**操作系统容器**

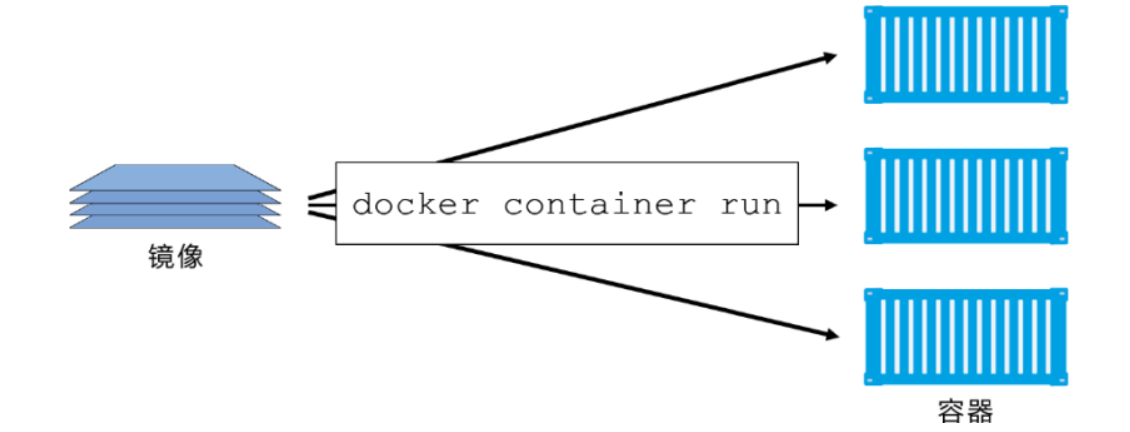
1190201215

冯开来

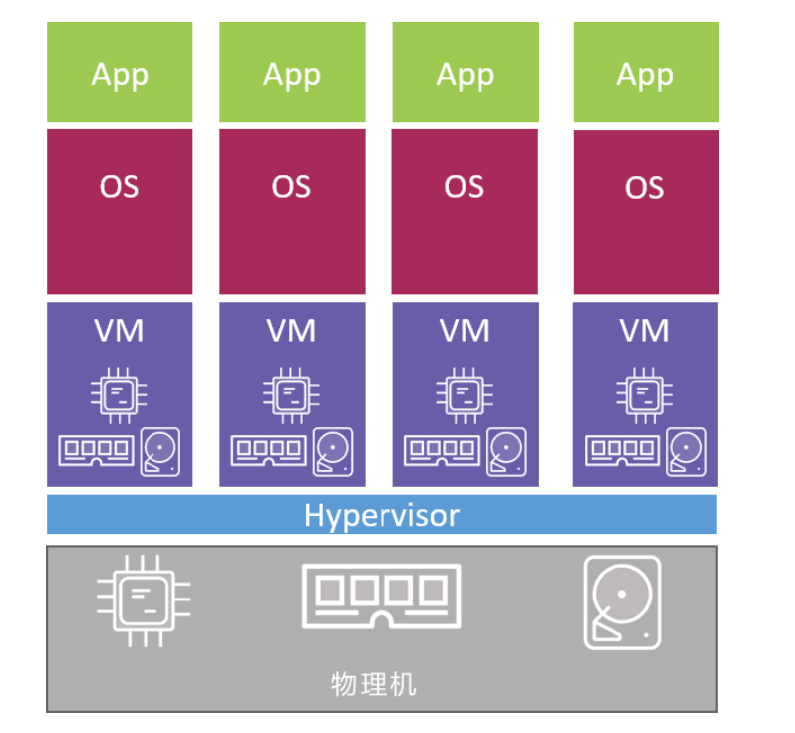
首先，我们先从两个角度先认识一下容器的概念。专业地讲，容器是实现操作系统虚拟化的一种途径，可以让用户在资源受到隔离的进程中运行应用程序及其依赖关系；简单地讲，容器就是基于linux内核技术的一种隔离手段。很多人认为容器就是另外一种虚拟机（VM）。实际上，虚拟机通过Guest OS来运行对应的应用的，而容器通过使用Docker Engine使用系统原生隔离技术进行隔离。

