讨论课 3 --复用云技术

1352871 康慧琳

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。。。

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

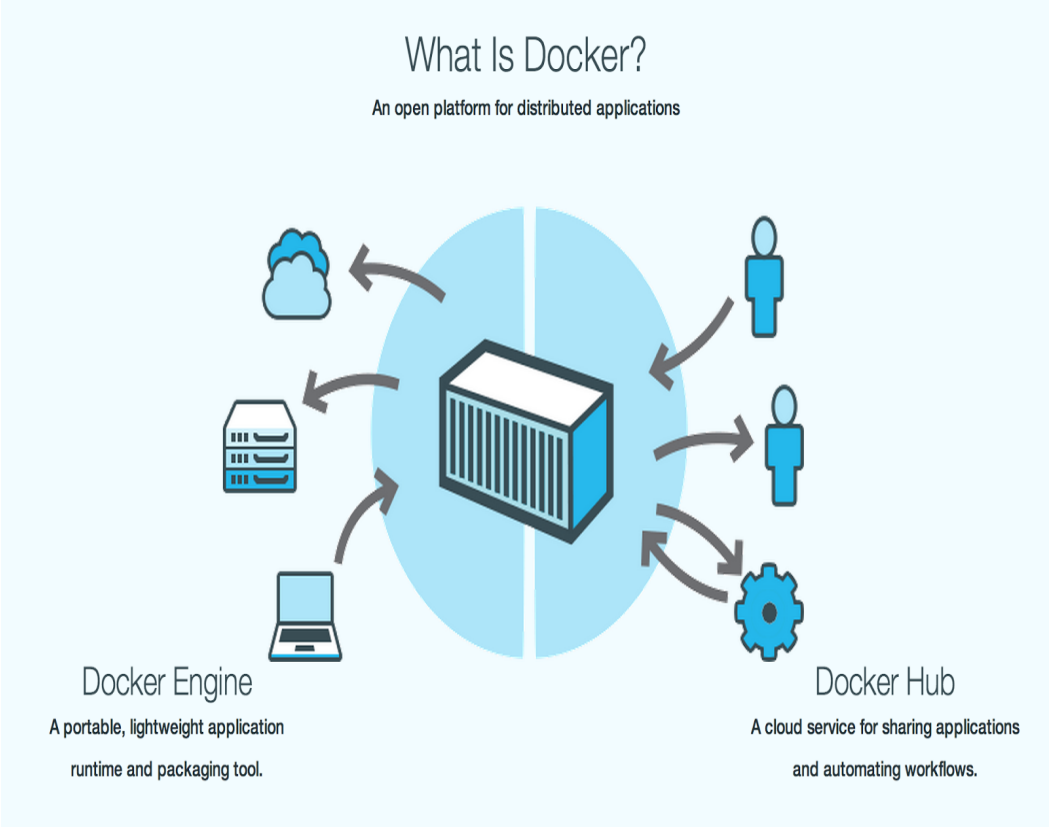
> 主要考虑容器技术：Docker

Docker简介

1. 由PaaS到Container

2013年2月，前Gluster的CEO Ben Golub和dotCloud的CEO Solomon Hykes坐在一起聊天时，Solomon谈到想把dotCloud内部使用的Container容器技术单独拿出来开源，然后围绕这个技术开一家新公司提供技术支持。28岁的Solomon在使用python开发dotCloud的PaaS云时发现，使用 LXC(Linux Container) 技术可以打破产品发布过程中应用开发工程师和系统工程师两者之间无法轻松协作发布产品的难题。这个Container容器技术可以把开发者从日常部署应用的繁杂工作中解脱出来，让开发者能专心写好程序；从系统工程师的角度来看也是一样，他们迫切需要从各种混乱的部署文档中解脱出来，让系统工程师专注在应用的水平扩展、稳定发布的解决方案上。他们越深入交谈，越觉得这是一次云技术的变革，紧接着在2013年3月Docker 0.1发布，拉开了基于云计算平台发布产品方式的变革序幕。

2. Docker



Docker是一个用于开发、交付和运行应用的开放平台，Docker设计用来更快的交付你的应用程序。Docker可以将你的应用程序和基础设施层隔离， 并且还可以将你的基础设施当作程序一样进行管理。Docker可以帮助你更块地打包你代码、测试以及部署，并且也可以减少从编写代码到部署运行代码的周期。

Docker将一个轻量级的容器虚拟化平台和一组标准工作流程、工具进行集成，来帮助你方便地管理和部署应用。

核心是，Docker提供了一种在安全隔离的容器中运行近乎所有应用的方式，这种隔离性和安全性允许你在同一主机上同时运行多个容器，而容器 的这种轻量级特性，意味着你可以节省更多的硬件资源，因为你不必消耗运行hypervisor所需要的额外负载。

核心技术预览

Docker核心是一个操作系统级虚拟化方法, 理解起来可能并不像VM那样直观。我们从虚拟化方法的四个方面：隔离性、可配额/可度量、便携性、安全性来详细介绍Docker的技术细节。

1. 隔离性: Linux Namespace(ns)

每个用户实例之间相互隔离, 互不影响。 一般的硬件虚拟化方法给出的方法是VM，而LXC给出的方法是container，更细一点讲就是kernel namespace。其中pid、net、ipc、mnt、uts、user等namespace将container的进程、网络、消息、文件系统、UTS("UNIX Time-sharing System")和用户空间隔离开。

