复用云技术

刘旭东

容器技术

> Containerization is a system of intermodal freight transport using intermodal containers made of weathering steel. The containers have standardized dimensions.
>
> 容器技术起源于码头运输采用的标准化集装箱技术。容器只对操作系统内核进行抽象处理，为应用程序提供了隔离的运行空间：每个容器内都包含一个独享的完整用户环境空间，并且一个容器内的变动不会影响其他容器的运行环境。
>
> 为了能达到这种效果，容器技术使用了一系列的系统级别的机制诸如利用Linux namespaces来进行空间隔离，通过文件系统的挂载点来决定容器可以访问哪些文件，通过cgroups来确定每个容器可以利用多少资源。此外

> 容器之间共享同一个系统内核，这样当同一个库被多个容器使用时，内存的使用效率会得到提升。

关键技术

- 资源独立、隔离

  资源隔离是云计算平台的最基本需求。Docker通过linux namespace, cgroup限制了硬件资源与软件运行环境，与宿主机上的其他应用实现了隔离，做到了互不影响。不同应用或服务以“集装箱”（container）为单位装“船”或卸“船”，“集装箱船”（运行container的宿主机或集群 ）上，数千数万个“集装箱”排列整齐，不同公司、不同种类的“货物”（运行应用所需的程序，组件，运行环境，依赖）保持独立。

- 安全性

  容器用户没有摆脱安全问题。例如，当部署Docker容器技术时，IT专业人员面临一系列的安全方面的担忧，根据这条来自David Linthicum的技术提示描述。例如，容器非常灵活，这使企业很容易就能运行多个容器实例。然而，这意味着不同的容器可能处在不同的安全补丁级别。 IT专业人员应该使用诸如Docker内容信任 (DCT) 这样的技术来确保容器没有被攻击。DCT使用密钥为Docker镜像增加了额外的安全层，并能显示是否有其他人已经篡改了该镜像。 此外，Docker的安全基准给出了84条Docker容器部署的最佳做法。大多数容器的安全问题都是来自于糟糕的设计，而阅读这些最佳做法可以帮助你确定哪里是最脆弱的环节。

- 环境的一致性

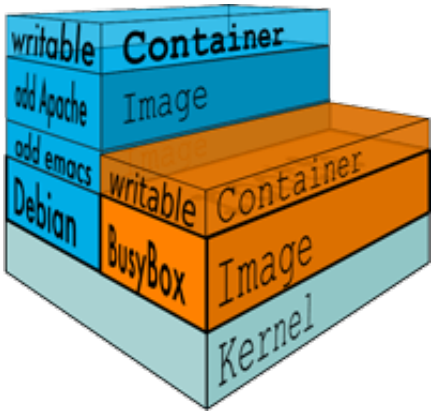
  开发工程师完成应用开发后build一个docker image，基于这个image创建的container像是一个集装箱，里面打包了各种“散件货物”（运行应用所需的程序，组件，运行环境，依赖）。无论这个集装箱在哪里：开发环境、测试环境、生产环境，都可以确保集装箱里面的“货物”种类与个数完全相同，软件包不会在测试环境缺失，环境变量不会在生产环境忘记配置，开发环境与生产环境不会因为安装了不同版本的依赖导致应用运行异常。这样的一致性得益于“发货”（build docker image）时已经密封到”集装箱“中，而每一个环节都是在运输这个完整的、不需要拆分合并的”集装箱“。

- 轻量化

相比传统的虚拟化技术（VM），使用docker在cpu, memory, disk IO, network IO上的性能损耗都有同样水平甚至更优的表现。Container的快速创建、启动、销毁受到很多赞誉。

- Build Once, Run Everywhere

  “货物”（应用）在“汽车”，“火车”，“轮船”（私有云、公有云等服务）之间迁移交换时，只需要迁移符合标准规格和装卸方式的“集装箱”（docker container），削减了耗时费力的人工“装卸”（上线、下线应用），带来的是巨大的时间人力成本节约。这使未来仅有少数几个运维人员运维超大规模装载线上应用的容器集群成本可能，如同60年代后少数几个机器操作员即可在几小时内连装带卸完一艘万级集装箱船。



