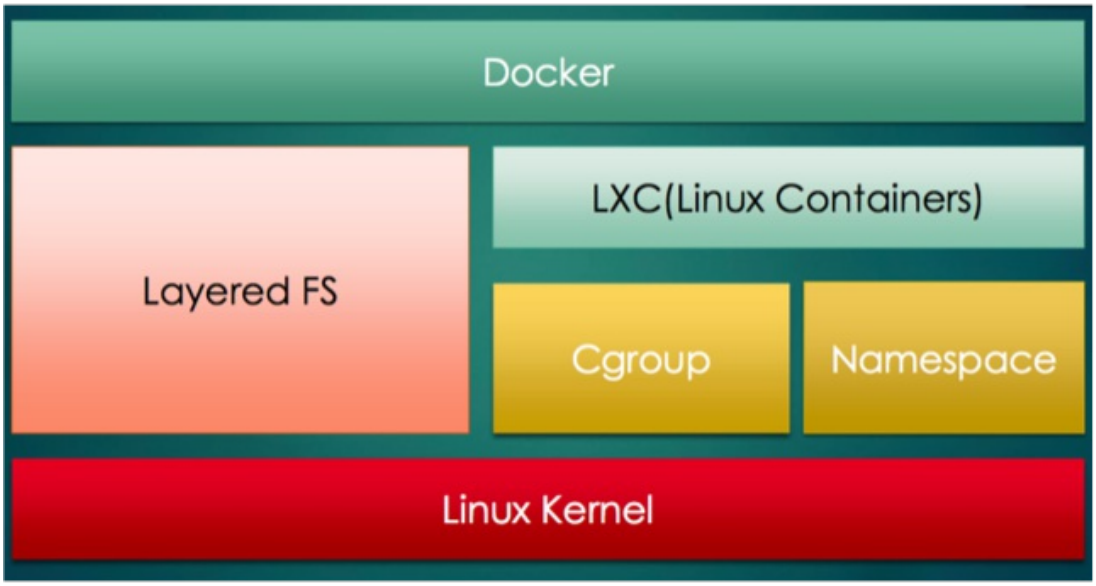
Docker 的一些介绍

Docker简介

Docker是一个由GO语言开发的，基于 LXCs（Linux containers） 的高级容器引擎，实现了一种应用程序级别的隔离，源代码托管在Github上,遵从Apache2.0协议开源。

Container技术并非Docker的创新，很多云服务商都采用了类似这种轻量级的虚拟化技术，但Docker是第一个将这这种Container技术大规模开源并被社区广泛接受的。



> - Namespace主要负责资源隔离，可以保障一个容器中运行一个进程而且不能看到和影响容器外的其它进程。
> - cgroups 实现了对资源的配额和度量
> - 借助于namespace的隔离机制和cgroup限额功能，LXC提供了一套统一的API和工具来建立和管理container
> - AUFS技术利用了写时拷贝技术，只读部分是Image，可写部分是container。允许read-only和read-write目录并存；可以实现把多个不同目录的内容合并在一起。

Docker的核心组件

1. Docker镜像

Docker 镜像是 Docker 容器运行时的只读模板，每一个镜像由一系列的层 (layers) 组成。

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。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。

2. Docker仓库

类似于Github的一种代码仓库，同样，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。Docker 仓库是 Docker 的分发部分。

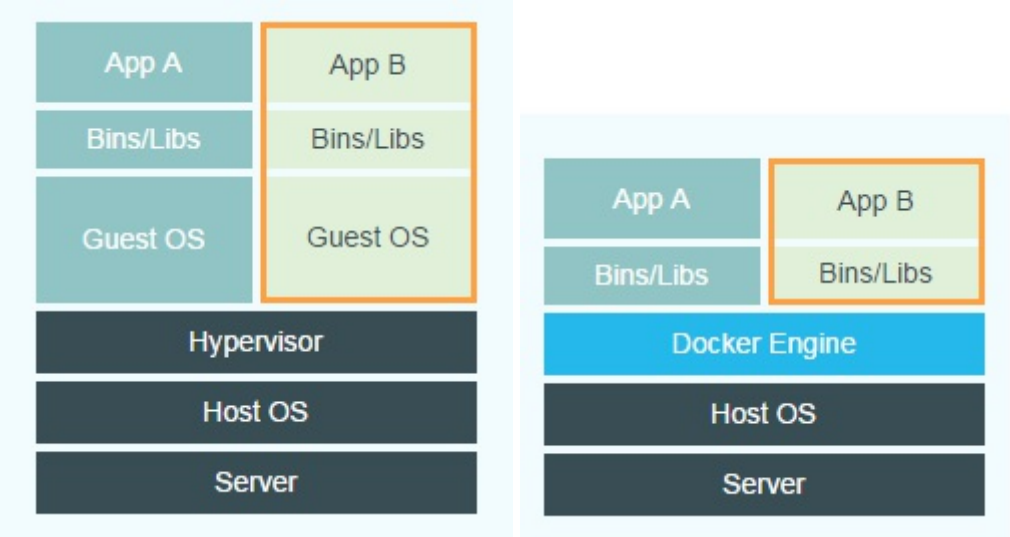
3. Docker容器

Docker 容器和文件夹很类似，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，Docker 容器是 Docker 的运行部分。

