Docker (http://dockone.io/topic/Docker)

可能是把Docker的概念讲的最清楚的一篇文章

【编者的话】本文只是对Docker的概念做了较为详细的介绍，并不涉及一些像Docker环境的安装以及Docker的一些常见操作和命令。

Docker是世界领先的软件容器平台，所以想要搞懂Docker的概念我们必须先从容器开始说起。**如果你想和更多Docker技术专家交流，可以加我微信liyingjiese，备注『加群』。群里每周都有全球各大公司的最佳实践以及行业最新动态。**

先从认识容器开始

什么是容器?

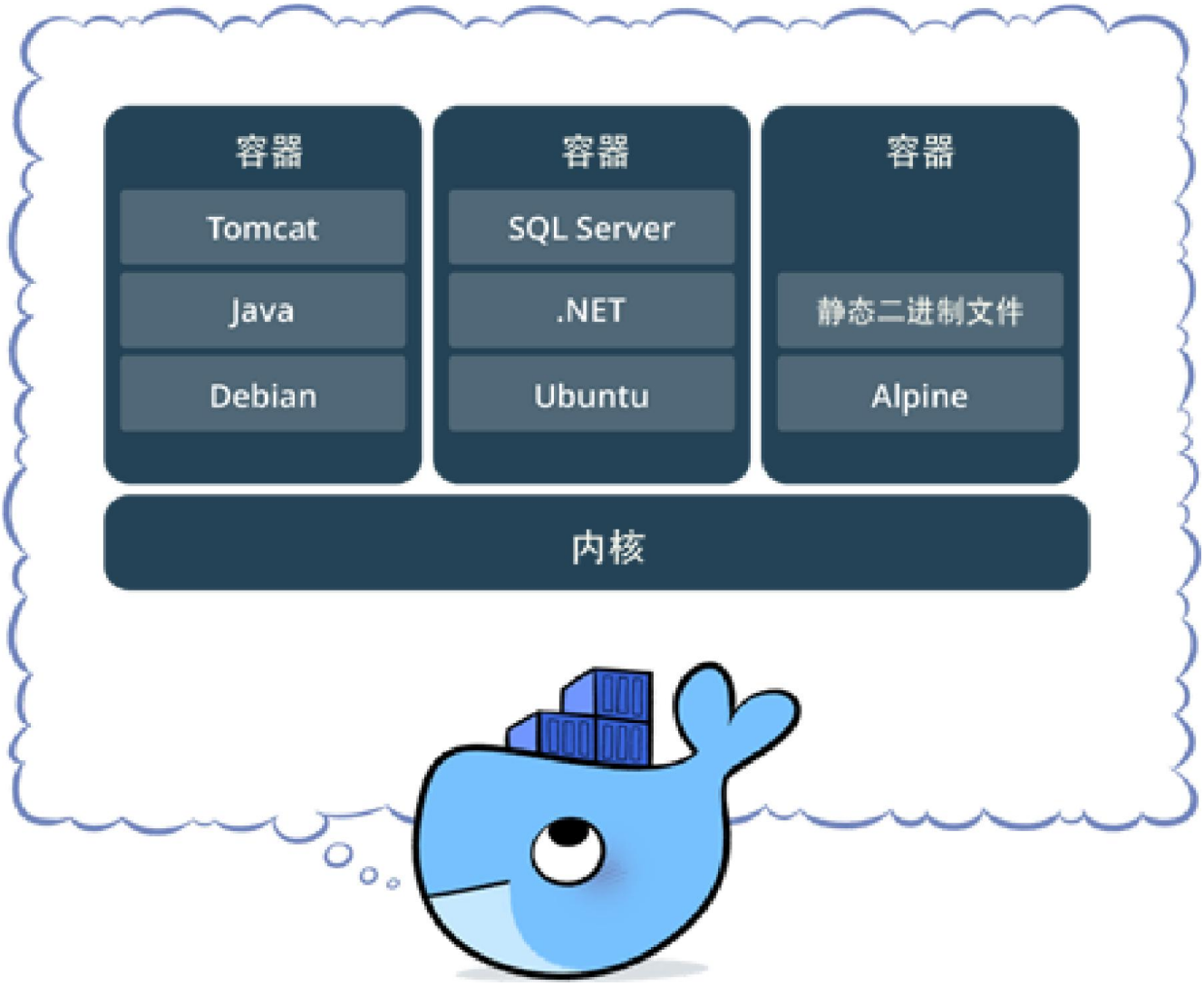
先来看看容器较为官方的解释：

一句话概括容器：容器就是将软件打包成标准化单元，以用于开发、交付和部署。

- 容器镜像是轻量的、可执行的独立软件包 ，包含软件运行所需的所有内容：代码、运行时环境、系统工具、系统库和设置。
- 容器化软件适用于基于Linux和Windows的应用，在任何环境中都能够始终如一地运行。
- 容器赋予了软件独立性，使其免受外在环境差异（例如，开发和预演环境的差异）的影响，从而有助于减少团队间在相同基础设施上运行不同软件时的冲突。

再来看看容器较为通俗的解释：

