<https://zhuanlan.zhihu.com/p/187505981>

作为程序员我们应怎样理解docker？

**[容器技术](https://zhida.zhihu.com/search?content_id=129800958&content_type=Article&match_order=1&q=%E5%AE%B9%E5%99%A8%E6%8A%80%E6%9C%AF&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)的起源**

假设你们公司正在秘密研发下一个“今日头条”APP，我们姑且称为明日头条，程序员自己**从头到尾搭建了一套环境**开始写代码，写完代码后程序员要把代码交给测试同学测试，这时测试同学开始**从头到尾搭建这套环境**，测试过程中出现问题程序员也不用担心，大可以一脸无辜的撒娇，“明明在人家的环境上可以运行的”。

测试同学测完后终于可以上线了，这时运维同学又要重新**从头到尾搭建这套环境**，费了九牛二虎之力搭建好环境开始上线，糟糕，上线系统就崩溃了，这时心理素质好的程序员又可以施展演技了，“明明在人家的环境上可以运行的”。

从整个过程可以看到，不但我们重复搭建了三套环境还要迫使程序员转行演员浪费表演才华，典型的浪费时间和效率，聪明的程序员是永远不会满足现状的，因此又到了程序员改变世界的时候了，容器技术应运而生。

有的同学可能会说：“等等，先别改变世界，我们有虚拟机啊，VMware好用的飞起，先搭好一套虚拟机环境然后给测试和运维clone出来不就可以了吗？”

在没有容器技术之前，这确实是一个好办法，只不过这个办法还没有那么好。

先科普一下，现在[云计算](https://zhida.zhihu.com/search?content_id=129800958&content_type=Article&match_order=1&q=%E4%BA%91%E8%AE%A1%E7%AE%97&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)其底层的基石就是[虚拟机技术](https://zhida.zhihu.com/search?content_id=129800958&content_type=Article&match_order=1&q=%E8%99%9A%E6%8B%9F%E6%9C%BA%E6%8A%80%E6%9C%AF&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)，云计算厂商买回来一堆硬件搭建好数据中心后使用虚拟机技术就可以将硬件资源进行切分了，比如可以切分出100台虚拟机，这样就可以卖给很多用户了。

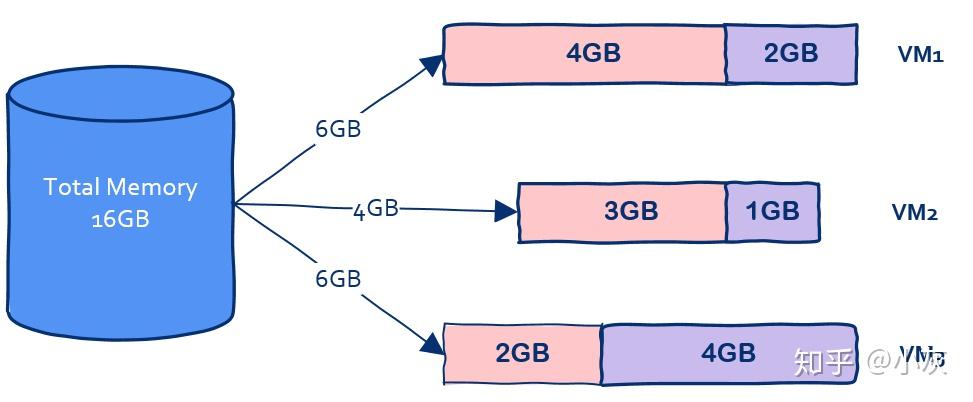
你可能会想这个办法为什么不好呢？

**容器技术 vs 虚拟机**

我们知道和一个单纯的应用程序相比，**操作系统是一个很重而且很笨的程序**，简称笨重，有多笨重呢？

我们知道操作系统运行起来是需要占用很多资源的，大家对此肯定深有体会，刚装好的系统还什么都没有部署，单纯的操作系统其磁盘占用至少几十G起步，内存要几个G起步。

假设我有一台机器，16G内存，需要部署三个应用，那么使用虚拟机技术可以这样划分：



在这台机器上开启三个虚拟机，每个虚拟机上部署一个应用，其中VM1占用2G内存，VM2占用1G内存，VM3占用了4G内存。

我们可以看到虚拟本身就占据了总共7G内存，因此**我们没有办法划分出更过虚拟机从而部署更多的应用程序**，可是我们部署的是应用程序，要用的也是应用程序而**不是操作系统**。

如果有一种技术可以让我们避免把内存浪费在“无用”的操作系统上岂不是太香？这是问题一，主要原因在于操作系统太重了。

还有另一个问题，那就是启动时间问题，我们知道操作系统重启是非常慢的，因为操作系统要从头到尾把该检测的都检测了该加载的都加载上，这个过程非常缓慢，动辄数分钟，因此操作系统还是太笨了。

