## 容器技术的代表------ Docker

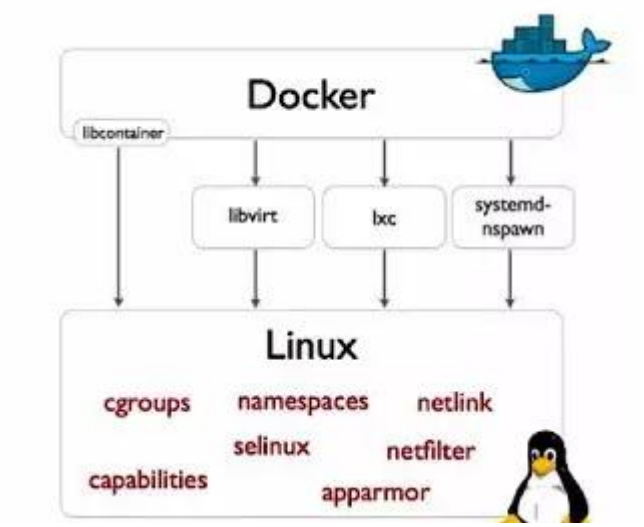
### Docker 简介：

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口。

Docker 故事：

> **环境**管理复杂 - 从各种 OS 到各种中间件到各种 app，一款产品能够成功作为开发者需要关心的东西太多，且难于管理，这个问题几乎在所有现代 IT 相关行业都需要面对。
>
> **云计算**时代的到来 - AWS 的成功，引导开发者将应用转移到 cloud 上，解决了硬件管理的问题，然而中间件相关的问题依然存在 (所以 openstack HEAT 和 AWS cloudformation 都着力解决这个问题)。开发者思路变化提供了可能性。
>
> 虚拟化手段的变化 - cloud 时代采用标配硬件来降低成本，采用虚拟化手段来满足用户按需使用的需求以及保证**可用性和隔离性**。然而无论是 KVM 还是 Xen 在 docker 看来,都在浪费资源，因为用户需要的是高效运行环境而非 OS, GuestOS 既浪费资源又难于管理，更加轻量级的 LXC 更加灵活和快速
>
> LXC 的移动性 - LXC 在 linux 2.6 的 kernel 里就已经存在了，但是其设计之初并非为云计算考虑的，缺少**标准化的描述手段和容器的可迁移性**，决定其构建出的环境难于迁移和标准化管理(相对于 KVM 之类 image 和 snapshot 的概念)。docker 就在这个问题上做出实质性的革新。这是 docker 最独特的地方。

### Docker 的安全与隔离：




## 虚拟化和隔离

操作系统级的虚拟化、容器、空间以及“chroot with steroids”，其实都定义了同一个概念：用户空间隔离。类似 Docker 的产品都使用了操作系统级的虚拟化，通过用户空间隔离可以提供额外的安全性。

Docker 包含了 libcontainer 库作为它直接虚拟化的方法，这个功能由 Linux 内核提供。 此外，它还通过 LXC[1],systemd-nspawn,和 libvert 使用了抽象虚拟接口。这些虚拟化库全部利用了 Linux 的原始容器（参见上图）

namespaces
cgroups
capabilities 等等。
Docker 在一个包装中联合了以上功能，并称之为容器格式。
libcontainer
默认的容器格式被称为 libcontainer。
Docker 也支持使用 LXC 的传统 Linux 容器。在将来，Docker 可能会支持其他的容器格式，比如结合 BSD jails 或者 Solaris Zones。
执行驱动程序是一种特殊容器格式的实现，用来运行 docker 容器。在最新的版本中，libcontainer 有以下特性：
是运行 docker 容器的默认执行驱动程序。
和 LXC 同时装载。
使用没有任何其他依赖关系的 Go 语言设计的库，来直接访问内核容器的 API。
目前的 Docker 涵盖的功能有：命名空间使用，cgroups 管理，capabilities 权限集，进程运行的环境变量配置以及网络接口防火墙设置——所有功能是固定可预测的，不依赖 LXC 或者其它任何用户区软件包。
只需提供一个根文件系统，和 libcontainer 对容器的操作配置，它会帮你完成剩下的事情。
支持新建容器或者添加到现有的容器。
事实上，对 libcontainer 最迫切的需求是稳定，开发团队也将其设为了默认。
在 Docker 0.9 中，LXC 现在可以选择关闭。
注意：LXC 在将来会继续被支持。
如果想要重新使用 LXC 驱动，只需输入指令 docker -d –e lxc,然后重启 Docker。