如果需要通俗的描述容器的话，我觉得容器就是一个存放东西的地方，就像书包可以装各种文具、衣柜可以放各种衣服、鞋架可以放各种鞋子一样。我们现在所说的容器存放的东西可能更偏向于应用比如网站、程序甚至是系统环境。



(http://dockone.io/uploads/article/20190626/0eb95638b712ca6ad211c76f73a366af.png)

图解物理机、虚拟机与容器

关于虚拟机与容器的对比在后面会详细介绍到，这里只是通过网上的图片加深大家对于物理机、虚拟机与容器这三者的理解。

物理机：

(http://dockone.io/uploads/article/20190626/e9444d264eeaea2b178a71c9ed823c9c.jpeg)

虚拟机：

(http://dockone.io/uploads/article/20190626/bb23f04f2b23b2312b04ba87dc8ad31d.jpeg)

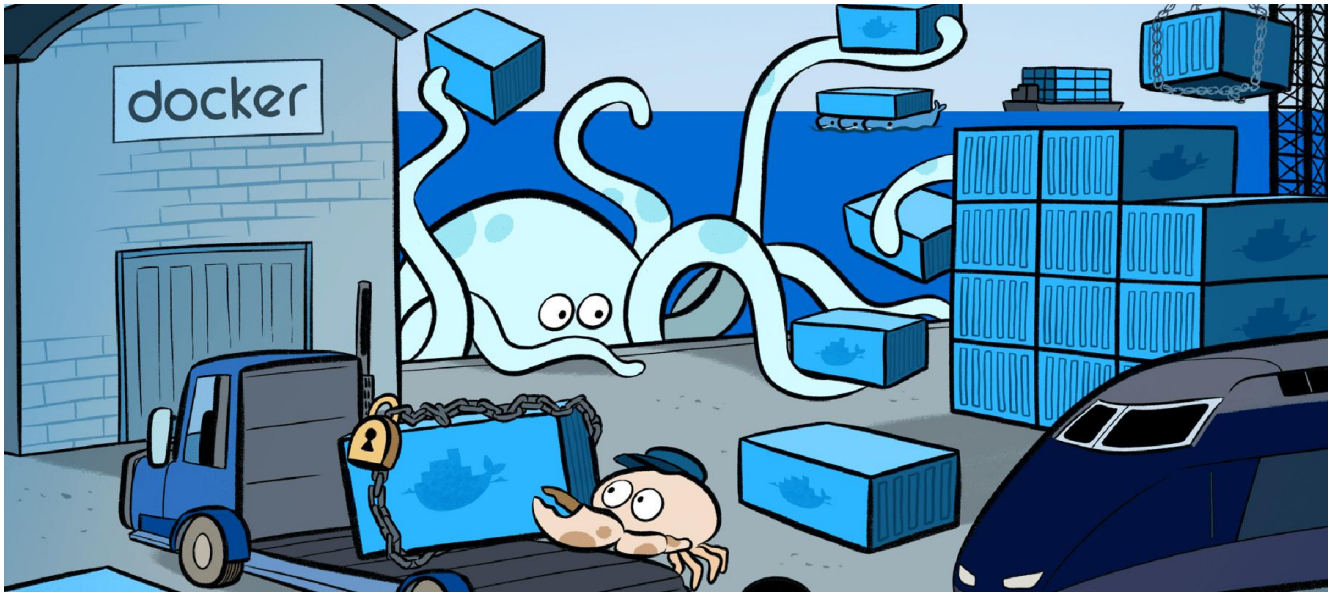
容器：

(http://dockone.io/uploads/article/20190626/e34f0f2a4b8718f09c8b366a14d5e9a9.jpeg)