容器技术的优势

应用开发人员

- 版本质量更高
- 应用可扩展性更强
- 应用隔离效果更好

IT 架构师

- 横向扩展速度更快
- 测试周期更短
- 部署错误更少

IT 运营人员

- 版本质量更高
- 更高效地替换生产环境中的全部虚拟机
- 应用管理更便捷

容器技术的限制

- 不能应用在所有场景当中

  虽然容器技术拥有很强的兼容性，但是仍然不能完全取代现有的虚拟机环境。就像虚拟化技术刚刚出现的时候，一些传统的应用程序更加适合运行在物理环境当中一样，现在，一些应用程序并不适合运行在容器虚拟化环境当中。 比如，容器技术非常适合用于开发微服务类型的应用程序——这种方式将复杂的应用程序拆分为基本的组成单元，每个组成单元部署在独立的容器当中，之后将相关容器链接在一起，形成统一的应用程序。可以通过增加新的组成单元容器的方式对应用程序进行扩展，而不再需要对整个应用程序进行重新开发。 但是另一方面，一些应用程序只能以统一整体的形式存在——它们在最初设计时就采用了这种方式，很难实现高扩展性和快速部署等特性。对于这种情况来说，容器技术反而会对应用负载造成限制。最好的检验方式就是进行大量试验，查看哪种现有应用程序能够通过容器技术发挥最大优势。一般来说，新的应用程序研发过程很可能从容器技术当中获益。而那些不能被容器化的应用程序仍然可以运行在传统hypervisor的全功能虚拟机当中。一位来自知名保险提供商的IT架构师表示应该放缓应用程序容器化趋势。“虽然容器技术非常具有吸引力，但是软件开发团队需要一段时间及时跟进，才能够真正地高效利用容器技术所带来的优势。”

- 难以解决依赖关系问题

  大多数虚拟机都是相对独立的，每台虚拟机都包含自己的操作系统、驱动和应用程序组件。只要拥有合适的hypervisor，还可以将虚拟机迁移到其他任何虚拟化平台当中。但是对比来说，容器运行在物理操作系统之上，相互之间共享大量底层的操作系统内核、库文件以及二进制文件。容器之间的现有依赖关系可能会限其在服务器之间的可移植性。比如，位于Linux操作系统上的Docker容器就不能运行在当前版本的Windows Server操作系统上。 对于这种问题来说，当前的解决方案并不止一种——容器可以在数秒钟之内完成复制过程，操作系统也在不断发展，开始提供“micro OS”和“nano OS”等多种类型，提供了高稳定性以及快速重启等特性。从容器自身的角度来说其更加适合于这些环境，只要数据中心当中的其他服务

器可用，仍然能够对其进行迁移。 随着操作系统的逐渐发展，这些依赖关系问题也在不断得到解决。比如，Windows Server 2016承诺同时支持Docker和原生Hyper-V容器。除了Docker之外，还有许多其他容器平台可供选择，比如LXC、Parallels Virtuozzo、Joyent、Canonical LXD、Spoon等等，VMware也有可能随时加入到竞争行列中来。

- 较差的隔离性

  基于hypervisor的虚拟机拥有完善的隔离特性，由于系统硬件资源完全是虚拟的，由hypervisor分配给虚拟机使用，因此bug、病毒或者入侵有可能影响一台虚拟机，但是不会蔓延到其他虚拟机上。 容器的隔离性较差因为其共享同一个操作系统内核以及其他组件，在开始运行之前就已经获得了统一的底层授权（对于Linux环境来说通常是root权限）。因此，漏洞和攻击更加有可能进入到底层的操作系统，或者转移到其他容器当中——潜在的传播行为远比最初的事件更加严重。 尽管容器平台也在不断发展，开始隔离操作系统权限、减少脆弱的安全特性等，但是仍然推荐管理员通过在虚拟机当中运行容器来提升安全性。比如，可以在Hyper-V当中部署一台Linux虚拟机，在Linux虚拟机当中安装Docker容器。这样即便虚拟机当中的容器出现问题，这种漏洞也只存在于当前虚拟机当中——限制了潜在的受攻击范围。