与虚拟机对比

1. 实现原理

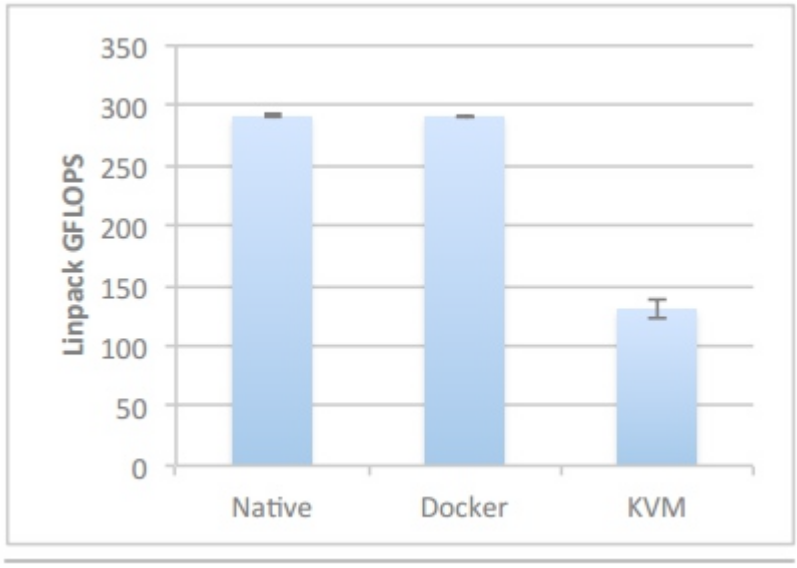


通过docker和虚拟机实现原理的比较，我们大致可以得出一些结论：

1. docker有着比虚拟机更少的抽象层。由于docker不需要Hypervisor实现硬件资源虚拟化，运行在docker容器上的程序直接使用的都是实际物理机的硬件资源。因此在CPU、内存利用率上docker将会在效率上有优势，具体的效率对比在下几个小节里给出。在IO设备虚拟化上，docker的镜像管理有多种方案，比如利用Aufs文件系统或者Device Mapper实现docker的文件管理，各种实现方案的效率略有不同。

2. docker利用的是宿主机的内核，而不需要Guest OS。因此，当新建一个容器时，docker不需要和虚拟机一样重新加载一个操作系统内核。我们知道，引导、加载操作系统内核是一个比较费时费资源的过程，当新建一个虚拟机时，虚拟机软件需要加载Guest OS，这个新建过程是分钟级别的。而docker由于直接利用宿主机的操作系统，则省略了这个过程，因此新建一个docker容器只需要几秒钟。另外，现代操作系统是复杂的系统，在一台物理机上新增加一个操作系统的资源开销是比较大的，因此，docker对比虚拟机在资源消耗上也占有比较大的优势。事实上，在一台物理机上我们可以很容易建立成百上千的容器，而只能建立几个虚拟机。

2. 计算效率



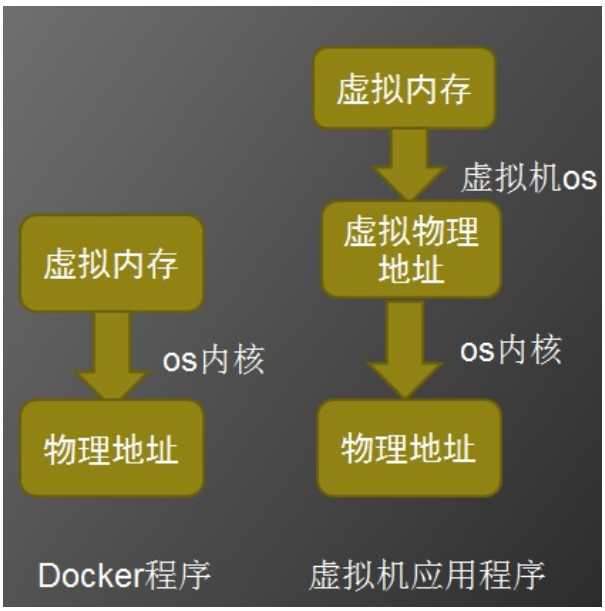
可见docker相对于物理机其计算能力几乎没有损耗，而虚拟机对比物理机则有着非常明显的损耗。虚拟机的计算能力损耗在50%左右。

为什么会有这么大的性能损耗呢？

> 一方面是因为虚拟机增加了一层虚拟硬件层，运行在虚拟机上的应用程序在进行数值计算时是运行在Hypervisor虚拟的CPU上的

> 另外一方面是由于计算程序本身的特性导致的差异。虚拟机虚拟的cpu架构不同于实际cpu架构，数值计算程序一般针对特定的cpu架构有一定的优化措施，虚拟化使这些措施作废，甚至起到反效果。