通过上面这三张抽象图，我们大概可以通过类比概括出： 容器虚拟化的是操作系统而不是硬件，容器之间是共享同一套操作系统资源的。虚拟机技术是虚拟出一套硬件后，在其上运行一个完整操作系统。因此容器的隔离级别会稍低一些。

相信通过上面的解释大家对于容器这个既陌生又熟悉的概念有了一个初步的认识，下面我们就来谈谈Docker的一些概念。

再来谈谈Docker的一些概念

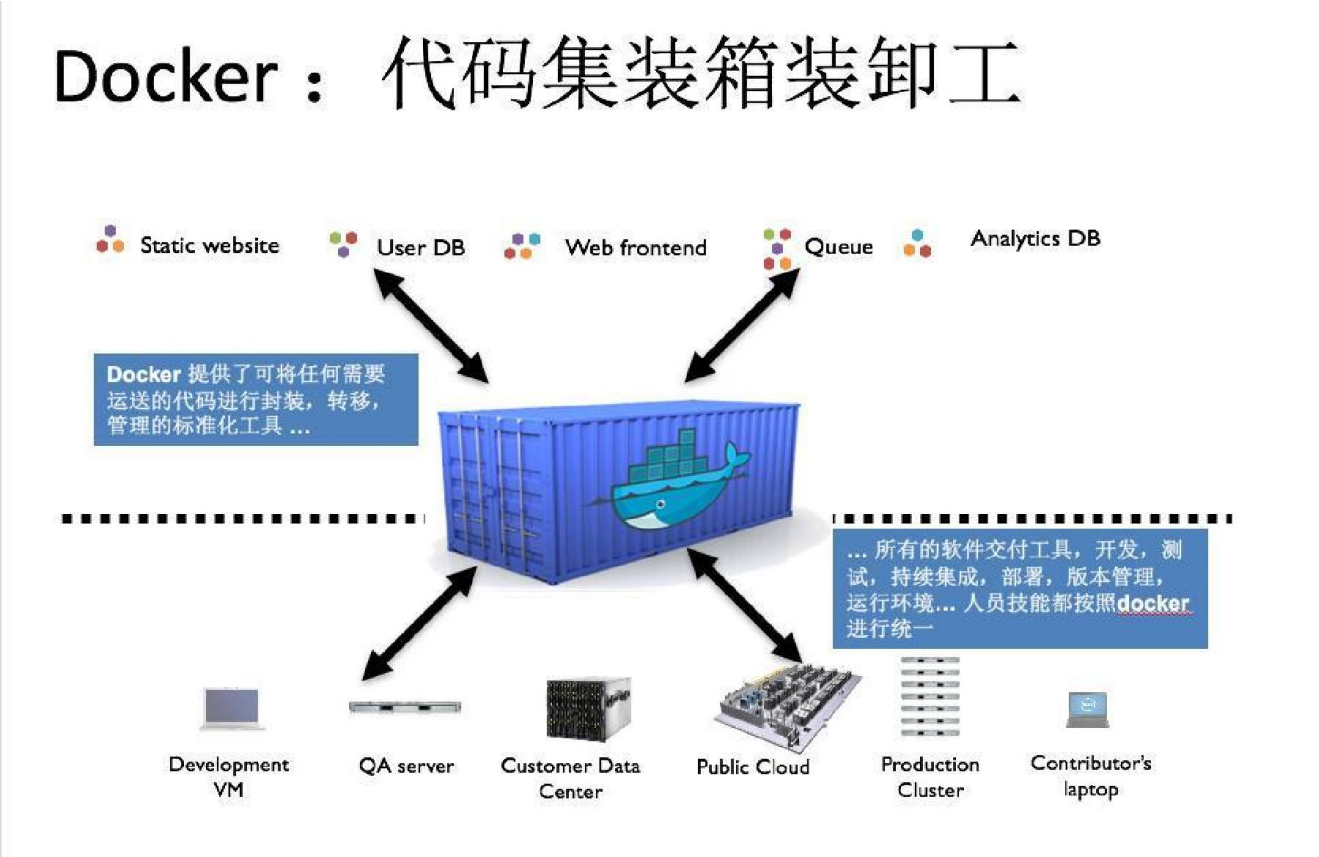


(http://dockone.io/uploads/article/20190626/57be762d7a61f06eedbec4142f3e2291.png)

