Docker

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](http://baike.baidu.com/view/1634.htm) 机器上，也可以实现[虚拟化](http://baike.baidu.com/view/729629.htm)。容器是完全使用[沙箱](http://baike.baidu.com/subview/720343/13998082.htm)机制，相互之间不会有任何接口。

架构：

Docker 使用客户端-服务器 (C/S) 架构模式，使用远程API来管理和创建Docker容器。Docker 容器通过 Docker 镜像来创建。容器与镜像的关系类似于面向对象编程中的对象与类。

Docker采用 C/S架构 Docker daemon 作为服务端接受来自客户的请求，并处理这些请求（创建、运行、分发容器）。 客户端和服务端既可以运行在一个机器上，也可通过 socket 或者RESTful API 来进行通信。

Docker daemon 一般在宿主主机后台运行，等待接收来自客户端的消息。 Docker 客户端则为用户提供一系列可执行命令，用户用这些命令实现跟 Docker daemon 交互。

特性：

在docker的网站上提到了docker的典型场景：

* Automating the packaging and deployment of applications（使应用的打包与部署自动化）
* Creation of lightweight, private PAAS environments（创建轻量、私密的PAAS环境）
* Automated testing and continuous integration/deployment（使连续的集成/部署、测试自动化）
* Deploying and scaling web apps, databases and backend services（部署、衡量网页app、数据库和后台服务）

由于其基于LXC的轻量级虚拟化的特点，docker相比KVM之类最明显的特点就是启动快，资源占用小。因此对于构建隔离的标准化的运行环境，轻量级的PaaS(如dokku), 构建自动化测试和持续集成环境，以及一切可以横向扩展的应用(尤其是需要快速启停来应对峰谷的web应用)。

1. 构建标准化的运行环境，现有的方案大多是在一个baseOS上运行一套puppet/chef，或者一个image文件，其缺点是前者需要base OS许多前提条件，后者几乎不可以修改(因为copy on write 的文件格式在运行时rootfs是read only的)。并且后者文件体积大，环境管理和版本控制本身也是一个问题。
2. PaaS环境是不言而喻的，其设计之初和dotcloud的案例都是将其作为PaaS产品的环境基础
3. 因为其标准化构建方法(buildfile)和良好的REST API，自动测试和持续集成/部署能够很好的集成进来
4. 因为LXC轻量级的特点，其启动快，而且docker能够只加载每个container变化的部分，这样资源占用小，能够在单机环境下与KVM之类的虚拟化方案相比能够更加快速和占用更少资源

原理：

Docker核心解决的问题是利用LXC来实现类似VM的功能，从而利用更加节省的硬件资源提供给用户更多的计算资源。同VM的方式不同, [LXC](http://baike.baidu.com/view/6572152.htm) 其并不是一套硬件虚拟化方法 - 无法归属到全虚拟化、部分虚拟化和半虚拟化中的任意一个，而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。所以我们从虚拟化到docker要解决的问题出发，看看他是怎么满足用户虚拟化需求的。

Linux Namespace (ns)

LXC所实现的隔离性主要是来自kernel的namespace, 其中pid, net, ipc, mnt, uts 等namespace将container的进程, 网络, 消息, 文件系统和hostname 隔离开。