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。

Docker能处理的事情包括：

[图片]隔离应用依赖

[图片]创建应用镜像并进行复制

[图片]创建容易分发的即启即用的应用

[图片]允许实例简单、快速地扩展

[图片]测试应用并随后销毁它们

Docker两个最重要的概念是镜像和容器。除此之外，链接和数据卷也很重要。我们先从镜像入手。

# 一、基本概念

## 镜像

Docker的镜像类似虚拟机的快照，但更轻量，非常非常轻量（下节细说）。

创建一个镜像，你可以拿一个镜像，对它进行修改来创建它的子镜像

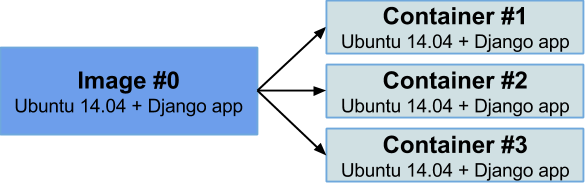
1. 在一个文件中指定一个基础镜像及需要完成的修改；
2. 或通过“运行”一个镜像，对其进行修改并提交。

镜像可以命名为类似ubuntu:latest、ubuntu:precise、django:1.6、django:1.7等等。

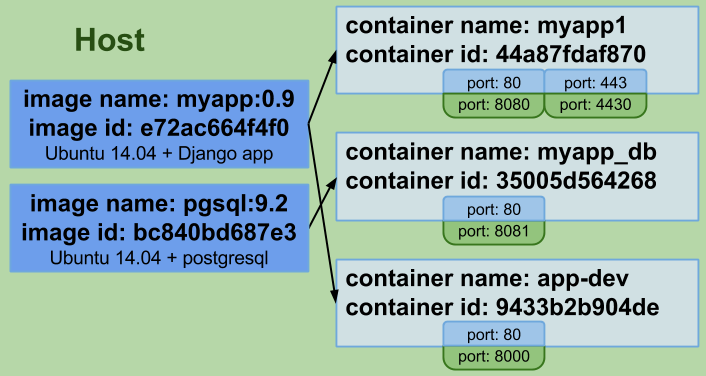
唯一ID，以及一个供人阅读的名字和标签对

## 容器

现在说容器了。你可以从镜像中创建容器，这等同于从快照中创建虚拟机，不过更轻量。应用是由容器运行的。



容器与虚拟机一样，是隔离的（有一点要注意，我稍后会讨论到）。它们也拥有一个唯一ID和唯一的供人阅读的名字。容器对外公开服务是必要的，因此Docker允许公开容器的特定端口。



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，运行类似ssh的东西来访问运行中的容器，不过docker exec命令解决了这个问题。

容器是设计来运行一个应用的，而非一台机器。

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

【容器和虚拟机的第二个巨大差异是：当你停止一个虚拟机时，可能除了一些临时文件，没有文件会被删除；当你停止一个Docker容器，对初始状态（创建容器所用的镜像的状态）做的所有变化都会丢失。】

