讨论课3

张航 1352960

容器技术的理解

在这个时代，为自己的网络服务和应用开辟一片市场，需要敏锐的洞察力和强大的研发部门。全球各地的公司一直都在探索推动敏捷的新方式，以缩短产品的发布周期。

微服务便是其中一个方法。它是一种系统架构的设计理念。在这种架构中，复杂的应用程序由多个较小的独立进程组合而成，这些独立进程有自己的运行环境和权限。进程之间通过语言无关的API接口或者简单的协议进行通信，比如远程方法调用（RMI），RESTful Web服务或消息推送。模块化的设计便于公司将工作分发到不同的团队，每个团队也有更大的自由度去开发自己的功能模块，有助于加快开发周期。

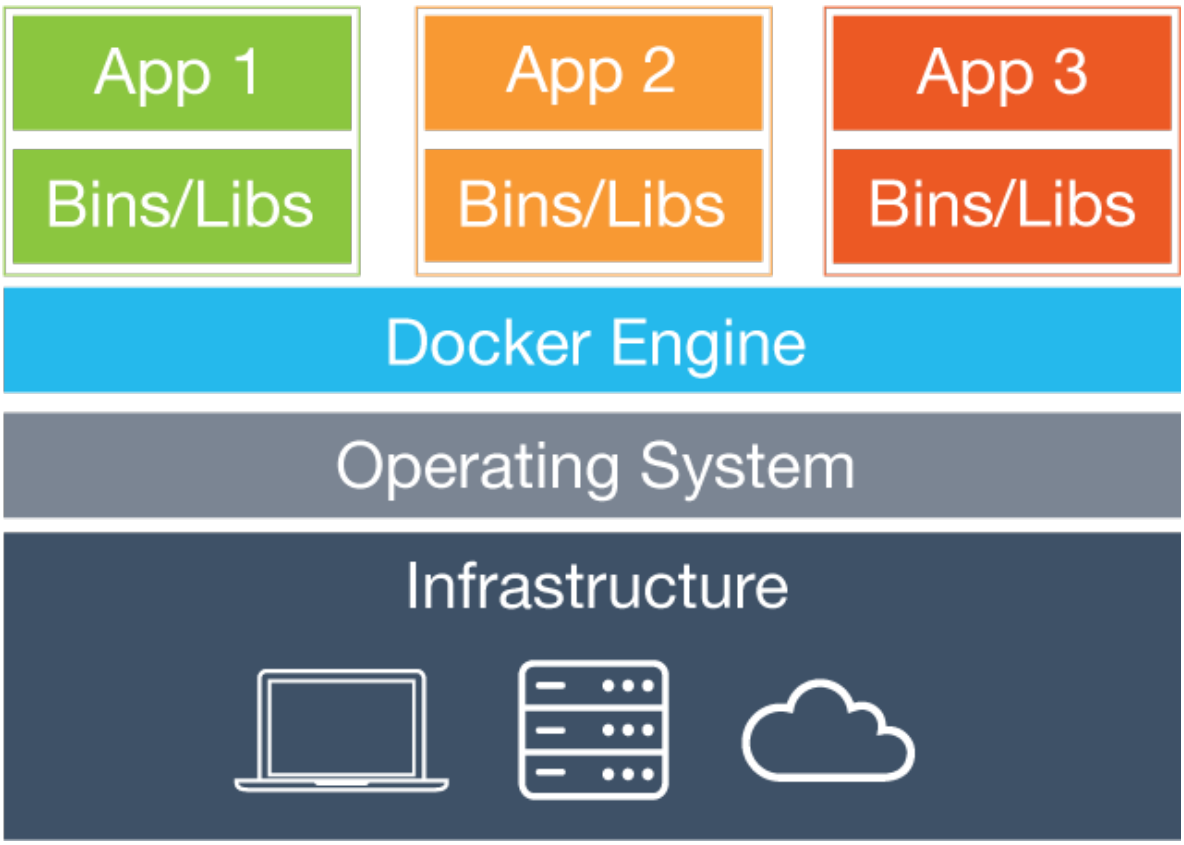
每个组件都有自己的存储空间，内存或CPU资源，便于分配和跟踪硬件的使用情况，尤其是在PaaS环境中。在保证组件接口能够与其它组件正常通信的前提下，开发人员可以使用任何技术栈进行开发。尽管微服务架构降低了管理的复杂度，但是在涉及对生产环境进行监控和升级时，仍然需要关注团队之间的合作，比如防止一个组件更新导致另一个组件的异常。详尽的文档可以帮助项目新人快速熟悉代码库，有助于减少类似问题的出现。

