的资源使用限制。系统在子进程所处namespace内部，创建需要的隔离环境，如隔离网络栈等。

# 2、Docker的系统架构

Docker是一个C/S模式的架构，而Docker的后端是一个非常松耦合的架构，总体架构如下所示：



从图中可以看出，Docker软件架构分为以下几个部分：

## 2.1 Docker Client

Docker Client是Docker架构中用户用来和Docker Daemon建立通信的客户端，用户使用的可执行文件为docker，通过docker命令行工具可以发起众多管理container的请求。Docker Client发送容器请求后，由Docker Daemon接收并处理请求，当Docker Client接收到返回的请求响应并简单处理后，Docker Client一次完整的生命周期就结束了。当继续发送容器管理请求时，用户必须再次通过docker可执行文件创建Docker Client。

## 2.2 Docker Daemon

Docker架构中的一个常住后台的系统进程，其接收并处理Docker Client发送的请求。该守护进程在后台启动了一个Server，Server负责接受Docker Client发送的请求；接收请求后，Server通过路由与分发调度，找到相应的Handler来执行请求。Docker Daemon启动所使用的可执行文件也为docker，与Docker Client启动所使用的可执行文件Docker相同，在Docker命令执行时，通过传入的参数来判断Docker Daemon与Docker Client。

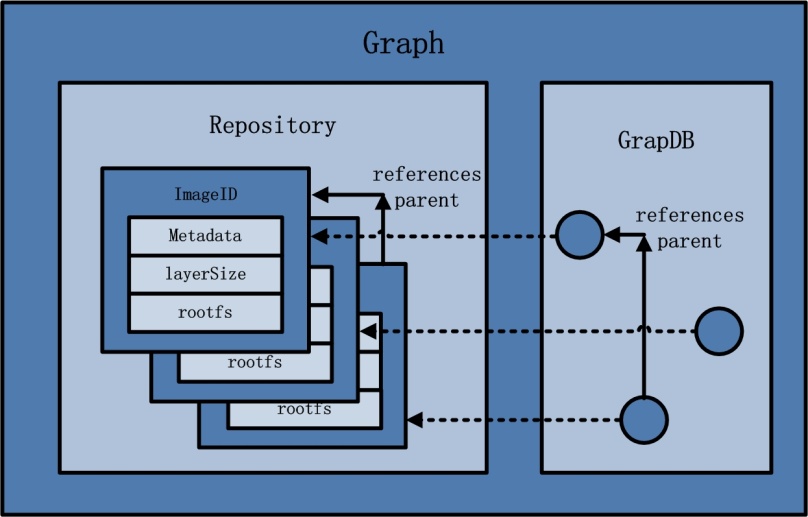
Docker Daemon 大致分为三个部分，Docker Server,Engine和Job。Docker Server在上面已经介绍。Engine是Docker架构中的运行引擎，同时也是Docker运行的核心模块，在Engine数据结构的涉及和实现的过程中，定义了handler对象，存储关于众多特定job的handler处理访问。Job是Docker架构中Engine内部最基本的工作执行单元，Docker可以做的每一项工作，都可以抽象为一个job。

## 2.3 Docker Registry

Docker Registry是一个存储容器镜像的仓库，而容器镜像是在容器被创建时，被加载用来初始化容器的文件架构和目录。在Docker运行过程中，Docker Daemon会与Docker Registry通信，并实现搜索镜像、下载镜像和上次镜像三个功能，对应Job的名称分别为search,pull与push。

## 2.4 Graph

Graph在Docker架构中扮演已下载容器镜像的保管者，以及已下载容器镜像之间关系的记录者。Graph的架构如下图：



GraphDB是构建在SQLite之上的小型图数据库，实现了节点的命名以及节点之间关联关系的记录。同时，Grap本地目录中，存储的信息包括该容器镜像的元数据、容器镜像的大小信息，以及该容器镜像所代表的rootfs。