什么是Docker

说实话关于Docker是什么并太好说，下面我通过四点向你说明Docker到底是个什么东西。

- Docker是世界领先的软件容器平台。
- Docker使用Google公司推出的Go语言进行开发实现，基于Linux内核的cgroup，namespace，以及AUFS类的UnionFS等技术，对进程进行封装隔离，属于操作系统层面的虚拟化技术。 由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为容器。Docke最初实现是基于LXC。
- Docker能够自动执行重复性任务，例如搭建和配置开发环境，从而解放了开发人员以便他们专注在真正重要的事情上：构建杰出的软件。
- 用户可以方便地创建和使用容器，把自己的应用放入容器。容器还可以进行版本管理、复制、分享、修改，就像管理普通的代码一样。



(http://dockone.io/uploads/article/20190626/125334d2b6dc3e34e6943ad5acc5e868.jpeg)

Docker思想

- 集装箱
- 标准化： ①运输方式、②存储方式、 ③API接口
- 隔离

Docker容器的特点

- 轻量，在一台机器上运行的多个Docker容器可以共享这台机器的操作系统内核；它们能够迅速启动，只需占用很少的计算和内存资源。镜像是通过文件系统层进行构造的，并共享一些公共文件。这样就能尽量降低磁盘用量，并能更快地下载镜像。
- 标准，Docker容器基于开放式标准，能够在所有主流Linux版本、Microsoft Windows以及包括VM、裸机服务器和云在内的任何基础设施上运行。
- 安全，Docker赋予应用的隔离性不仅限于彼此隔离，还独立于底层的基础设施。Docker默认提供最强的隔离，因此应用出现问题，也只是单个容器的问题，而不会波及到整台机器。

为什么要用Docker

- Docker的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会再出现“这段代码在我机器上没问题啊”这类问题；——一致的运行环境
- 可以做到秒级、甚至毫秒级的启动时间。大大的节约了开发、测试、部署的时间。——更快速的启动时间
- 避免公用的服务器，资源会容易受到其他用户的影响。——隔离性
- 善于处理集中爆发的服务器使用压力；——弹性伸缩，快速扩展
- 可以很轻易的将在一个平台上运行的应用，迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常运行的情况。——迁移方便
- 使用Docker可以通过定制应用镜像来实现持续集成、持续交付、部署。——持续交付和部署

每当说起容器，我们不得不将其与虚拟机做一个比较。

容器 VS 虚拟机

简单来说： 容器和虚拟机具有相似的资源隔离和分配优势，但功能有所不同，因为容器虚拟化的是操作系统，而不是硬件，因此容器更容易移植，效率也更高。

两者对比图

传统虚拟机技术是虚拟出一套硬件后，在其上运行一个完整操作系统，在该系统上再运行所需应用进程；而容器内的应用进程直接运行于宿主的内核，容器内没有自己的内核，而且也没有进行硬件虚拟。因此容器要比传统虚拟机更为轻便。

(http://dockone.io/uploads/article/20190626/2e2b95eebf60b6d03f6c1476f4d7c697.png)