用户命名空间

Docker 不是虚拟化，相反的，它是一个支持命名空间抽象的内核，提供了独立工作空间(或容器)。当你运行一个容器的时候，Docker 为容器新建了一系列的 namespace。
一些 Docker 使用的 linux 命名空间：pid namespace 用作区分进程（PID: Process ID）。
容器中运行的进程就如同在普通的 Linux 系统运行一样，尽管它们和其他进程共享一个底层内核。
总之，cgroups 可以让 Docker：
实现组进程并且管理它们的资源总消耗。
分享可用的硬件资源到容器。
限制容器的内存和 CPU 使用。
可以通过更改相应的 cgroup 来调整容器的大小。
通过检查 Linux 中的/sys/fs/cgroup 对照组来获取容器中的资源使用信息。
提供了一种可靠的结束容器内所有进程的方法。

Capabilities
Linux 使用的是“POSIX capabilities”。这些权限是所有强大的 root 权限分割而成的一系列权限。在 Linux manpages 上可以找到所有可用权限的清单。Docker 丢弃了除了所需权限外的所有权限，使用了白名单而不是黑名单。一般服务器（裸机或者虚拟机）需要以 root 权限运行一系列进程。包括：
SSH

cron
syslogd
硬件管理工具 (比如负载模块)
网络配置工具 (比如处理 DHCP, WPA, or VPNs)等。
每个容器都是不同的，因为几乎所有这些任务都由围绕容器的基础设施进行处理。默认的，Docker 启用一个严格限制权限的容器。大多数案例中，容器不需要真正的 root 权限。举个例子，进程（比如说网络服务）只需要绑定一个小于 1024 的端口而不需要 root 权限：他们可以被授予 CAPNETBIND_SERVICE 来代替。因此，容器可以被降权运行：意味着容器中的 root 权限比真正的 root 权限拥有更少的特权。 Capabilities 只是现代 Linux 内核提供的众多安全功能中的一个。为了加固一个 Docker 主机，你可以使用现有知名的系统：

TOMOYO
AppArmor
SELinux
GRSEC, etc.
如果你的发行版本附带了 Docker 容器的安全模块，你现在就可以使用它们。比如，装载了 AppArmor 模板的 Docker 和 Red Hat 自带 SELinux 策略的 Docker。

**安全：**

确保 Docker 环境安全

确保容器部署安全性

硬件上的 Docker 安全中心

**Docker 的未来：**

Docker 的未来
Docker 公司已经建立了清晰的道路，即发展核心能力（libcontainer）、跨业务管理（libswarm）和容器间消息（libchan）。与此同时，通过收购果园实验室（Orchard labs），Docker 公司表达了利用自身生态系统的意愿。但是，这不仅仅关注 Docker 公司，这个项目的贡献者还来自于一些大牌公司，如谷歌、IBM 和 Red Hat。在仁慈的独裁者、首席技术官 Solomon Hykes 的掌舵下，Docker 公司和 Docker 项目的技术领先有着明确的联系。在项目初始的 18 个月里，它已经显示出通过自己的输出来快速前进的能力，并且没有减弱的迹象。
许多投资者正着眼于十年前 VMware 公司 ESX/ vSphere 平台的功能矩阵，试图找出已经由虚拟机普及而驱动的企业预期和现有 Docker 生态系统之间的差距（和机会）。在网络存储和细粒度的版本管理（用于容器中的内容）领域，现有 Docker 生态系统做得并不好，这就为初创企业和在职人员提供了机会。
随着时间的推移，虚拟机和容器（Docker 中的“运行”部分）之间的区别很可能变得不再那么重要，这将使注意力转到“构建（build）”和“交付（ship）”方面。这些变化将使 “Docker 会发生什么？”的问题，相比“Docker 会带给 IT 业什么？”的问题，变得更不重要。