讨论课3

> 2016.6.6
>
> Created by Fowafolo

讨论内容

> 侧重于容器技术,讨论各种方案以及挑战
>
> 1. 容器技术的理解
>
>   关键技术
>
>   - 隔离
>   - 安全
>
>   挑战
>
>   - 管理(Orchestrating at Scale)
> 2. 如何复用
>
>   我们需要和希望解决的问题
>
>   选取什么样的容器技术方案

根据要求我进行了如下的学习~

容器技术

首先，我查阅了 `容器技术` 这一概念。

定义

`容器技术虚拟化技术` 已经成为一种被大家广泛认可的容器技术服务器资源共享方式，容器技术可以在按需构建容器技术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。由于hypervisor虚拟化技术仍然存在一些性能和资源使用效率方面的问题，因此出现了一种称为容器技术（Container）的新型虚拟化技术来帮助解决这些问题。

从2014年或更早，就有专家预见到Docker/容器技术会是云计算的颠覆力量（disruptive force）。坦率地讲，

云计算作为一种服务和应用的业务模式，很难讲会被颠覆，但容器技术的确是云计算的game changer，它改变了我们思考云计算的视角，是云计算的reinventor。

安全性、资源隔离

相信很多开发者都默认Docker这样的容器是一种沙盒（sandbox）应用，也就是说他们可以用root权限在Docker中运行随便什么应用，而Docker有安全机制能保护宿主系统。比如，有些人觉得Docker容器里面的进程跟虚拟机里面的进程一样安全；还有的人随便找个源就下载没有验证过的Docker镜像，看都不看内容就在宿主机器上尝试、学习和研究；还有一些提供PaaS服务的公司竟然允许用户向多租户系统中提交自己定制的Docker镜像。请注意，上述行为均是不安全的。

1. 何为安全性?

单单就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。而Docker的安全问题本质上就是容器技术的安全性问题，这包括共用内核问题以及Namespace还不够完善的限制：

/proc、/sys等未完全隔离 Top, free, iostat等命令展示的信息未隔离 Root用户未隔离 /dev设备未隔离 内核模块未隔离 SELinux、time、syslog等所有现有Namespace之外的信息都未隔离 当然，镜像本身不安全也会导致安全性问题。

2. 与虚拟机对比安全性

其实传统虚拟机系统也绝非100%安全，只需攻破Hypervisor便足以令整个虚拟机毁于一旦，问题是有谁能随随便便就攻破吗？如上所述，Docker的隔离性主要运用Namespace 技术。传统上Linux中的PID是唯一且独立的，在正常情况下，用户不会看见重复的PID。然而在Docker采用了Namespace，从而令相同的PID可于不同的Namespace中独立存在。举个例子，A Container 之中PID=1是A程序，而B Container之中的PID=1同样可以是A程序。虽然Docker可透过Namespace的方式分隔出看似是独立的空间，然而Linux内核（Kernel）却不能Namespace，所以即使有多个Container，所有的system call其实都是通过主机的内核处理，这便为Docker留下了不可否认的安全问题。

传统的虚拟机同样地很多操作都需要通过内核处理，但这只是虚拟机的内核，并非宿主主机内核。因此万一出现问题时，最多只影响到虚拟系统本身。当然你可以说黑客可以先Hack虚拟机的内核，然后再找寻Hypervisor的漏洞同时不能被发现，之后再攻破SELinux，然后向主机内核发动攻击。文字表达起来都嫌繁复，更何况实际执行？所以Docker是很好用，但在迁移业务系统至其上时，请务必注意安全性！

3. 如何解决安全性问题

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：

- 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。

- 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。

- 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。

- 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。

- 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)

- 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

  未合入原因：cgroupfs已经导出了所有信息，/proc展现的工作可以由用户态实现，比如fuse。 不过，从08年cgroup/ns基本成型后，至今还没有新的namespace加入内核，cgroup在子系统上做了简单的补充，多数工作都是对原有subsystem的完善。内核社区对容器技术要求的隔离性，本的原则是够用就好，不能把内核搞的太复杂。

一些企业也做了很多工作，比如一些项目团队采用了层叠式的安全机制，这些可选的安全机制具体如下：

1、文件系统级防护

文件系统只读：有些Linux系统的内核文件系统必须要mount到容器环境里，否则容器里的进程就会罢工。这给恶意进程非常大的便利，但是大部分运行在容器里的App其实并不需要向文件系统写入数据。基于这种情况，开发者可以在mount时使用只读模式。比如下面几个： /sys 、/proc/sys 、/proc/sysrq-trigger 、/proc/irq、/proc/bus 写入时复制（Copy-On-Write）： Docker采用的就是这样的文件系统。所有运行的容器可以先共享一个基本文件系统镜像，一旦需要向文件系统写数据，就引导它写到与该容器相关的另一个特定文件系统中。这样的机制避免了一个容器看到另一个容器的数据，而且容器也无法通过修改文件系统的内容来影响其他容器。 2、Capability机制