3. 内存访问效率



内存访问效率的比较相对比较复杂一点，主要是内存访问有多种场景：

> （1）大批量的，连续地址块的内存数据读写。这种测试环境下得到的性能数据是内存带宽，性能瓶颈主要在内存芯片的性能上；

> （2）随机内存访问性能。这种测试环境下的性能数据主要与内存带宽、cache的命中率和虚拟地址与物理地址转换的效率等因素有关。

在应用程序内存访问上，虚拟机的应用程序要进行2次的虚拟内存到物理内存的映射，读写内存的代价比docker的应用程序高。

4. 启动时间及资源耗费

无论从启动时间还是从启动资源耗费角度来说。docker直接利用宿主机的系统内核，避免了虚拟机启动时所需的系统引导时间和操作系统运行的资源消耗。利用docker能在几秒钟之内启动大量的容器，这是虚拟机无法办到的。快速启动、低系统资源消耗的优点使docker在弹性云平台和自动运维系统方面有着很好的应用前景。

5. Docker的劣势

1. 资源隔离方面不如虚拟机，docker是利用cgroup实现资源限制的，只能限制资源消耗的最大值，而不能隔绝其他程序占用自己的资源。
2. 安全性问题。docker目前并不能分辨具体执行指令的用户，只要一个用户拥有执行docker的权限，那么他就可以对docker的容器进行所有操作，不管该容器是否是由该用户创建。比如A和B都拥有执行docker的权限，由于docker的server端并不会具体判断docker cline是由哪个用户发起的，A可以删除B创建的容器，存在一定的安全风险。
3. docker目前还在版本的快速更新中，细节功能调整比较大。一些核心模块依赖于高版本内核，存在版本兼容问题
4. 真实投入生产，还需要多种开源组件和技术的支持，如kubernetes管理容器、etcd管理存储、应用打包技术dockerfile、容器间的网络管理flannel、私有仓库的构建、持续集成jenkins的结合、监控docker的工具等等。

Docker的应用场景

1. 权衡资源隔离和低消耗的场景

- 资源隔离：比如限制应用最大内存使用量，或者资源加载隔离，彼此资源占用互相不影响等。
- 低消耗：虚拟化本身带来的损耗需要尽量的低

基于以上两点，我们不可能在一台机器上开 500 个虚拟机，虽然可以在资源隔离方面做的很好，但这种虚拟化本身带来的资源消耗太严重。

另一个方面，我们可以考虑使用语言级别沙箱，虽然这种「虚拟化」本身的消耗可以低到忽略不计，但是资源隔离方面绝对是噩梦，比如你打算在一个 JVM 里隔离内存的使用。

而 Docker 很好的权衡了两者，即拥有不错的资源隔离能力，又有很低的虚拟化开销。

2. 从轻量级工具、持续集成演进到微服务架构

早在14年国内有一些公司，开始尝试docker，当时毕竟docker是一个新事物，很多新特性方面的优点，并没有被大大的利用起来，这个也可以理解。那时docker对一些企业的价值在于计算的轻量级，也就是对于一些计算型的任务，通过docker的形式来分发，部署快，隔离性好，这样的任务包括：消息传递，图像处理等。

14下半年到15年初，docker的价值被更大化，应用的运行，服务的托管，外界的接受度也变高，国内也出现了一些startup公司，比如DaoCloud,灵雀云等。但这些仅仅是这些公司的第一步，后续紧跟的更多的是基于代码与镜像之间的CI/CD,缩减开发测试发布的流程，这方面的实践逐渐成熟。

微服务架构的兴起。微服务会对现阶段的软件架构有一些冲击，同样也是软件系统设计方法论的内容。这些方面国外讨论的要多一些，相信这一点也会近年来多家公司发力的地方。

3. 需要配置多种不同环境

这是Docker公司宣传的Docker的主要使用场景。虚拟机的最大好处是能在你的硬件设施上运行各种配置不一样的平台（软件、系统），Docker在降低额外开销的情况下提供了同样的功能。它能让你将运行环境和配置放在代码中然后部署，同一个Docker的配置可以在不同的环境中使用，这样就降低了硬件要求和应用环境之间耦合度。

4. 快速部署和整合服务器

正如通过虚拟机来整合多个应用，Docker隔离应用的能力使得Docker可以整合多个服务器以降低成本。由于没有多个操作系统的内存占用，以及能在多个实例之间共享没有使用的内存，Docker可以比虚拟机提供更好的服务器整合解决方案。在虚拟机之前，引入新的硬件资源需要消耗几天的时间。Docker的虚拟化技术将这个时间降到了几分钟，Docker只是创建一个容器进程而无需启动操作系统，这个过程只需要秒级的时间。

5. 不同linux平台下的开发

举个例子比如我要设置mips的编译环境，而我宿主机是debian.有些优秀的工具只存在于gentoo平台，而gentoo又不好安装，我们使用docker只要从docker hub上面pull下镜像，就可以完美使用，这使得开发人员效率最大化