复用云技术——容器技术

1352873　王刚

关于容器技术的理解：

首先，我想介绍一下容器技术的相关基本概念。

首先，宏观来说，我认为容器有点像虚拟机，因为它们都是应用程序运行时的环境，我们在其内部运行应用程序。但是两者比较起来，容器要比虚拟机轻便得多。具体来说，容器为应用程序提供了隔离的运行空间，即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。也就是说，容器是一个可管理的执行环境，与主机系统共享内核，可与系统中的其他容器进行隔离。

顾名思义，我们不难想到“容器技术”可能和容器之间存在某种关系。其实，在别人的研究中，我们不难看出码头上用以装载货物的集装箱确实和容器技术的类似之处，下面是一个关于两者的类别[这里用近两年来容器技术的代表 Docker 代表容器技术进行比较]：

集装箱	类比	容器技术(Docker)
发货商	<=>	应用的发布者，现实中多为应用的生产方，即开发者
客户	<=>	使用应用的互联网用户
货物	<=>	构成应用的代码、组件、依赖等
集装箱	<=>	Docker容器
装卸货	<=>	应用的发布、撤销
码头工人	<=>	实际操作应用发布过程的人，现实中多为运维人员
散件装卸、运输方式	<=>	应用发布过程中逐个安装部署代码、组件、依赖、配置环境等
集装箱装卸、运输方式	<=>	把应用运行所需的外部环境、内部代码、组件、依赖打包放进容器
港口的码头、起重机、集装箱堆场	<=>	应用发布所需的基础设施与工具
轮船/轮船公司	<=>	容器运行平台，如可以运行容器的云计算平台

关键技术：

资源隔离、独立：

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：

单就 Docker 来说，安全性可以概括为两点：

不会对主机造成影响
不会对其他容器造成影响

所以安全性问题 90%以上可以归结为隔离性问题。

在接纳了“容器并不是全封闭”这种思想以后，开源社区尤其是红帽公司，连同 Docker 一起改进 Docker 的安全性，改进项主要包括保护宿主不受容器内部运行进程的入侵、防止容器之间相互破坏。开源社区在解决 Docker 安全性问题上的努力包括：

1. Audit namespace
作用：隔离审计功能
未合入原因：意义不大，而且会增加 audit 的复杂度，难以维护。
2. Syslognamespace
作用：隔离系统日志
未合入原因：很难完美的区分哪些 log 应该属于某个 container。
3. Device namespace
作用：隔离设备（支持设备同时在多个容器中使用）
未合入原因：几乎要修改所有驱动，改动太大。
4. Time namespace
作用：使每个容器有自己的系统时间
未合入原因：一些设计细节上未达成一致，而且感觉应用场景不多。
5. Task count cgroup
作用：限制 cgroup 中的进程数，可以解决 fork bomb 的问题
未合入原因：不太必要，增加了复杂性，kmemlimit 可以实现类似的效果。(最近可能会被合入)
6. 隔离/proc/meminfo 的信息显示
作用：在容器中看到属于自己的 meminfo 信息

还有一些安全技术有：
1、文件系统级防护

2、Capability 机制
3、NameSpace 机制
4、Cgroups 机制
5、SELinux

除了上面提到的两点关于容器技术的诱人之处外，还有下面几点：

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 ：

这个特性着实吸引了我，“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。

另外还有以下优势（按角色划分）：

对于应用开发人员：
1. 版本质量更高
2. 应用可扩展性更强
3. 应用隔离效果更好

对于 IT 架构师：
1. 横向扩展速度更快
2. 测试周期更短

3.部署错误更少

对于 IT 运营人员：

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

挑战：

管理：

Libcontainer 是 Docker 中用于容器管理的包，它基于 Go 语言实现，通过管理 namespaces、cgroups、capabilities 以及文件系统来进行容器控制。你可以使用 Libcontainer 创建容器，并对容器进行生命周期管理。
其中包括资源管理，在《Docker 背后的内核知识：cgroups 资源隔离》一文中已经提到，Docker 使用 cgroups 进行资源管理与限制，包括设备、内存、CPU、输入输出等。目前除网络外所有内核支持的子系统都被加入到 Libcontainer 的管理中，所以 Libcontainer 使用 cgroups 原生支持的统计信息作为资源管理的监控展示。

管控能力的缺失：

在分布式环境下，对动态容器进行应用管理、监控、日志收集对依然存在挑战；目前 Docker 原生方案，流行的开源方案还有很多初创公司都在提供自己的解决方案试图解决这些问题。在应用架构层面，微服务架构一方面很好的解决了移动互联网时代对扩展性和敏捷性需求;但另一方面由于分布式系统和弹性部署的引入增加了运维的复杂性，同时可能造成技术的碎片化；这就需要容器平台能够更加自动化的进行运维，为用户提供以应用中心的管控视图。

如何复用：

我们需要解决和希望解决的问题：

Gartner 副总裁兼著名分析师 Thomas Bittman 指出了容器技术存在的很多缺陷。这些是我们需要解决和希望解决的问题。下面我们将逐个分析这些缺陷并且讨论如何进行解决。

1、不能应用在所有场景当中：

Bittman 认为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。”

2、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。Bittman 进一步解释说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。

3、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 Bittman 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。

4、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。Bittman 还表示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。

5、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。

Bittman 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。

Bittman 对于容器技术充满热情，认为其能够快速交付轻量级的应用程序，提升资源使用效率和扩展性；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。

选取容器技术方案：

根据之前对于容器技术的研究和探索，我觉得选取容器技术方案时主要应考虑以下几点：

要选择能与目标系统切合兼容的可复用容器；

要保证复用的容器能够保障一个相对隔离独立的环境；

选择性能尽量好的、易于管理的成熟的可复用容器。

参考资料：

《浅谈 Docker 隔离性和安全性》
http://www.freebuf.com/articles/system/69809.html

《为什么容器技术将主宰世界》
http://blog.csdn.net/gaoyingju/article/details/49616295

《虚拟化 VS 容器化》
http://dockone.io/article/326

《关于红帽》
https://www.redhat.com/zh/insights/containers

《Docker 背后的容器管理——Libcontainer 深度解析》
http://www.infoq.com/cn/articles/docker-container-management-libcontainer-depth-analysis

《【深度】阿里 Docker 服务开发中的 5 大挑战与经验沉淀》
https://yq.aliyun.com/articles/4124