容器与虚拟机 (VM) 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	一般几十个

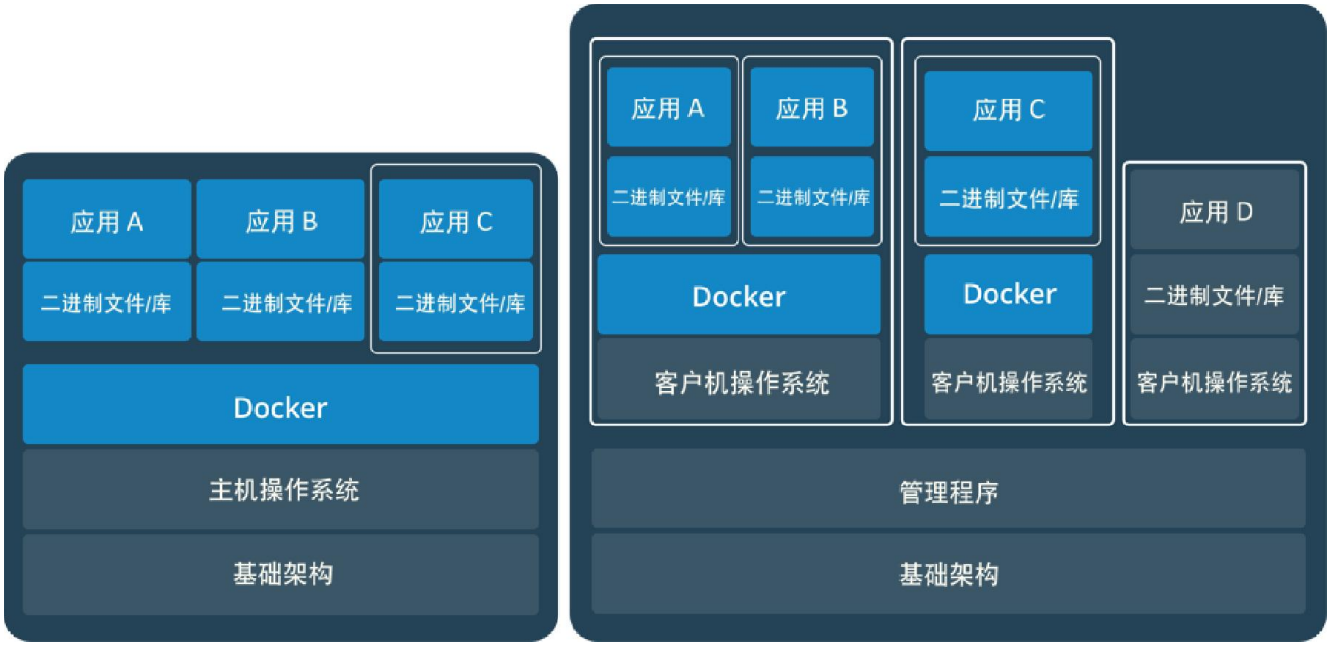
(http://dockone.io/uploads/article/20190626/4ef8691d67eb1eb53217099d0a691eb5.png)

- 容器是一个应用层抽象，用于将代码和依赖资源打包在一起。 多个容器可以在同一台机器上运行，共享操作系统内核，但各自作为独立的进程在用户空间中运行 。与虚拟机相比， 容器占用的空间较少（容器镜像大小通常只有几十兆），瞬间就能完成启动 。
- 虚拟机（VM）是一个物理硬件层抽象，用于将一台服务器变成多台服务器。 管理程序允许多个VM在一台机器上运行。每个VM都包含一整套操作系统、一个或多个应用、必要的二进制文件和库资源，因此占用大量空间。而且VM启动也十分缓慢 。

通过Docker官网，我们知道了这么多Docker的优势，但是大家也没有必要完全否定虚拟机技术，因为两者有不同的使用场景。虚拟机更擅长于彻底隔离整个运行环境。例如，云服务提供商通常采用虚拟机技术隔离不同的用户。而Docker通常用于隔离不同的应用 ，例如前端，后端以及数据库。