微服务架构的早期用户包括亚马逊网络服务（AWS）、谷歌、 eBay 和Netflix，针对频繁更新的服务和应用，他们都在不断提升持续交付的体验。这些服务和应用通过网络分发给用户的PC、平板和智能手机。拥有如此高逼格的倡导者，微服务概念迅速扩展也就不足为奇。最近IDG Connect对一百名在美国、英国、以色列和印度的高级IT员工进行了一项调查，调查表明，只有6%表示对微服务一无所知。只有少数人（10%）表示他们已经使用微服务架构，超过四分之一（28%）表示明年将会使用微服务，27%表示未来会采用微服务架构。

目前，有大量的服务和应用都部署在云端，而微服务架构严重依赖于容器技术。一个容器只会使用虚拟的操作系统，而不需要整个虚拟机，也不需要底层硬件。微服务进程运行在容器中，由容器提供隔离。

Docker是最常见和使用最广泛的容器技术，它是一个基于Linux的开源软件，目前已经获取多个公司的支持，包括Canonical，Red Hat和Parallels。现有的PaaS服务，如Google App Engine，Red Hat Open Shift，以及VMware的Cloud Foundry也支持Linux容器技术（LXC）。

微服务仍然是一个新兴的技术，对容器的支持计划在很多公司看来也很领先。IDG Connect的调查显示，18%的组织已经部署某些形式的容器平台，另外57%计划在未来这样做，只有3%的人说他们没有任何关于容器的计划。在不久的将来，微服务和容器的结合将在应用和服务开发领域占据重要的位置。



隔离

容器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。

Linux内核实现namespace的主要目的之一就是实现轻量级虚拟化(容器)服务。在同一个namespace下的进程可以感知彼此的变化，而对外界的进程一无所知。这样就可以让容器中的进程产生错觉，仿佛自己置身于一个独立的系统环境中，以达到独立和隔离的目的。+

一般,Docker容器需要并且Linux内核也提供了这6种资源的namespace的隔离：

- UTS : 主机与域名
- IPC : 信号量、消息队列和共享内存
- PID : 进程编号
- NETWORK : 网络设备、网络栈、端口等
- Mount : 挂载点(文件系统)
- User : 用户和用户组

Pid namespace

用户进程是lxc-start进程的子进程，不同用户的进程就是通过pid namespace隔离开的，且不同namespace中可以有相同PID,具有以下特征： 1、每个namespace中的pid是有自己的pid=1的进程(类似/sbin/init进程） 2、每个namespace中的进程只能影响自己的同一个namespace或子namespace中的进程 3、因为/proc包含正在运行的进程，因此而container中的pseudo-filesystem的/proc目录只能看到自己namespace中的进程 4、因为namespace允许嵌套，父进程可以影响子namespace进程，所以子namespace的进程可以在父namespace中看到，但是具有不同的pid。

Net namespace

有了pid namespace ,每个namespace中的pid能够相互隔离，但是网络端口还是共享host的端口。网络隔离是通过netnamespace实现的， 每个net namespace有独立的network devices, IP address, IP routing tables, /proc/net目录，这样每个container的网络就能隔离开来， LXC在此基础上有5种网络类型，docker默认采用veth的方式将container中的虚拟网卡同host上的一个docker bridge连接在一起。

IPC namespace Container

IPC namespace Container中进程交互还是采用linux常见的进程间交互方法(interprocess communication-IPC)包括常见的信号量、消息队列、内存共享。然而同VM不同， container的进程交互实际是host具有相同pid namespace中的进程间交互，因此需要在IPC资源申请时加入namespace信息，每个IPC资源有一个唯一的32bit ID。