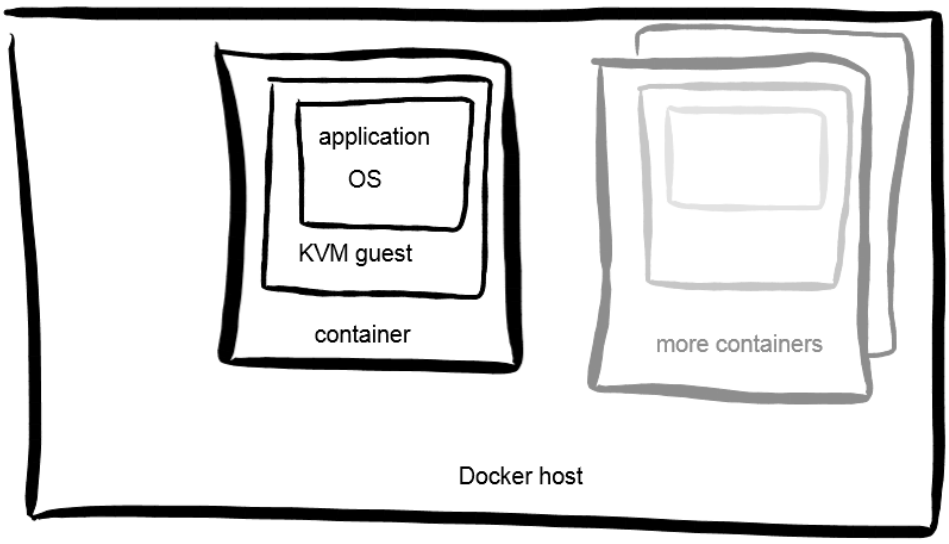
- 潜在的蔓延问题

  就像虚拟机生命周期管理对于hypervisor环境来说十分重要一样，生命周期管理对于容器来说也是至关重要的。容器可以被大量快速复制，这是容器技术的重要优势之一，但是也有可能在管理员没有注意到的情况下消耗大量计算资源。如果应用程序所在的容器不再使用时能够被及时删除，那么情况还不算太坏。但是如果对一个容器化应用程序进行扩展之后忘记将其缩减回之前的规模，那么将会为企业带来大量的（并且不必要的）云计算开销。云提供商十分高兴看到这种情况发生——因为他们就是通过出租计算资源而获利的——因此用户需要自己关注容器的部署情况。

- 缺乏工具

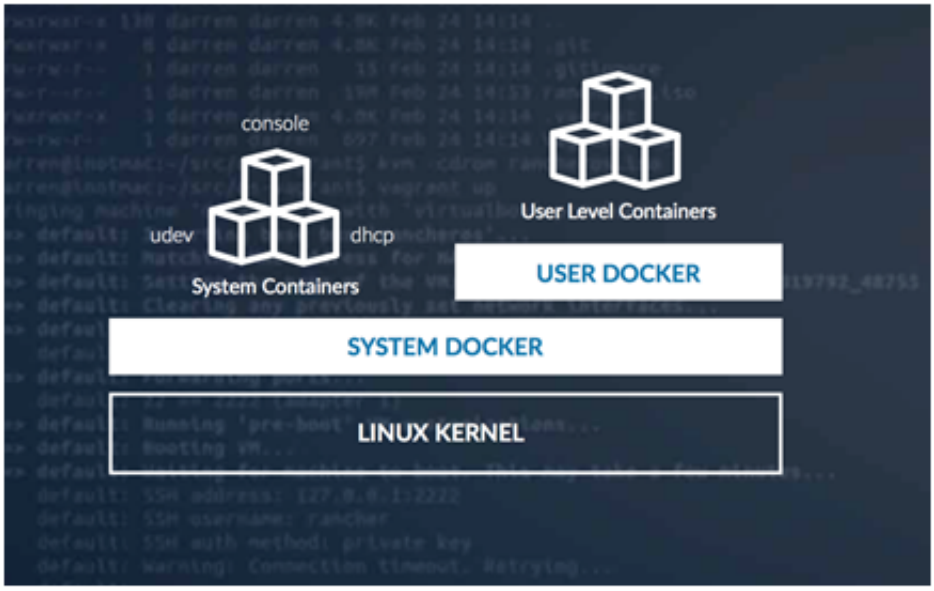
  对于这个行业来说，用于监控和管理容器的工具种类仍然十分缺乏。这并不是一种最近产生的现象，在基于hypervisor虚拟化的早期也曾经出现过可用工具十分匮乏的情况。就像优秀的虚拟机监控和管理工具逐渐增多一样，容器管理领域也在不断出现新的工具。其中包括谷歌的开源Docker管理工具Kubernetes，此外DockerUI使用基于web的前端界面替换Linux的命

令行功能，Logspout能够将容器日志汇集到一个集中位置。 建议管理员可以通过将容器运行在虚拟机当中缓解容器管理工具缺乏的问题，这样就可以使用虚拟机工具来完成一些监控和管理功能了。因为虚拟机工具更加成熟和多样化，因此在容器工具逐渐成熟之前，可以将其作为临时的替代产品。容器自身（非虚拟化I/O）还能够实现更好的性能表现；已经拥有像Docker这样优秀的开发架构，像GitHub这样吸引广泛关注的共享和协作平台。但是容器并不是一种能够满足所有虚拟化任务的解决方案，只是虚拟化工具箱提供的另外一种工具——通常可以和传统虚拟机很好地协同工作。



行业流行解决方案

- Docker
- LXC
- Parallels Virtuozzo
- Canonical LXD



参考（Reference)

- 为什么容器技术将主宰世界
- 虚拟化 VS 容器化
- 操作系统层虚拟化 wikipedia
- What is docker
- 容器技术的五大缺陷