Linux对Capability机制阐述的还是比较清楚的，即为了进行权限检查，传统的UNIX对进程实现了两种不同的归类，高权限进程（用户ID为0，超级用户或者root），以及低权限进程（UID不为0的）。高权限进程完全避免了各种权限检查，而低权限进程则要接受所有权限检查，会被检查如UID、GID和组清单是否有效。从2.2内核开始，Linux把原来和超级用户相关的高级权限划分成为不同的单元，称为Capability，这样就可以独立对特定的Capability进行使能或禁止。通常来讲，不合理的禁止Capability，会导致应用崩溃，因此对于Docker这样的容器，既要安全，又要保证其可用性。开发者需要从功能性、可用性以及安全性多方面综合权衡Capability的设置。目前Docker安装时默认开启的Capability列表一直是开发社区争议的焦点，作为普通开发者，可以通过命令行来改变其默认设置。

3、NameSpace机制

Docker提供的一些命名空间也从某种程度上提供了安全保护，比如PID命名空间，它会将全部未运行在开发者当前容器里的进程隐藏。如果恶意程序看都看不见这些进程，攻击起来应该也会麻烦一些。另外，如果开发者终止pid是1的进程命名空间，容器里面所有的进程就会被全部自动终止，这意味着管理员可以非常容易地关掉容器。此外还有网络命名空间，方便管理员通过路由规则和iptable来构建容器的网络环境，这样容器内部的进程就只能使用管理员许可的特定网络。如只能访问公网的、只能访问本地的和两个容器之间用于过滤内容的容器。

4、Cgroups机制

主要是针对拒绝服务攻击。恶意进程会通过占有系统全部资源来进行系统攻击。Cgroups机制可以避免这种情况的发生，如CPU的cgroups可以在一个Docker容器试图破坏CPU的时候登录并制止恶意进程。管理员需要设计更多的cgroups，用于控制那些打开过多文件或者过多子进程等资源的进程。

5、SELinux

SELinux是一个标签系统，进程有标签，每个文件、目录、系统对象都有标签。SELinux通过撰写标签进程和标签对象之间访问规则来进行安全保护。它实现的是一种叫做MAC（Mandatory Access Control）的系统，即对象的所有者不能控制别人访问对象。

4. 安全建议

最简单的就是不要把Docker容器当成可以完全替代虚拟机的东西。跑在Docker容器中的应用在很长一段时间内都将会是选择性的，通常只跑测试系统或可信业务。

门槛再高一点，我们对系统做减法，通过各种限制来达到安全性。这也是最主流的、有效的安全加固方法，比如上一章节介绍的几种安全机制。同时一定要保证内核的安全和稳定。外部工具的监控、容错等系统也必不可少。

总之通过适配、加固的Docker容器方案，在安全性上完全可以达到商用标准。就是可能对实施人员的技术要求和门槛较高。

特性

1. 高密度 高弹性

2. 兼具IaaS的灵活和PaaS的便利
3. 容器化应用是基石，一切都封装在镜像里
4. 实现更快速的交付和部署
5. 更易于微服务架构的实现
6. 更高效的虚拟化
7. 容器的启动是（毫）秒级的
8. 像搭积木一样的进行资源编排
9. 更简单的管理，负载均衡、弹性伸缩、日志监控、滚动升级等举手可得
10. 易于扩展和迁移

复用容器

借助经过全面调优的容器系统，你就可以在同一硬件上拥有数量比使用Xen虚拟机或KVM虚拟机多出四到六倍的服务器应用实例。

容器可以追溯到至少2000年和FreeBSD Jails。甲骨文Solaris也有一个类似概念，名为Zones;Parallels、谷歌和Docker等公司一直在致力于研发诸如OpenVZ和LXC(Linux容器)之类的开源项目，旨在让容器运行起来顺畅又安全。

的确，很少有人知道容器，但大多数人多年来一直在使用容器。谷歌就有自己的开源容器技术lmctfy(Let Me Contain That For You，意为“让我容纳你的程序”)。只要你使用谷歌的某项功能：比如搜索、Gmail、Google Docks或无论其他什么，就分配了一个新的容器。

然而，Docker建立在LXC的基础上。与任何容器技术一样，就该程序而言，它有自己的文件系统、存储系统、处理器和内存等部件。容器与虚拟机之间的区别主要在于，虚拟机管理程序对整个设备进行抽象处理，而容器只是对操作系统内核进行抽象处理。

这反过来意味着：虚拟机管理程序能做容器做不了的一件事就是，使用不同的操作系统或内核。所以，举例说，你可以使用微软Azure，同时运行Windows Server2012的实例和SUSE Linux企业级服务器的实例。至于Docker，所有容器都必须使用同样的操作系统和内核。

另一方面，如果你只是想让尽可能多的服务器应用实例在尽可能少的硬件上运行，可能不大关心运行多个操作系统虚拟机。要是同一应用程序的多个副本正是你需要的，那么你会喜欢上容器。

改用Docker这一举措有望每年为数据中心或云计算服务提供商节省数千万美元的电力和硬件成本。所以难怪它们在一窝蜂地尽快采用Docker。

参考资料

参考资料名称	链接地址
浅谈Docker隔离性和安全性	http://www.freebuf.com/articles/system/69809.html
【深入浅出容器云】关于容器云你不得不知的十大特性	https://phphub.org/topics/1836