Mnt namespace

Mnt namespace类似chroot ,将一个进程放到一个特定的目录执行，mnt namespace允许不同namespace的进程看到的文件结构不同，这样每个namespace中的进程所看到的文件目录被隔离开了，同chroot不同，每个namespace中的container在/proc/mounts的信息只包含所在namespace 的mount point。

Uts namespace UTS

Uts namespace UTS(UNIX Time sharing System）namespace允许每个container拥有独立地hostname和domain name,使其在网络上被视作一个独立的节点而非Host上的一个进程 User namespace 每个container可以有不同的user和group id ,也就是说可以container内部的用户在container内部执行程序而非 Host上的用户。

安全

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。

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！

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

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

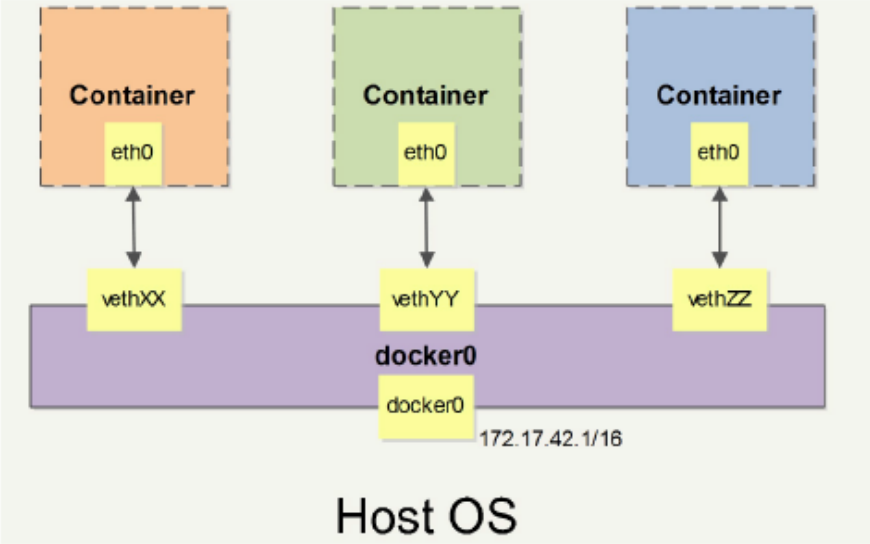
最简单的就是不要把Docker容器当成可以完全替代虚拟机的东西。跑在Docker容器中的应用在很长一段时间内都将会是选择性的，通常只跑测试系统或可信业务。

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。就是可能对实施人员的技术要求和门槛较高。

管理

Docker启动时，会自动在主机上创建一个docker0虚拟网桥，实际上是Linux的一个bridge,可以理解为一个软件交换机，它会而挂载到它的网口之间进行转发 当创建一个Docker容器的时候，同理会创建一对veth pair接口(当数据包发送到一个接口时，另外一个接口也可以收到相同的数据包)，这对接口一端在容器内，即eth0;另一端在本地并被挂载到docker0网桥，名称以veth开头。



Docker容器的5种网络模式

在使用docker run创建docker容器时，可以用--net选项指定容器的网络模式，Docker有以下5种网络模式：

bridge模式

使用docker run --net=bridge指定，bridge模式是Docker默认的网络设置，此模式会为每一个容器分配Network Namespace、设置IP等，并将一个主机上的Docker容器连接到一个虚拟网桥上。 此模式与外界通信使用NAT协议，增加了通讯的复杂性，在复杂场景下使用会有诸多限制。 route -n 查看 IP routing tables; iptables -t nat -L -n 查看iptables rules.