那么有没有一种技术可以让我们获得虚拟机的好处又能克服这些缺点从而一举实现鱼和熊掌的兼得呢？

答案是肯定的，这就是容器技术。

**什么是容器**

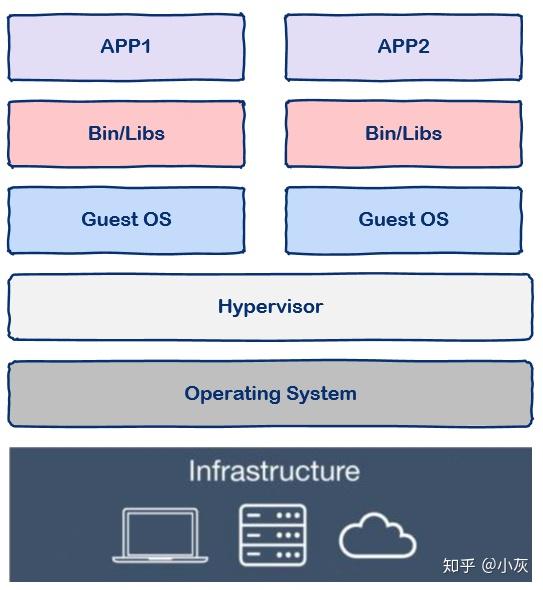
容器一词的英文是container，其实container还有集装箱的意思，集装箱绝对是商业史上了不起的一项发明，大大降低了海洋贸易运输成本。让我们来看看集装箱的好处：

* 集装箱之间相互隔离
* 长期反复使用
* 快速装载和卸载
* 规格标准，在港口和船上都可以摆放



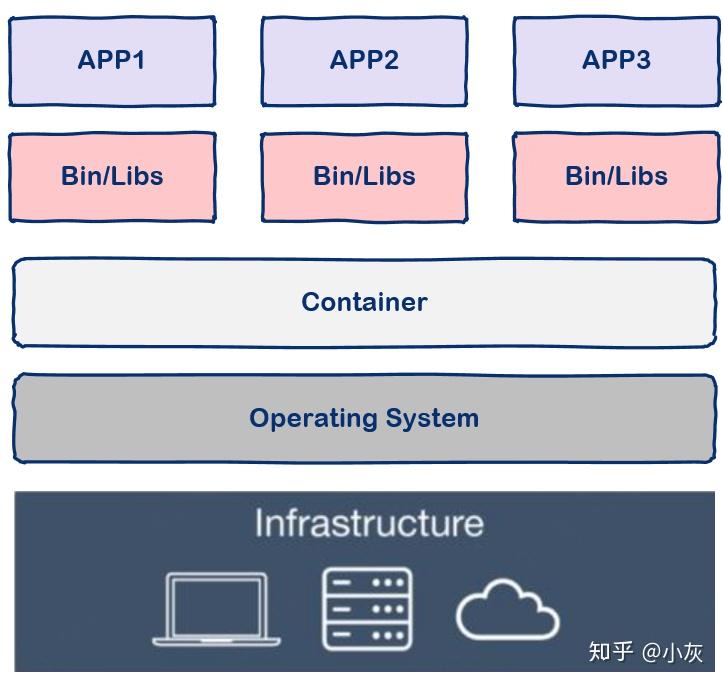
回到软件中的容器，其实容器和集装箱在概念上是很相似的。

现代软件开发的一大目的就是隔离，应用程序在运行时相互独立互不干扰，这种隔离实现起来是很不容易的，其中一种解决方案就是上面提到的虚拟机技术，通过将应用程序部署在不同的虚拟机中从而实现隔离。



但是虚拟机技术有上述提到的各种缺点，那么容器技术又怎么样呢？

与虚拟机通过操作系统实现隔离不同，容器技术**只隔离应用程序的运行时环境但容器之间可以共享同一个操作系统**，这里的运行时环境指的是程序运行依赖的各种库以及配置。



从图中我们可以看到容器更加的**轻量级且占用的资源更少**，与操作系统动辄几G的内存占用相比，容器技术只需数M空间，因此我们可以在同样规格的硬件上**大量部署容器**，这是虚拟机所不能比拟的，而且不同于操作系统数分钟的启动时间容器几乎瞬时启动，容器技术为**打包服务栈**提供了一种更加高效的方式，So cool。

那么我们该怎么使用容器呢？这就要讲到docker了。

注意，容器是一种通用技术，docker只是其中的一种实现。

**什么是docker**

docker是一个用Go语言实现的开源项目，可以让我们方便的创建和使用容器，docker将程序以及程序所有的依赖都打包到docker container，这样你的程序可以在任何环境都会有一致的表现，这里程序运行的依赖也就是容器就好比集装箱，容器所处的操作系统环境就好比货船或港口，**程序的表现只和集装箱有关系(容器)，和集装箱放在哪个货船或者哪个港口(操作系统)没有关系**。