容器与虚拟机（VM）两者是可以共存的

就我而言，对于两者无所谓谁会取代谁，而是两者可以和谐共存。



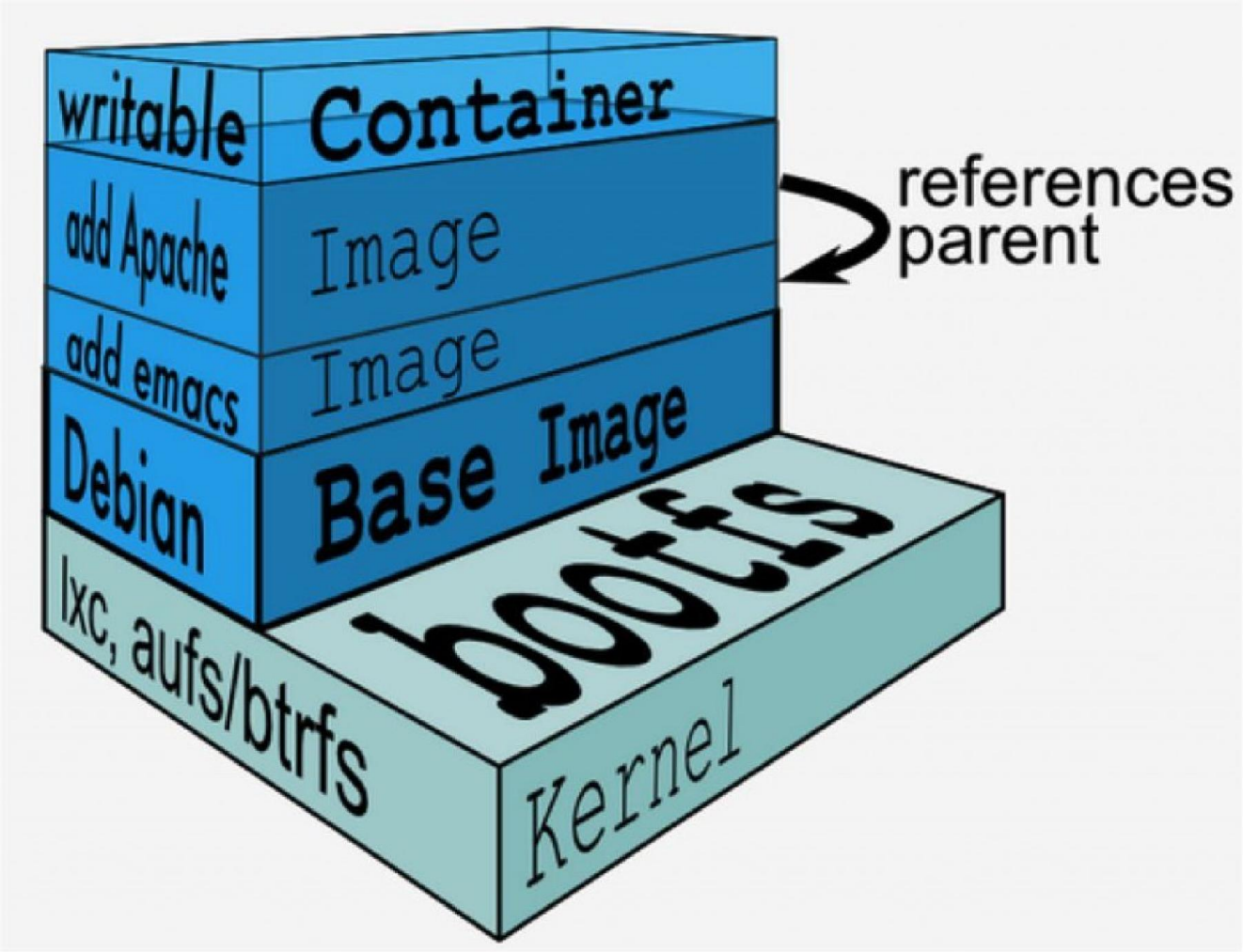
(http://dockone.io/uploads/article/20190626/056c87751b9dd7b56f4264240fe96d00.png)

Docker中非常重要的三个基本概念，理解了这三个概念，就理解了 Docker 的整个生命周期。

Docker基本概念

Docker包括三个基本概念：

- 镜像（Image）
- 容器（Container）
- 仓库（Repository）



(http://dockone.io/uploads/article/20190626/e7c002d3b8de0b7c89f5ff81e6cd9e43.jpeg)

镜像（Image）——一个特殊的文件系统

操作系统分为内核和用户空间。对于Linux而言，内核启动后，会挂载root文件系统为其提供用户空间支持。而Docker镜像（Image），就相当于是一个root文件系统。

Docker镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。 镜像不包含任何动态数据，其内容在构建之后也不会被改变。

Docker设计时，就充分利用Union FS的技术，将其设计为分层存储的架构。 镜像实际是由多层文件系统联合组成。

镜像构建时，会一层层构建，前一层是后一层的基础。每一层构建完就不会再发生改变，后一层上的任何改变只发生在自己这一层。比如，删除前一层文件的操作，实际不是真的删除前一层的文件，而是仅在当前层标记为该文件已删除。在最终容器运行的时候，虽然不会看到这个文件，但是实际上该文件会一直跟随镜像。因此，在构建镜像的时候，需要额外小心，每一层尽量只包含该层需要添加的东西，任何额外的东西应该在该层构建结束前清理掉。