host模式

使用docker run --net=host指定，这种模式Docker Server将不为Docker容器创建网络协议栈，即不会创建独立的network namespace,Docker容器中的进程处于宿主机的网络环境中，相当于Docker容器的宿主机共用同一个network namespace,使用宿主机的网卡、IP、端口等信息。此模式没有网络隔离性，同时会引起网络资源的竞争与冲突。

container模式

使用docker run --net=container:othercontainer_name指定，这种模式与host模式相似，指定新创建的容器和

已经存在的某个容器共享同一个network namespace, 以下两种模式都共享network namespace,区别就在于host模与宿主机共享，而container模式与某个存在的容器共享。 在container模式下，两个容器的进程可以通过lo回环网络设备通讯，增加了容器间通讯的便利性和效率。container模式的应用场景就在于可以将一个应用的多个组件放在不同的容器趾，这些 容器配成container模式的网络，这样它们可以作为一个整体对外提供服务。同时，这种模式也降低了容器间的隔离性。 docker run -it --name helloworld busybox sh docker run -it --name helloword-con --net=container:helloword busybox sh

none模式

使用docker run --net=none指定，在这种模式下，Docker容器拥有自己的Network Namespace，但是，并不为Docker容器进行任何网络配置。也就是说，这个Docker容器没有网卡、IP、路由等信息。需要我们自己为Docker容器添加网卡、配置IP等。这种模式如果不进行特定的配置是无法正常使用的，但它也给了用户最大的自由度来自定义容器的网络环境。

overlay模式

overlay网络特点：

- 跨主机通讯
- 无需做端口映射
- 无需担心IP冲突

DaoCloud

DaoCloud产品服务包括：

- DaoCloud 应用管理平台。 DaoCloud应用管理平台对接GitHub等国内外代码托管库，无缝完成持续集成，容器镜像构建，镜像托管和云平台资源调配。DaoCloud采用标准化容器技术缩短应用交付周期，降低发布风险，更高效利用IaaS云主机资源。
- DaoStack企业服务。服务包括技术咨询、培训服务和定制解决方案，帮助企业快速掌握容器技术，建立以容器为核心的高效开发测试流程，与企业现有私有云资源对接，搭建容器管理云平台，让企业从容应对云端挑战。
- DaoMirror 镜像服务。DaoMirror采用Docker官方镜像技术，结合国内CDN网络，为开发者提供透明和高速的Docker Hub镜像下载解决方案。

与之前的云平台技术相比，DaoCloud创新性体现在：

1. 采用Docker轻量级虚拟化技术。针对分布式应用的痛点，推出支持多种语言和后台服务的DaoCloud持续集成服务。SaaS化的持续集成服务，是DaoCloud在国内的首创。
2. 提供创新性Docker镜像构建过程。包含了大量的网络层优化，为用户提供了基于集群环境的高速构建能力，极大的提升了发布效率。同时，依托DaoCloud遍布全球的云服务节点，能够帮助用户实现秒级的全球业务启停。
3. 采取了一键部署上云端的Docker化PaaS技术。允许用户在多种基础云平台之间灵活选择，以非常直观的方式实现应用的负载均衡、私有域名绑定、性能监控等应用生命周期服务。

DaoCloud总部位于上海，团队目前有20人，以技术人员为主。团队核心成员曾在PaaS领域耕耘多年，积累了虚拟化、云计算平台、大规模分布式系统、企业级云计算服务、开源社区建设等方面的丰富经验。