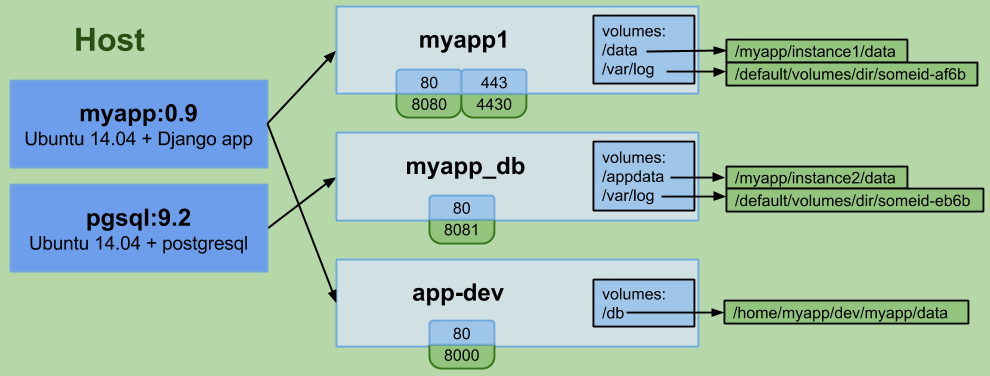
## 数据卷

数据卷让你可以不受容器生命周期影响进行数据持久化。

它们表现为容器内的空间，但实际保存在容器之外

Docker允许你定义应用部分和数据部分，并提供工具让你可以将它们分开。使用Docker时必须做出的最大思维变化之一就是：容器应该是短暂和一次性的。

**卷** 是针对容器的，你可以使用同一个镜像创建多个容器并定义不同的卷。卷保存在运行Docker的宿主文件系统上，你可以指定卷存放的目录，或让Docker保存在默认位置。



## 链接

容器启动时，将被分配一个随机的私有IP，其它容器可以使用这个IP地址与其进行通讯。

重要！！！

一是它提供了容器间相互通信的渠道，

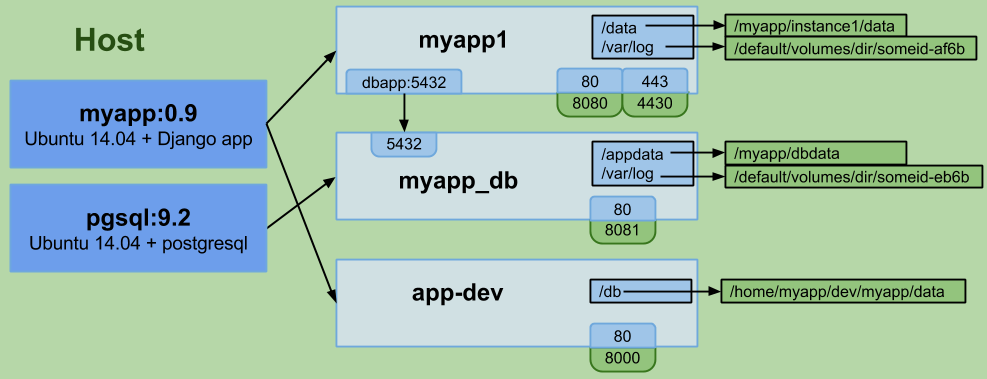
二是容器将共享一个本地网络。

实例问题：

我曾经碰到一个问题，在同一台机器上为两个客户启动两个elasticsearch容器，但保留集群名称为默认设置，结果这两台elasticsearch服务器立马变成了一个自主集群

要**开启容器间通讯**，Docker允许你在创建一个新容器时引用其它现存容器，在你刚创建的容器里被引用的容器将获得一个（你指定的）别名。我们就说，这两个容器链接在了一起。

Docker更进一步，要求你声明容器在被链接时要开放哪些端口给其他容器，否则将没有端口可用。



# 二、基础特性

可移植性

Docker允许你在一个镜像中指定卷和端口。从这个镜像创建的容器继承了这些设置。

镜像必须完全可移植，Docker不允许例外。

Docker不允许你在镜像上指定任何不可移植的内容。

## Docker如何完成它需要完成的东西？

cgroups和union文件系统。

Docker使用cgroup来提供容器隔离，

而union文件系统用于保存镜像并使容器变得短暂。

## Cgroups

这是Linux内核功能，它让两件事情变成可能：

[图片]限制Linux进程组的资源占用（内存、CPU）

[图片]为进程组制作 PID、UTS、IPC、网络、用户及装载命名空间

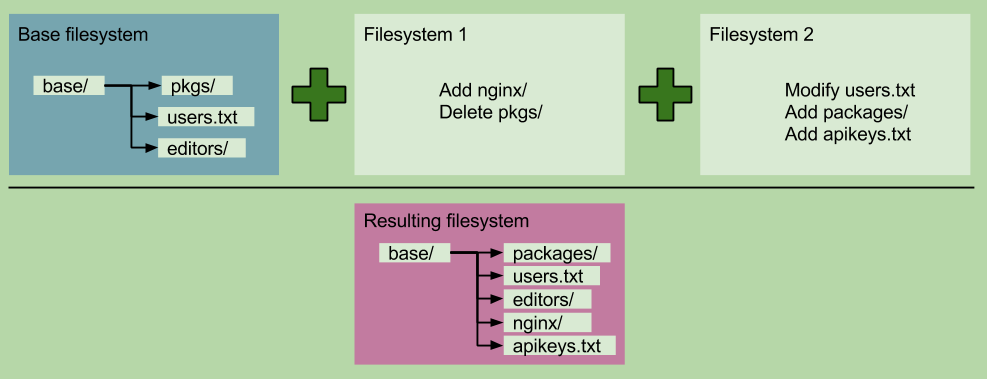
命名空间:一个PID命名空间允许它里面的进程使用隔离的PID，并与主PID命名空间独立开来，因此你可以在一个PID命名空间里拥有自己的PID为1的初始化进程。其他命名空间与此类似。

然后你可以使用cgroup创建一个环境，进程可以在其中运行，并与操作系统的其他进程隔离开，但这里的关键点是这个环境上的进程使用的是已经加载和运行的内核，因此额外支出与运行其他进程几乎是一样的。

## Union文件系统

Union文件系统允许通过union装载来达到一个分层的积累变化。

在union文件系统里，文件系统可以被装载在其他文件系统之上，其结果就是一个分层的积累变化。每个装载的文件系统表示前一个文件系统之后的变化集合，就像是一个diff。