分层存储的特征还使得镜像的复用、定制变的更为容易。甚至可以用之前构建好的镜像作为基础层，然后进一步添加新的层，以定制自己所需的内容，构建新的镜像。

容器（Container）——镜像运行时的实体

镜像（Image）和容器（Container）的关系，就像是面向对象程序设计中的类和实例一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等 。

容器的实质是进程，但与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的命名空间。前面讲过镜像使用的是分层存储，

容器也是如此。

容器存储层的生存周期和容器一样，容器消亡时，容器存储层也随之消亡。因此，任何保存于容器存储层的信息都会随容器删除而丢失。

按照Docker最佳实践的要求，容器不应该向其存储层内写入任何数据 ，容器存储层要保持无状态化。所有的文件写入操作，都应该使用数据卷（Volume）、或者绑定宿主目录，在这些位置的读写会跳过容器存储层，直接对宿主（或网络存储）发生读写，其性能和稳定性更高。数据卷的生存周期独立于容器，容器消亡，数据卷不会消亡。因此， 使用数据卷后，容器可以随意删除、重新run，数据却不会丢失。

仓库（Repository）——集中存放镜像文件的地方

镜像构建完成后，可以很容易的在当前宿主上运行，但是， 如果需要在其它服务器上使用这个镜像，我们就需要一个集中的存储、分发镜像的服务，Docker Registry就是这样的服务。

一个Docker Registry中可以包含多个仓库（Repository）；每个仓库可以包含多个标签（Tag）；每个标签对应一个镜像。所以说：镜像仓库是Docker用来集中存放镜像文件的地方类似于我们之前常用的代码仓库。

通常，一个仓库会包含同一个软件不同版本的镜像，而标签就常用于对应该软件的各个版本 。我们可以通过<仓库名>:<标签>的格式来指定具体是这个软件哪个版本的镜像。如果不给出标签，将以latest作为默认标签。

这里补充一下Docker Registry公开服务和私有Docker Registry的概念：

Docker Registry公开服务是开放给用户使用、允许用户管理镜像的Registry服务。一般这类公开服务允许用户免费上传、下载公开的镜像，并可能提供收费服务供用户管理私有镜像。

最常使用的Registry公开服务是官方的Docker Hub ，这也是默认的Registry，并拥有大量的高质量的官方镜像，网址为：hub.docker.com/ 。在国内访问Docker Hub可能会比较慢国内也有一些云服务商提供类似于Docker Hub的公开服务。

除了使用公开服务外，用户还可以在本地搭建私有Docker Registry 。Docker官方提供了Docker Registry镜像，可以直接使用做为私有Registry服务。开源的Docker Registry镜像只提供了Docker Registry API的服务端实现，足以支持Docker命令，不影响使用。但不包含图形界面，以及镜像维护、用户管理、访问控制等高级功能。

最后谈谈：Build，Ship，and Run

如果你搜索Docker官网，会发现如下的字样：“Docker - Build, Ship, and Run Any App, Anywhere”。那么Build，Ship，and Run到底是在干什么呢?

(http://dockone.io/uploads/article/20190626/653dbc01d29a85f51cd32b045507d27c.png)

- Build（构建镜像）：镜像就像是集装箱包括文件以及运行环境等等资源。
- Ship（运输镜像）：主机和仓库间运输，这里的仓库就像是超级码头一样。
- Run （运行镜像）：运行的镜像就是一个容器，容器就是运行程序的地方。

Docker运行过程也就是去仓库把镜像拉到本地，然后用一条命令把镜像运行起来变成容器。所以，我们也常常将Docker称为码头工人或码头装卸工，这和Docker的中文翻译搬运工人如出一辙。

总结

本文主要把Docker中的一些常见概念做了详细的阐述，但是并不涉及Docker的安装、镜像的使用、容器的操作等内容。这部分东西，希望读者自己可以通过阅读书籍与官方文档的形式掌握。

原文链接：https://mp.weixin.qq.com/s/xSbYTJmLuqsyYEDEIsndZw (https://mp.weixin.qq.com/s/xSbYTJmLuqsyYEDEIsndZw)

DockOne，新圈子，新思路，新视野