1) pid namespace

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。所有的LXC进程在docker中的父进程为docker进程，每个lxc进程具有不同的namespace。同时由于允许嵌套，因此可以很方便的实现 Docker in Docker。

2) 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net namespace实现的， 每个net namespace有独立的 network devices, IP addresses, IP routing tables, /proc/net 目录。这样每个container的网络就能隔离开来。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: docker0连接在一起。

3) 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、消息队列和共享内存。然而同 VM 不同的是，container 的进程间交互实际上还是host上具有相同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 32 位 ID。

4) 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就被隔离开了。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。

5) uts namespace

UTS("UNIX Time-sharing System") namespace允许每个container拥有独立的hostname和domain name, 使其在网络上可以被视作一个独立的节点而非Host上的一个进程。

6) user namespace

每个container可以有不同的 user 和 group id, 也就是说可以在container内部用container内部的用户执行程序而非Host上的用户。

2. 可配额/可度量 - Control Groups (cgroups)

cgroups 实现了对资源的配额和度量。 cgroups 的使用非常简单，提供类似文件的接口，在 /cgroup目录下新建一个文件夹即可新建一个group，在此文件夹中新建task文件，并将pid写入该文件，即可实现对该进程的资源控制。groups可以限制blkio、cpu、cpuacct、cpuset、devices、freezer、memory、net_cls、ns九大子系统的资源，以下是每个子系统的详细说明：

blkio 这个子系统设置限制每个块设备的输入输出控制。例如:磁盘，光盘以及usb等等。

cpu 这个子系统使用调度程序为cgroup任务提供cpu的访问。

cpuacct 产生cgroup任务的cpu资源报告。

cpuset 如果是多核心的cpu，这个子系统会为cgroup任务分配单独的cpu和内存。

devices 允许或拒绝cgroup任务对设备的访问。

freezer 暂停和恢复cgroup任务。

memory 设置每个cgroup的内存限制以及产生内存资源报告。

net_cls 标记每个网络包以供cgroup方便使用。

ns 名称空间子系统。

3. 便携性: AUFS

AUFS (AnotherUnionFS) 是一种 Union FS, 简单来说就是支持将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)的文件系统, 更进一步的理解, AUFS支持为每一个成员目录(类似Git Branch)设定readonly、readwrite 和 whiteout-able 权限, 同时 AUFS 里有一个类似分层的概念, 对 readonly 权限的 branch 可以逻辑上进行修改(增量地, 不影响 readonly 部分的)。通常 Union FS 有两个用途, 一方面可以实现不借助 LVM、RAID 将多个disk挂到同一个目录下, 另一个更常用的就是将一个 readonly 的 branch 和一个

writeable 的 branch 联合在一 起，Live CD正是基于此方法可以允许在 OS image 不变的基础上允许用户在其上进行一些写操作。Docker 在 AUFS 上构建的 container image 也正是如此， 接下来我们从启动 container 中的 linux 为例来介绍 docker 对AUFS特性的运用。

4.安全性

单单就Docker来说，安全性可以概括为两点： ``` 1.不会对主机造成影响

2.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。

# docker面临的挑战

### 1.容器技术自身的成熟度

比如拿Docker 1.9推出的libnetwork，虽然支持的跨节点overlay网络的支持是一个巨大的进步，但是还有很多不足，比如容器间域名解析发 现要到将要发布的 Docker 1.10才能比较完善；同时跨节点网络性能依然存在问题，我们需要结合虚拟化技术一体解决；

Docker自身的安全性虽然已经得到很大提升，但是依然需要继续增强：比如用户名空间的支持还在试验阶段；对容器镜像的渗透检测、来源 追踪、授权管理等对生成环境管控很重要，但目前相关工具和产品还不完善等等
```

2.管控能力的缺失

在分布式环境下，对动态容器进行应用管理、监控、日志收集对依然存在挑战；目前Docker原生方案，流行的开源方案还有很多初创公司都 在提供自己的解决方案试图解决这些问题。在应用架构层面，微服务架构一方面很好的解决了移动互联网时代对扩展性和敏捷性需求；但另一方面由于分布 式系统和弹性部署的引入增加了运维的复杂性，同时可能造成技术的碎片化；这就需要容器平台能够更加自动化的进行运维，为用户提供以应用中心的管控 视图。

docker 应用场景

1. web应用的自动化打包和发布；
2. 自动化测试和持续集成、发布；
3. 在服务型环境中部署和调整数据库或其他的后台应用；
4. 从头编译或者扩展现有的OpenShift或Cloud Foundry平台来搭建自己的PaaS环境。

参考资料

参考资料名称	链接地址
【Docker官方文档】理解Docker	[http://www.open-open.com/lib/view/open1426036763326.html]http://www.open-open.com/lib/view/open1426036763326.html
【深度】阿里Docker服务开发中的5大挑战与经验沉淀	https://yq.aliyun.com/articles/4124
Docker安全性探讨与实践：探讨篇	http://www.infoq.com/cn/news/2014/09/docker-safety/
浅谈Docker隔离性和安全性	http://www.freebuf.com/articles/system/69809.html
深入浅出Docker（一）：Docker核心技术预览	http://www.infoq.com/cn/articles/docker-core-technology-preview/