讨论课 03

1352911 曹琦

关键技术

概述

容器将应用程序和库及其运行所需的其它二进制文件打包在一起，从而为应用程序提供了一个独立的操作系统环境。Docker 是容器技术行业的领先者，本部分主要基于docker介绍容器技术的关键和优势。

我们以集装箱类比容器如下：

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，应用发布以容器为单位
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

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。

Linux Namespace是Linux提供的一种内核级别环境隔离的方法。Linux Namespace在文件系统不能访问外部内容的基础上，提供了对UTS、IPC、mount、PID、network、User等的隔离机制。下面举例说明：

举例说明，Linux下的超级父亲进程的PID是1，所以如果我们可以把用户的进程空间jail到某个进程分支下，并让其下面的进程看到的那个超级父进程的PID为1，就可以达到资源隔离的效果了（不同的PID namespace中的进程无法看到彼此）。

Linux Namespace 有如下种类：

分类	系统调用参数	相关内核版本
Mount namespaces	CLONE_NEWNS	Linux 2.4.19
UTS namespaces	CLONE_NEWUTS	Linux 2.6.19
IPC namespaces	CLONE_NEWIPC	Linux 2.6.19
PID namespaces	CLONE_NEWPID	Linux 2.6.24
Network namespaces	CLONE_NEWNET	始于Linux 2.6.24
User namespaces	CLONE_NEWUSER	始于 Linux 2.6.23

主要是三个系统调用：

- clone() – 实现线程的系统调用，用来创建一个新的进程，并可以通过设计上述参数达到隔离。
- unshare() – 使某进程脱离某个namespace
- setns() – 把某进程加入到某个namespace

Namespace解决的问题主要是环境隔离的问题，这只是虚拟化中最最基础的一步，我们还需要解决对计算机资源使用上的隔离。**Linux CGroup**全称Linux Control Group， 是Linux内核的一个功能，用来限制，控制与分离一个进程组群的资源（如CPU、内存、磁盘输入输出等）。

Linux CGroup主要提供了如下功能：

- Resource limitation: 限制资源使用，比如内存使用上限以及文件系统的缓存限制。
- Prioritization: 优先级控制，比如：CPU利用和磁盘IO吞吐。
- Accounting: 一些审计或一些统计，主要目的是为了计费。
- Control: 挂起进程，恢复执行进程。

使用 cgroup，系统管理员可更具体地控制对系统资源的分配、优先顺序、拒绝、管理和监控。可更好地根据任务和用户分配硬件资源，提高总体效率。在实践中，系统管理员一般会利用CGroup做下面这些事（有点像为某个虚拟机分配资源似的）：

- 隔离一个进程集合（比如：nginx的所有进程），并限制他们所消费的资源，比如绑定CPU的核。
- 为这组进程 分配其足够使用的内存
- 为这组进程分配相应的网络带宽和磁盘存储限制
- 限制访问某些设备（通过设置设备的白名单）

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
- 当然，镜像本身不安全也会导致安全性问题。

所以，我们需要接受“容器不是全封闭”的思想，之后改进Docker的安全性。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。
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
   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。

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

Docker采用的就是写入时复制（Copy-On-Write）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

Capability机制是为了进行权限检查，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。

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。

为避免安全问题，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。

轻量级

Docker在cpu, memory, disk IO, network IO上的性能损耗表现优秀。Container的快速创建、启动、销毁受到很多赞誉。

运行在单个机器上的容器共享同一个操作系统内核，所以它们可以即时启动并且更有效地使用RAM。镜像构建在分层的文件系统上所以它们可以共享公有文件，使硬盘使用和镜像下载更加高效。

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。

Docker & Virtual Machine

正如Docker 官网所言，“Containers have similar resource isolation and allocation benefits as virtual machines but a different architectural approach allows them to be much more portable and efficient.”
Virtual Machine: 每一个虚拟机都包含了应用，必要的二进制文件和所需要的库以及单独的操作系统，对存储资源有极大的消耗。（如 figure 1 所示）
Containers: 容器包含了应用及各自的库文件等，但是共享同一个内核。它们在主操作系统上的用户空间独立运行。它们也不依赖于特定的基础硬件。Docker 容器可以运行在任意计算机及云平台上。