截至2015年1月4日，Docker允许在union文件系统中使用aufs、btrfs或设备映射（device mapper）。

## ps

**镜像**只是一个json，它指定了从该镜像运行的容器的特性, union装载点保存在哪里，要公开什么端口等等。

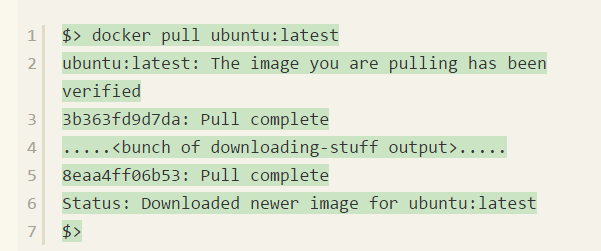
每个**镜像**与一个union文件系统相关联，每个Docker上的union文件系统都有一个上层，就像是计算机科技树（不像其他树有一大堆的家族）

# 三、安装

第一步，安装Docker。

Docker命令工具需要root权限才能工作. 你可以将你的用户放入docker组来避免每次都要使用sudo。

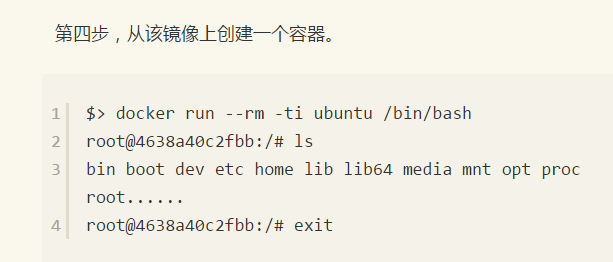
第二步，使用以下命令从公共registry下载一个镜像：



第三步，列出你的镜像：



第四步，从该镜像上创建一个容器。



--rm：告诉Docker一旦运行的进程退出就删除容器。这在进行测试时非常有用，可免除杂乱

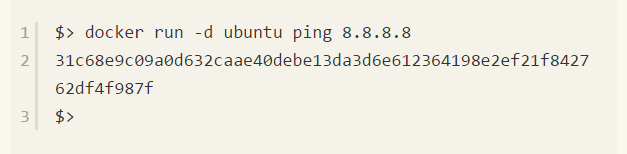
[图片]-ti：告诉Docker分配一个伪终端并进入交互模式。这将进入到容器内

ubuntu：这是容器立足的镜像

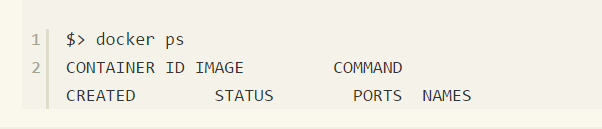
/bin/bash：要运行的命令，因为我们以交互模式启动，它将显示一个容器的提示符

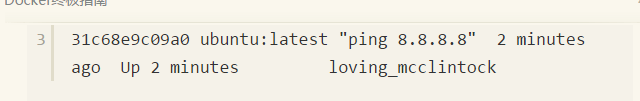
**在运行run命令时，你可指定链接、卷、端口、窗口名称（如果你没提供，Docker将分配一个默认名称）等等。**

现在，我们在后台运行一个容器：

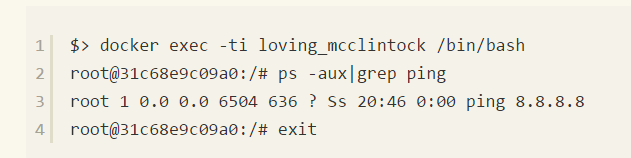


检查一下容器是否开始运行了：

****

****

**我们看看容器里正在发生什么：**

****

# 四、linux命名空间教学

chroot：将应用隔离到一个虚拟的私有root下

LXC——LinuX Containers，它是一个加强版的Chroot。简单的说，LXC就是将不同的应用隔离开来，这有点类似于chroot，而LXC在这之上更进了一步。

事实上，LXC所做的事情远不止**隔离**，它还包含**模板管理**、**冻结**以及**其它**更多的功能

LXC内部依赖Linux内核的3种隔离机制（isolation infrastructure）：

**1.Chroot**

**2.** **Cgroups**

**3.Namespaces**

**Linux的容器工具并没有提供一个类似黑盒般神秘的容器解决方案，而是提供单独隔离构件（isolation building block），统称为Namespaces。**

**Linux的3.12内核支持6种Namespace：**

**UTS: hostname（本文介绍）**

**IPC: 进程间通信 （之后的文章会讲到）**

**PID: "chroot"进程树（之后的文章会讲到）**

**NS: 挂载点，首次登陆Linux（之后的文章会讲到）**

**NET: 网络访问，包括接口（之后的文章会讲到）**

**USER: 将本地的虚拟user-id映射到真实的user-id（之后的文章会讲到）**