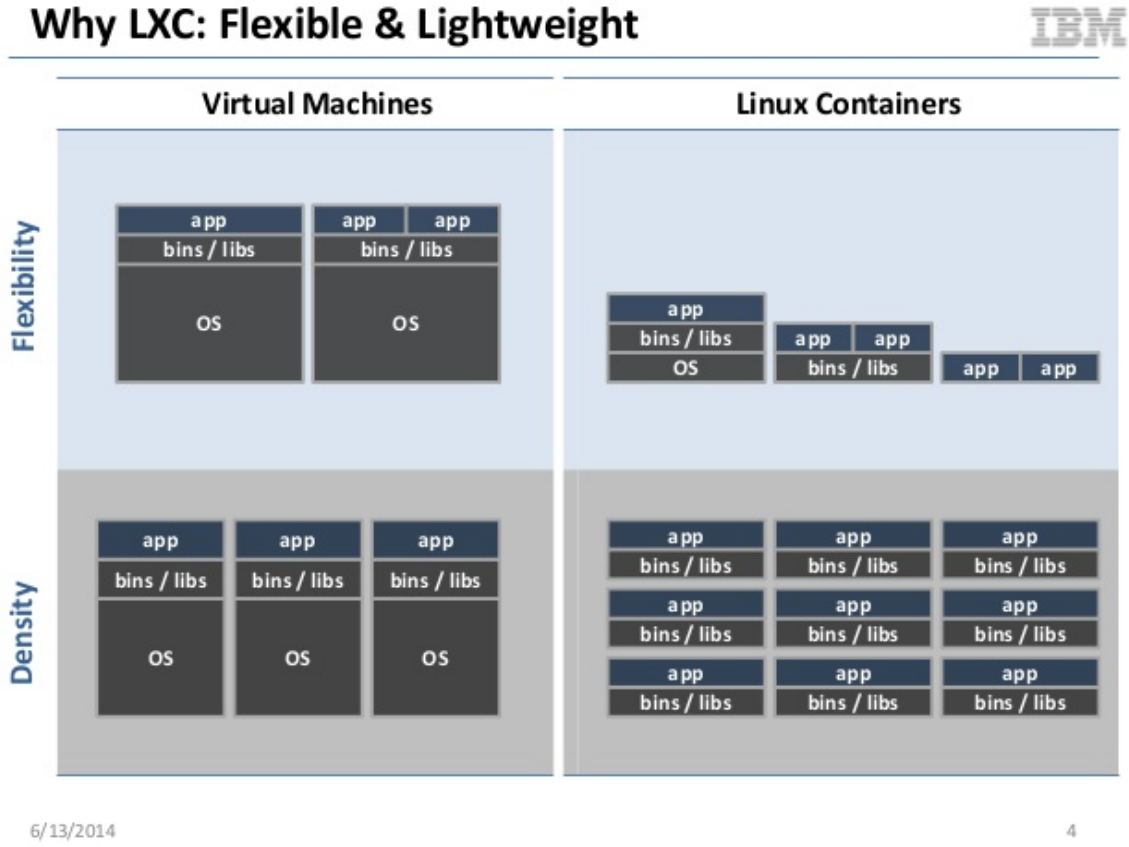
“Docker 代表了一种趋势”，团队创始人给创业邦这样介绍他们做DaoCloud的初衷：“互联网业务求新求快求变，需要新技术和方法论的出现来引领科技创新。以Docker为代表的容器技术就是这样一个顺势而为的产物。DaoCloud的团队作为中国最早的云平台技术布道者，一路见证了这场技术变革的威力。在这样的大环境下，DaoCloud团队决定离职创业，打造国内领先的Docker容器技术服务平台，并与各类云主机服务商对接，为开发者提供一站式解决方案。”

LXC

LXC，其名称来自Linux软件容器（Linux Containers）的缩写，一种操作系统层虚拟化（Operating system–level virtualization）技术，为Linux内核容器功能的一个用户空间接口。它将应用软件系统打包成一个软件容器（Container），内含应用软件本身的代码，以及所需要的操作系统核心和库。通过统一的名字空间和共用API来分配不同软件容器的可用硬件资源，创造出应用程序的独立沙箱运行环境，使得Linux用户可以容易的创建和管理系统或应用容器。[1]

在Linux内核中，提供了cgroups功能，来达成资源的区隔化。它同时也提供了名称空间区隔化的功能，使应用程序看到的操作系统环境被区隔成独立区间，包括进程树，网络，用户id，以及挂载的文件系统。但是cgroups并不一定需要引导任何虚拟机。

LXC利用cgroups与名称空间的功能，提供应用软件一个独立的操作系统环境。LXC不需要Hypervisor这个软件层，软件容器（Container）本身极为轻量化，提升了创建虚拟机的速度。软件Docker被用来管理LXC的环境。



参考资料

http://dockone.io/article/326

https://www.ibm.com/developerworks/cn/linux/l-lxc-containers/

https://hujb2000.gitbooks.io/docker-flow-evolution/content/cn/index.html

http://www.infoq.com/cn/articles/docker-container-management-libcontainer-depth-analysis