从容器发展的历史来看，可以说PaaS（Platform-as-a-Service，平台即服务）的发展是带动了操作系统容器即Docker技术的发展。PaaS把服务器平台作为一种提供服务的商业模式。之前的PaaS平台是使用VM的方式去部署平台应用服务的，但是由于VM启动的特点，部署一个应用时间十分长。前期，这样的部署还可以适应企业的需要，但随着互联网发展，企业越来越需要适应迅速多变的业务场景，部署在主机上的的环境不可控制及启动时间问题成为了PaaS平台发展的阻力；后期，微服务架构及持续集成部署CI/CD大行其道，PaaS部署应用必须要有新的解决方案。容器为PaaS平台带来了以下特点：首先是环境一致性。通过容器及其独特文件系统，使得应用程序在开发、测试、生产环节中高度一致，实现跨多环境无差别移动。同时解放开发人员生产力。使用容器主要的目的是采用微服务架构改变应用，开发人员把主要精力集中在当前业务服务的开发，不必再担心语言依赖库问题。第三是运营高效。由于容器基于原生linux内核的虚拟隔离技术，容器运行与原生进程消耗资源基本一致。容器可以让用户在一个系统中轻松运行多个容器镜像，这与在一个系统中运行多个进程一样。用户还可以创建容器映像，将其作为其他映像的基础，使大应用迁移部署达到秒级。最后是便于版本控制。运营团队可以创建基本映像，将操作系统、配置和所需的各种工具实用程序囊括在其中。开发团队可以在基础镜像的基础上构建其不同版本的服务镜像，这样有利于从开发测试到生产使用统一配置环境。

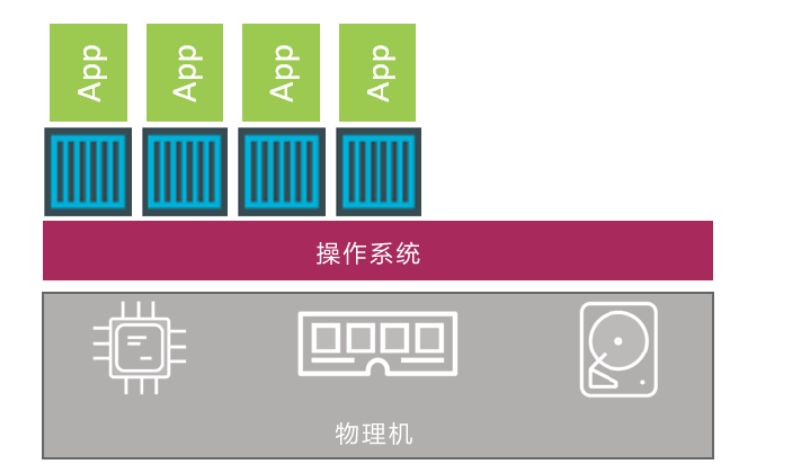
容器是镜像的运行时实例。正如从虚拟机模板上启动VM一样，用户也同样可以从单个镜像上启动一个或多个容器。虚拟机和容器最大的区别是容器更快并且更轻量级——与虚拟机运行在完整的操作系统之上相比，容器会共享其所在主机的操作系统/内核。下图为使用单个 Docker 镜像启动多个容器的示意图。



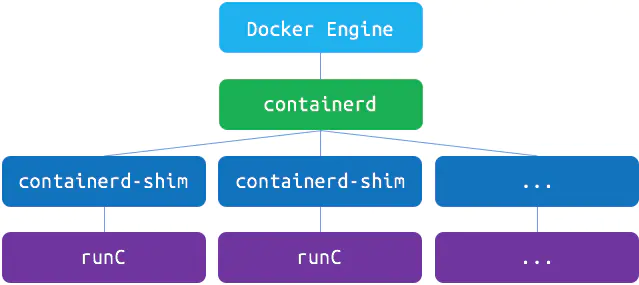
容器和虚拟机都依赖于宿主机才能运行。宿主机可以是笔记本，是数据中心的物理服务器，也可以是公有云的某个实例。我们假设宿主机是一台需要运行4个业务应用的物理服务器。在虚拟机模型中，首先要开启物理机并启动Hypervisor引导程序。一旦Hypervisor启动，就会占有机器上的全部物理资源，如CPU、RAM、存储和NIC。Hypervisor接下来就会将这些物理资源划分为虚拟资源，并且看起来与真实物理资源完全一致。然后Hypervisor会将这些资源打包进一个叫作虚拟机（VM）的软件结构当中。这样用户就可以使用这些虚拟机，并在其中安装操作系统和应用。前面提到需要在物理机上运行4个应用，所以在Hypervisor之上需要创建4个虚拟机并安装4个操作系统，然后安装4个应用。当操作完成后，结构如下图所示。



而容器模型则不同。服务器启动之后，所选择的操作系统会启动。在Docker世界中可以选择Linux，或者内核支持内核中的容器原语的新版本Windows。与虚拟机模型相同，OS也占用了全部硬件资源。在OS层之上，需要安装容器引擎。容器引擎可以获取系统资源，比如进程树、文件系统以及网络栈，接着将资源分割为安全的互相隔离的资源结构，称之为容器。每个容器看起来就像一个真实的操作系统，在其内部可以运行应用。按照前面的假设，需要在物理机上运行4个应用。因此，需要划分出4个容器并在每个容器中运行一个应用，如下图所示。



接下来我们来认识一下Docker架构。Docker在早期所有功能都是做在docker engine里面的，但是之后功能越来越多，越来越重，很多功能已经和基础的容器运行时没有关系了（比如swarm项目）。所以为了响应社区的呼声，也为了兼容oci标准，Docker也做了架构调整。将容器运行时相关的程序从docker daemon剥离出来，形成了containerd。Containerd向Docker提供运行容器的API，二者通过grpc进行交互。containerd最后会通过runc来实际运行容器。调整完后的Docker架构图如下。



