## 第三次讨论课

姓名：谢明玥

学号：135293

1. 容器技术的理解
2. 概念

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。

1. 关键技术
2. 隔离

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

1. 安全

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

1. 挑战
2. 管理（Orchestrating at Scale）

Docker使用cgroups 归类运行在容器中的进程。这使你可以管理一组进程的资源，可想而知，这是非常宝贵的。

如果你运行一个操作系统，其使用systemd管理服务。每个进程（不仅仅是容器中的进程）都将被放入一个cgroups树中。如果你运行systemd-cgls命令，你自己可以看到这个结构：

$ systemd-cgls

├─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 22

├─machine.slice

│ └─machine-qemu\x2drhel7.scope

│   └─29898 /usr/bin/qemu-system-x86\_64 -machine accel=kvm -name rhel7 -S -machine pc-i440fx-1.6,accel=kvm,usb=off -cpu SandyBridge -m 2048

├─system.slice

│ ├─avahi-daemon.service

│ │ ├─ 905 avahi-daemon: running [mistress.local

│ │ └─1055 avahi-daemon: chroot helpe

│ ├─dbus.service

│ │ └─890 /bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation

│ ├─firewalld.service

│ │ └─887 /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid

│ ├─lvm2-lvmetad.service

│ │ └─512 /usr/sbin/lvmetad -f

│ ├─abrtd.service

│ │ └─909 /usr/sbin/abrtd -d -s

│ ├─wpa\_supplicant.service

│ │ └─1289 /usr/sbin/wpa\_supplicant -u -f /var/log/wpa\_supplicant.log -c /etc/wpa\_supplicant/wpa\_supplicant.conf -u -f /var/log/wpa\_supplica

│ ├─systemd-machined.service

│ │ └─29899 /usr/lib/systemd/systemd-machined

[SNIP]

当我们想管理资源的时候，这个方法提供了很大的灵活性，因为我们可以分别管理每个组。

Docker能够指定（通过运行命令的-c开关）给一个容器的可用的CPU分配值。这是一个相对权重，与实际的处理速度无关。事实上，没有办法说一个容器只可以获得1Ghz CPU。请记住。

每个新的容器默认的将有1024CPU配额，当我们单独讲它的时候，这个值并不意味着什么。但是如果我们启动两个容器并且两个都将使用 100%CPU，CPU时间将在这两个容器之间平均分割，因为它们两个都有同样的CPU配额（为了简单起见，假设没有任何其他进程在运行）。

如果我们设置容器的CPU配额是512，相对于另外一个容器，它将使用一半的CPU时间。但这不意味着它仅仅能使用一半的CPU时间。如果另外一个容器（1024配额的）是空闲的 - 我们的容器将被允许使用100%CPU。这是需要注意的另外一件事。

限制仅仅当它们应该被执行的时候才会强制执行。CGroups不限制进程预先使用（比如，不允许它们更快地运行即使它们有空余资源）。相反的，它提供了它尽可能提供的以及它仅仅在必需的时候限制（比如，当太多的进程同时大量地使用CPU）。

当然，这很难说清楚（我想说的是这不可能说清楚的）多少资源应该被分配给你的进程。这实际取决于其他进程的行为以及多少配额被分配给它们。

1. 容器技术的理解
2. 我们需要和希望解决的问题

Docker安全性问题

1. 选取什么样的容器技术方案

* 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。

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

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。

* 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。

* 参考资料

《Docker容器资源管理》

<http://www.lupaworld.com/article-250948-1.html>

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

《Docker背后的内核知识——Namespace资源隔离》

<http://www.infoq.com/cn/articles/docker-kernel-knowledge-namespace-resource-isolation?utm_source=tuicool>