## 2.5 Driver

Driver是Docker架构中的驱动模块，通过Driver驱动，Docker可以实现对Docker容器执行环境的定制。由于Docker运行的生命周期中，并非所有用户所有的操作都是针对Docker容器的管理，另外还有关于Docker运行信息的获取，Graph的存储与记录等。因此，为了将Docker容器的管理从Docker Daemon内部业务逻辑区分开来，设计了Driver层驱动来接管所有这部分请求。

在Driver的实现中，可以分为以下三类驱动：graphdriver、networkdriver和execdriver。 graphdriver主要用于完成容器镜像的管理，包括存储于获取。即当用户需要下载指定的容器镜像时，graphdriver将容器镜像存储在本地的指定目录，同时当用户需要使用指定的容器镜像来创建容器的rootfs时，graphdriver从本地镜像存储目录获取指定的容器镜像。

networkdriver的用途是完成Docker容器网络环境的配置，其中包括Docker启动时Docker环境创建网桥，Docker容器创建时为其创建专属虚拟网卡设备；以及为Docker容器分配IP、端口并与宿主机做端口映射，设置容器防火墙策略等。

exedriver作为Docker容器的执行驱动，负责创建容器运行命名空间，负责容器资源使用的统计与限制，负责容器内部进程的真正运行等。execdriver模式使用native驱动，不依赖LXC。

## 2.6 libcontainer

libcontainer是Docker架构中一个使用Go语言设计实现的库，直接访问内核中与容器相关的API。正是由于libcontainer的存在，Docker可以直接调用libcontainer，最终操纵容器的namespace、cgroups、apparmor、网络设备以及防火墙规则等。libcontainer提供了一整套标准的接口来满足上层对容器管理的需求，屏蔽了Docker上层对容器的直接管理。