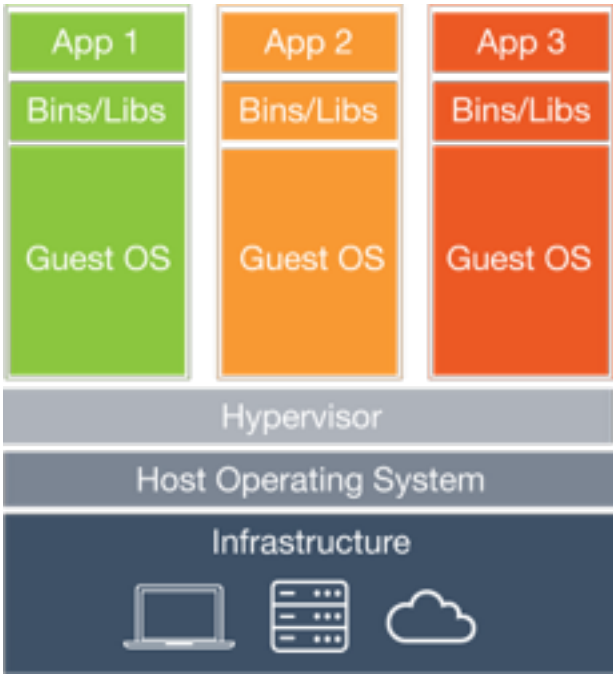

figure 1 Virtual Machine

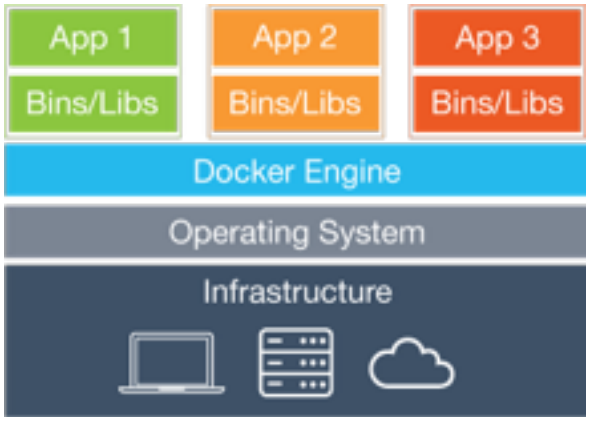

figure 2 Containers

虽然Docker相对于虚拟机在轻便性上有优势，但是在安全性方面却不如虚拟机。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，多个容器共用内核，所有的system call 都是通过主机内核处理的，这为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。

解决方案

蘑菇街是一个网上销售平台，对于蘑菇街而言，每年的 11.11 已经成为一年中最大的考验，考验的是系统稳定性，容灾能力，紧急故障处理，运维等各个方面的能力。蘑菇街的私有云平台经历了 3 次大促，稳定性方面得到了初步验证。

蘑菇街的私有云平台（以下简称蘑菇街私有云）是蘑菇街面向内部上层业务提供的基础性平台。通过基础设施的服务化和平台化，可以使上层业务能够更加专注在业务自身，而不是关心底层运行环境的差异性。它通过基于 Docker 的 CaaS 层和 KVM 的 IaaS 层来为上层提供 IaaS/PaaS 层的云服务，以提高物理资源的利用率，以及业务部署和交付的效率，并促进应用架构的拆分和微服务化。

在架构选型的时候， Docker 的轻量化，秒级启动，标准化的打包 / 部署 / 运行的方案，镜像的快速分发，基于镜像的灰度发布等特性，都十分适合蘑菇街的应用场景，即适合于弹性计算。而 Docker 自身的集群管理能力在当时条件下还很不成熟，因此我们没有选择刚出现的 Swarm，而是用了业界最成熟的 OpenStack，这样能同时管理 Docker 和 KVM 虚拟机。相对来说，Docker 适合于无状态，分布式的业务，KVM 适合对安全性，隔离性要求更高的业务。

针对Docker的安全性和隔离性有限的问题，蘑菇街采取了如下安全措施：

- **实时pid数量监控**。内核中的 pid_max(/proc/sys/kernel/pid_max)是全局共享的。当一个容器中的 pid 数目达到上限 32768，会导致宿主机和其他容器无法创建新的进程。
- **内存使用监控**。cgroup 提供的内存使用值是不准确的，比真实使用的内存值要低。因为内核 memcg 无法回收 slab cache，也不对 dirty cache 量进行限制，所以很难估算容器真实的内存使用情况。蘑菇街调整了容器内存的计算算法，精确内存计算。
- **日志乱序**。Docker 的宿主机内核日志中经常会产生字符乱序，这样会导致日志监控无法取到正确的关键字进行报警。通过修改 container 里的 rsyslog 配置，只让宿主机去读 kernel 日志即可解决问题。
- **隔离开关**。平时我们的容器是严格隔离的，包括 CPU、内存和磁盘 IO，网络 IO等。但双十一的业务量可能是平时的十几倍或几十倍。蘑菇街为双十一做了不少开关，在压力大的情况下，我们可以为个别容器进行动态的 CPU，内存等扩容或缩容，调整甚至放开磁盘 iops 限速和网络的 TC 限速。
- **灾备和紧急故障处理**。蘑菇街做了大量的灾备预案和技术**准备，**研究了不启动 Docker Daemon 的情况下，离线恢复 Docker 中数据的方法。具体来说，是用 dmsetup create 命令创建一个临时的 dm 设备，映射到 Docker 实例所用的 dm 设备号，通过 mount 这个临时设备，就可以恢复出原来的数据。
- **性能优化**。蘑菇街从系统层面也对 docker 做了大量的优化，比如针对磁盘 IO 的性能瓶颈，调优了像 vm.dirty_expire_centisecs,vm.dirty_writeback_centisecs,

vm.extra_free_kbytes 这样的内核参数。还引入了 Facebook 的开源软件 flashcache，将 SSD 作为 cache，显著的提高 docker 容器的 IO 性能。并且还通过合并镜像层次来优化 docker pull 镜像的时间。在 docker pull 时，每一层校验的耗时很长，通过减小层数，不仅大小变小，docker pull 时间也大幅缩短。

总体来说，容器的轻量性使它能在当今时代得到很好的应用和发展，适合于无状态，分布式的业务，在弹性计算（蘑菇街 & 京东）和构建PaaS平台（阿里百川 & SAE）方面都有优势，但是在使用时需要综合考虑其隔离和安全性方面的瑕疵，容器常常和虚拟机一起使用，同时设计安全保护机制。

参考资料

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

[2] Docker基础技术：Linux Namespace http://coolshell.cn/articles/17010.html

[3] Docker基础技术：Linux CGroup http://coolshell.cn/articles/17049.html

[4] 浅谈Docker隔离性和安全性 http://www.freebuf.com/articles/system/69809.html

[5] What Docker? https://www.docker.com/what-docker

[6] 10个精选的容器应用案例 http://www.infoq.com/resource/minibooks/10-Container-studies/zh/pdf/10%20Container%20studies.pdf