接着我们详细谈一谈容器技术带来的优点。容器技术虚拟化技术极大提升了效率。虚拟化技术已经成为一种被大家广泛认可的服务器资源共享方式，它可以在按需构建操作系统实例的过程当中为系统管理员提供极大的灵活性。

起初，大家普遍认为基于hypervisor的方式可以在最大程度上提供灵活性。所有虚拟机实例都能够运行任何其所支持的操作系统，而不受其他实例的影响。然而，越来越多的用户发现hypervisor提供这样一种广泛支持的特性其实是在给自己制造麻烦。对于hypervisor环境来说，每个虚拟机实例都需要运行客户端操作系统的完整副本以及其中包含的大量应用程序。从实际运行的角度来说，由此产生的沉重负载将会影响其工作效率及性能表现。

首先，每种操作系统和应用程序堆栈都需要使用DRAM。对于多个运行简单应用程序的小型虚拟机实例来说，这种方式可能产生很大的系统开销，降低性能表现。加载并卸载这些堆栈镜像需要花费很长时间，并且还会增加容器技术服务器的网络连接数量。部署虚拟服务器的目的之一在于快速创建新的虚拟机实例。然而从网络存储当中复制镜像需要花费大量时间，这些操作会延长启动过程，无疑会限制系统灵活性。

因此我们可以选择使用容器技术。只需要通过简单的观察我们便能够发现容器技术的出现是为了解决多操作系统/应用程序堆栈的问题：在单台服务器当中为所有虚拟机实例使用相同的操作系统对于大部分数据中心来说都不算是真正的限制。流程管理（Orchestration）可以轻松处理这种变化；许多应用程序堆栈都是相同的（如LAMP）；对于大规模集群来说，在本地硬盘当中存储操作系统副本将会使得更新过程变得更为复杂。最为重要的是，容器技术可以同时将操作系统镜像和应用程序加载到内存当中。还可以从网络磁盘进行加载，因为同时启动几十台镜像不会对网络和存储带来很大负载。之后的镜像创建过程只需要指向通用镜像，大大减少了所需内存。

了解了容器的概念以及简短的历史，然后我们比较了容器与虚拟机之间的区别，同时还学习了Docker的架构，同时更加详细的了解了容器的优势。在这里，我们简短对容器的原理进行简短的介绍。Docker的核心技术包含以下几点：Linux namespace、Linux cgroup、rootfs、镜像分层。

容器里的应用，是linux系统里的一个特殊进程，docker通过linux namespace技术对应用进程进行了隔离，使的应用只能看到指定的有限的系统信息，这就使应用“以为”自己在一个独立的操作系统环境下。Linux namespace，跟K8S、C++的namespace的功能是类似的，目的都是将一组资源限定在一个有限的可见范围内。Linux的Cgroup机制，是Docker利用的又一大利器。由于容器其实就是宿主机里的一个被框进来的进程，它不能看到外面，但它与宿主机上其他的进程共享了内核资源，所以接下来我们需要对它所能使用的资源作限制，这就是Cgroup机制所提供的。Cgroup的使用，比较简单，它利用一组目录和文件的组合，来实现配置和控制。一致性主要是为了解决应用跑在不同的宿主机上不受宿主机环境的差异影响的问题。容器技术出来之前，手动或脚本迁移应用的时候，往往会遇到新的宿主机缺少某个关键组件、或是某些依赖版本差异甚至是操作系统内核差异等因素导致的不一致问题。对于单个应用程序进程来说，对环境的依赖，关键体现在对操作系统所提供的文件系统的依赖，所以docker所要做的就是通过以下3步给你一套想要的文件系统：通过mount namespace将应用的文件系统隔离开；将应用所需要的文件系统拷贝到某个目录下；调用chroot将应用的根目录调整为该目录（mount namespace的隔离作用在这里体现出来了，chroot只对在当前namespace下的应用生效，应用在这个“根目录”下可以随便折腾，而不会影响到真实的宿主机根目录）。上面这3步所构造出来的文件系统，我们称之为rootfs（根文件系统）。不同的容器镜像共用相同的层，docker在这里运用了copy-on-write的手法。基于同样的底层镜像创建的新镜像，可以共用相同的镜像层，所以无论是下载还是硬盘存储，都不会太大。

最后我们对容器的未来发展做一些简单的介绍。首先互联网客户对容器的使用已经很广泛，而传统的大中型企业在加快容器的使用，这也促进Docker等产品的进一步成熟。其次是随着Kubernetes和Docker Swarm的进步，以及各大公有云公司提供对这两者的支持，开发和定制Kubernetes分发版的提供商要找到真正的增值。最后是简单易用的公有容器服务会变得流行。