## 2.7 Docker Container

Docker Container是Docker架构中服务交付的最终体现形式，Docker按照用户的需求与指令，定制相应的Docker容器：

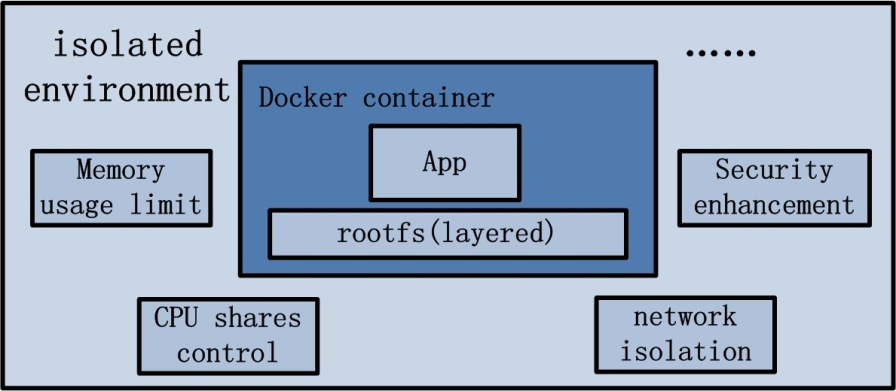
指定容器镜像，使得Docker容器可以自定义rootfs等文件系统

指定计算资源的配额，使得Docker容器使用指定的计算资源

通过网络配置及其安全策略，使得Docker容器拥有独立且安全的网络环境

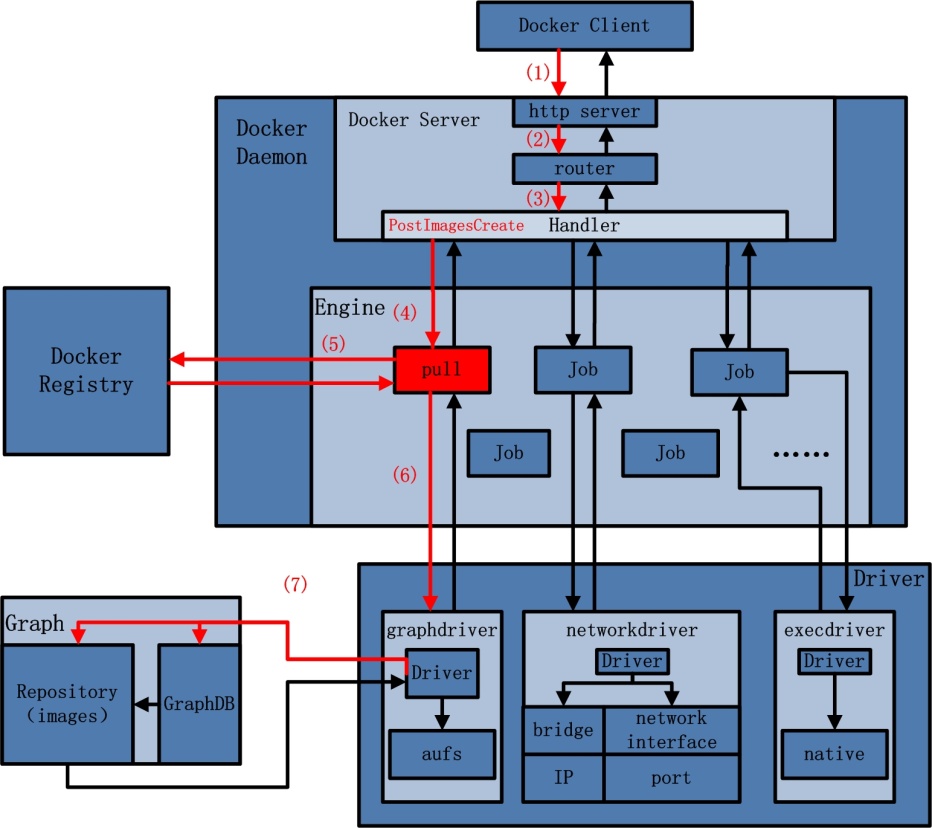
通过指定运行的命令，使得Docker容器执行指定的工作

Docker容器示意图如下：



## 2.8 Docker pull执行流程

Docker pull命令的作用为，从Docker Registry中下载指定的容器镜像，并存储在本地的Graph中，命令执行流程如下图所示：



图中红色箭头表示docker pull命令发起后，Docker所做的一系列运行，下面是具体的执行步骤：

1）Docker Client接受docker pull命令，解析完请求以及收集完请求参数之后，发送一个HTTP请求给Docker Server，HTTP请求方法为POST，请求URL为"/images/create? "+"xxx"；