因此我们可以看到docker可以屏蔽环境差异，也就是说，只要你的程序打包到了docker中，那么无论运行在什么环境下程序的行为都是一致的，程序员再也无法施展表演才华了，**不会再有“在我的环境上可以运行”**，真正实现“build once, run everywhere”。

此外docker的另一个好处就是**快速部署**，这是当前互联网公司最常见的一个应用场景，一个原因在于容器启动速度非常快，另一个原因在于只要确保一个容器中的程序正确运行，那么你就能确信无论在生产环境部署多少都能正确运行。

**如何使用docker**

docker中有这样几个概念：

* dockerfile
* image
* container

实际上你可以简单的把image理解为可执行程序，container就是运行起来的进程。

那么写程序需要源代码，那么“写”image就需要dockerfile，dockerfile就是image的源代码，docker就是"编译器"。

因此我们只需要在dockerfile中指定需要哪些程序、依赖什么样的配置，之后把dockerfile交给“编译器”docker进行“编译”，也就是docker build命令，生成的可执行程序就是image，之后就可以运行这个image了，这就是docker run命令，image运行起来后就是docker container。

具体的使用方法就不再这里赘述了，大家可以参考docker的官方文档，那里有详细的讲解。

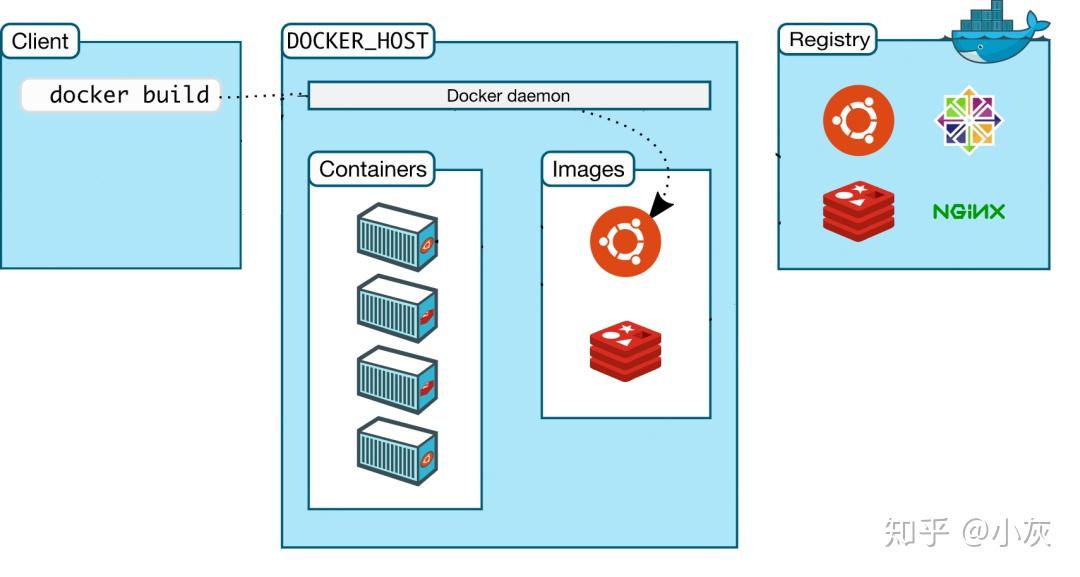
**docker是如何工作的**

实际上docker使用了常见的CS架构，也就是client-server模式，docker client负责处理用户输入的各种命令，比如docker build、docker run，真正工作的其实是server，也就是docker demon，值得注意的是，docker client和docker demon可以运行在同一台机器上。