2）Docker Server接受以上HTTP请求，并交给mux.Router，mux.Router通过URL以及请求方法来确定执行该请求的具体handler；

3）mux.Router将请求路由分发至相应的handler，具体为PostImagesCreate；

4）在PostImageCreate这个handler之中，一个名为"pull"的job被创建，并开始执行；

5）名为"pull"的job在执行过程中，执行pullRepository操作，即从Docker Registry中下载相应的一个或者多个image；

6）名为"pull"的job将下载的image交给graphdriver

7）graphdriver负责将image进行存储，一方创建graph对象，另一方面在GraphDB中记录image之间的关系。

# 3、Docker的安装和使用

## 3.1 内核升级

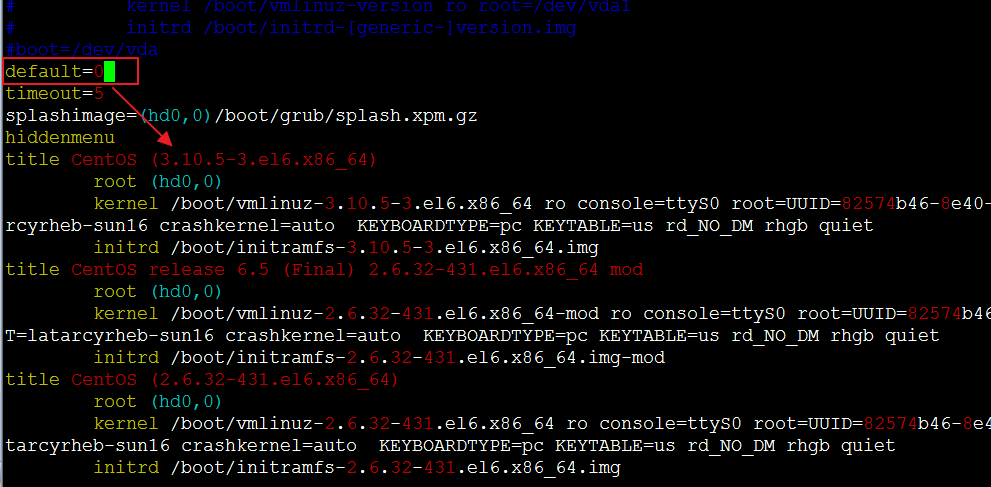
当前使用的Centos 6.5，其内核版本是2.6.32-431.el6.x86\_64，其中没有aufs。要在centos 6.5中安装及使用docker，要升级内核，命令如下：

$cd /etc/yun.repos.d

$wget http://www.hop5.in/yum/el6/hop5.repo

$yum install kernel-ml-aufs kernel-ml-aufs-devel

修改grup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）



重启系统后，centos 6.5内核版本3.10.5-3.el6.x86\_64。查看内核是否支持aufs：

grep aufs /proc/filesystems

查询结果为：nodev aufs，表示支持aufs。

## 3.2 Docker安装

通过yum命令安装

$yum -y install docker-io

启动docker

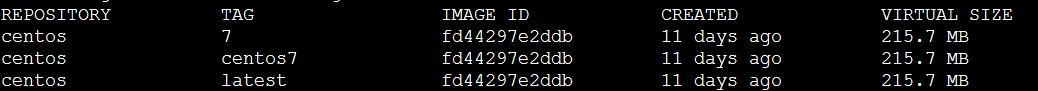
$service docker start //开启服务

$chkconfig docker on //开机启动

测试docker是否正常安装，获取最新的centos镜像，然后创建docker容器

$docker pull centos:latest

查看pull下来的images



创建一个Docker

$docker run -i -t centos /bin/bash

如果正常运行，会获得一个简单的bash提示，输入exit退出，如下图所示：



在命令行下，可运行Linux命令。

## 3.3 Docker的使用

### 3.3.1 Dockerfile

1）Dockerfile的命令格式为：

INSTRUCTION arguments

2）FROM命令：

FROM <image> // FROM <image>:<tag>

这个设置基本的镜像，为后续的命令使用，FROM命令为Dockerfile的第一条指令。比如

FROM ubuntu

3）RUN命令

RUN命令会在上面的FROM指定的镜像执行任何命令，然后提交结果，提交的镜像会在后面继续用到，两种格式：

RUN <command> (the command is run in a shell - `/bin/sh -c`)

或者

RUN ["executable", "param1", "param2" ... ] (exec form)

例如：

run yum -y install openssh-server openssh-clients

4）MAINTAINER命令

MAINTAINER <name>

MAINTAINER命令用来指定维护者的姓名和联系方式

5）ENTRYPOINT命令

有两种语法格式，一种是上面的shell方式

ENTRYPOINT cmd param1 param2 ...

另外一种是exec格式

ENTRYPOINT ["cmd", "param1", "param2"...]

ENTRYPOINT命令设置在容器启动时执行命令，例如：

ENTRYPOINT echo "Welcome"

6）USER命令

比如指定memcached的运行用户，比较好的方式如下：

ENTRYPOINT ["memcached"]

USER daemon

7）EXPOSE命令

EXPOSE命令可以设置一个端口在运行的镜像中暴露在外

EXPOSE <port> [<port>...]

比如memcached使用端口11211，可以把这个端口暴露在外，这样容器外可以看到这个端口并与其通信

EXPOSE 11211

8）ENV命令

用于设置环境变量

ENV <key> <value>

设置后，后续的RUN命令都可以使用

也可以通过在docker run时设置或修改环境变量

docker run -i -t --env name="tanky" ubuntu:newtest /bin/bash

9）ADD命令

把src复制文件到container的dest路径

ADD <src> <dest>

<src>是相对被构建的源目录的相对路径，可以是文件或目录的路径，也可以使一个远程的文件url

<dest>是container中的绝对路径

10）VOLUME命令

VOLUME ["<mountpoint>"]

如

VOLUME ["/data"]

创建一个挂载点用于共享目录

11）WORKDIR命令

WORKDIR /path/to/workdir

配置RUN,CMD,ENTRYPOINT命令设置当前的工作路径那个，可以设置多次，如果是相对路径，则相对前一个WORKDIR命令

例如：

WORKDIR /a WROKDIR b WORKDIR c RUN pwd

其实是在/a/b/c下执行pwd

12）CMD命令

有三种格式：

CMD ["executable","param1","param2"]

CMD ["param1","param2"]

CMD command param1 param2

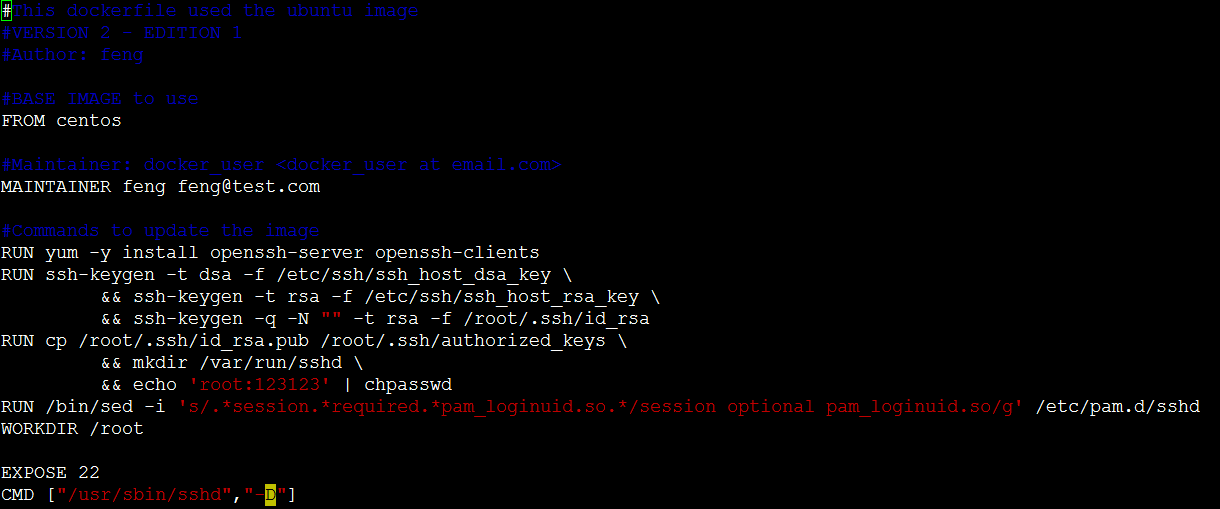
一个DockerFile中只能有一个CMD，如果有多个，只有最后一个生效。

常见的基本命令时：FROM / MAINTAINER/RUN/ENTRYPOINT/USER/PORT/ADD

13）DockerFile案例

创建安装SSH的Centos

dockerfile的文件格式及内容，如下所示：



使用Docker创建image的命令如下所示：

docker build -t soulmachie/centos - < myrepo/myapp/update.dockerfile

### 3.3.2 Docker中安装Tomcat

# 4、Hadoop Docker的配置和使用