接下来我们用几个命令来讲解一下docker的工作流程：

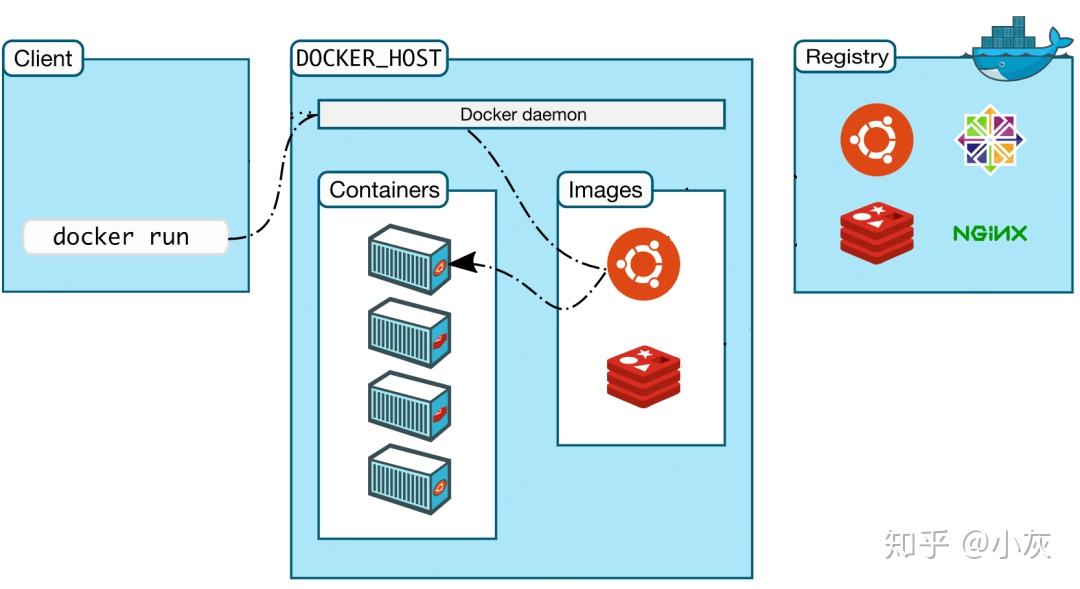
**1，docker build**

当我们写完dockerfile交给docker“编译”时使用这个命令，那么client在接收到请求后转发给docker daemon，接着docker daemon根据dockerfile创建出“可执行程序”image。



**2，docker run**

有了“可执行程序”image后就可以运行程序了，接下来使用命令docker run，docker daemon接收到该命令后找到具体的image，然后加载到内存开始执行，image执行起来就是所谓的container。



**3，docker pull**

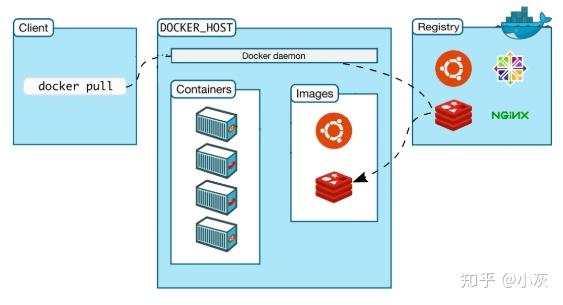
其实docker build和docker run是两个最核心的命令，会用这两个命令基本上docker就可以用起来了，剩下的就是一些补充。

那么docker pull是什么意思呢？

我们之前说过，docker中image的概念就类似于“可执行程序”，我们可以从哪里下载到别人写好的应用程序呢？很简单，那就是APP Store，即应用商店。与之类似，既然image也是一种“可执行程序”，那么有没有"Docker Image Store"呢？答案是肯定的，这就是[Docker Hub](https://zhida.zhihu.com/search?content_id=129800958&content_type=Article&match_order=1&q=Docker+Hub&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)，docker官方的“应用商店”，你可以在这里下载到别人编写好的image，这样你就不用自己编写dockerfile了。

docker registry 可以用来存放各种image，公共的可以供任何人下载image的仓库就是docker Hub。那么该怎么从Docker Hub中下载image呢，就是这里的docker pull命令了。

因此，这个命令的实现也很简单，那就是用户通过docker client发送命令，docker daemon接收到命令后向docker registry发送image下载请求，下载后存放在本地，这样我们就可以使用image了。



最后，让我们来看一下docker的底层实现。

**docker的底层实现**

docker基于Linux内核提供这样几项功能实现的：

* **[NameSpace](https://zhida.zhihu.com/search?content_id=129800958&content_type=Article&match_order=1&q=NameSpace&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)**  
  我们知道Linux中的PID、IPC、网络等资源是全局的，而NameSpace机制是一种资源隔离方案，在该机制下这些资源就不再是全局的了，而是属于某个特定的NameSpace，各个NameSpace下的资源互不干扰，这就使得每个NameSpace看上去就像一个独立的操作系统一样，但是只有NameSpace是不够。
* **Control groups**  
  虽然有了NameSpace技术可以实现资源隔离，但进程还是可以不受控的访问系统资源，比如CPU、内存、磁盘、网络等，为了控制容器中进程对资源的访问，Docker采用control groups技术(也就是cgroup)，有了cgroup就可以控制容器中进程对系统资源的消耗了，比如你可以限制某个容器使用内存的上限、可以在哪些CPU上运行等等。

有了这两项技术，容器看起来就真的像是独立的操作系统了。

**总结**

docker是目前非常流行的技术，很多公司都在生产环境中使用，但是docker依赖的底层技术实际上很早就已经出现了，现在以docker的形式重新焕发活力，并且能很好的解决面临的问题，希望本文能对大家理解docker有所帮助