Docker 快速入门

1. Docker网址： [www.docker.com](http://www.docker.com)
2. 什么是Docker

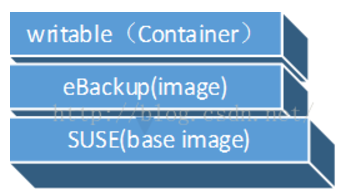
Docker is the world’s leading software container platform.

Docker是Docker公司开源的一个基于轻量级虚拟化技术的应用容器引擎项目，可以让开发者打包应用及其依赖到一个可移植的容器中，然后发布到任何流行的Linux机器上。

2.1 镜像

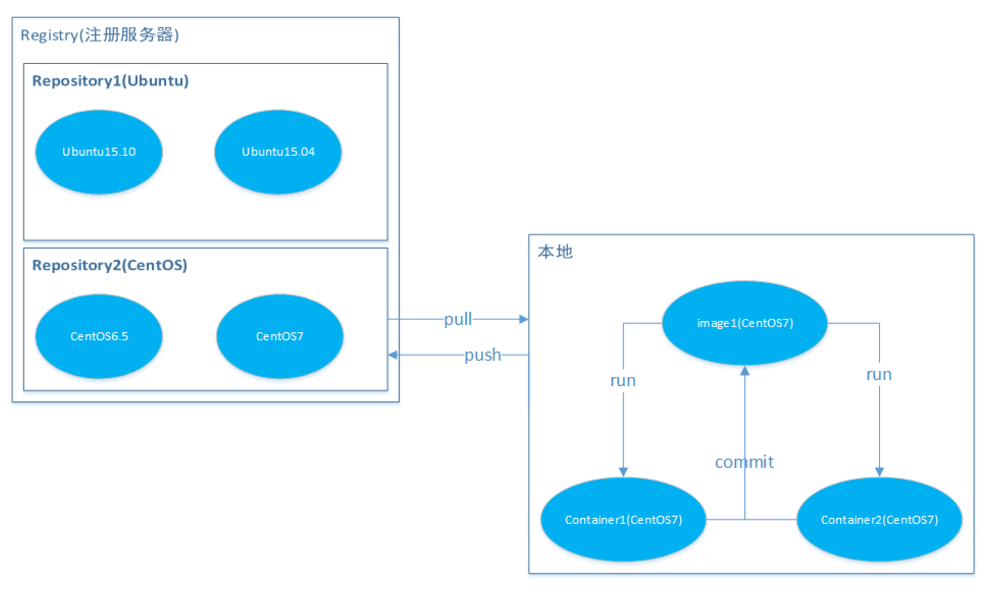
是一个只读的模板，一个独立的文件系统，包括运行容器中所需的数据，可以用来创建新的容器。

镜像=操作系统+软件运行环境+用户程序



2.2 仓库

用来保存Docker镜像，Docker Hub.



2.3 容器

Docker容器是由Docker镜像创建的运行实例。每个容器间是相互隔离的，容器中会运行特定的应用，包含特定应用的代码及所需的依赖文件。

可以把容器看作是一个简易版的 Linux 环境（包括root用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序。

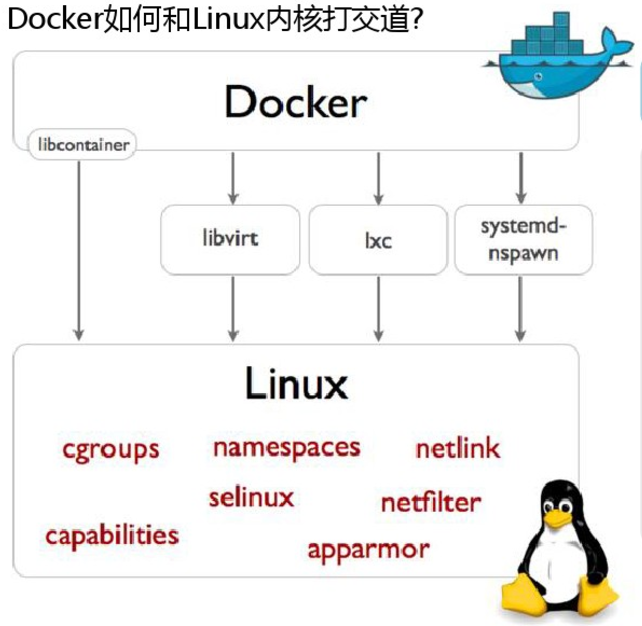
1. 尝试解决的问题：

* 资源利用率低
* 单物理机多应用无法有效隔离（进程空间，CPU资源，磁盘等）
* 运维部署不便
* 测试，版本管理复杂
* 迁移成本高
* 传统虚拟机，空间占用大，启动慢，管理复杂

1. 技术优点：

* 秒级启停，空间资源占用极少（几M）
* 实现进程级别的隔离
* 可在普通的服务器上建立上百至上千个Docker实例
* 加快开发测试部署的速度
* 简化版本管理

1. Docker与Linux



Docker基于Linux容器技术（LXC），Namespace，Cgroup，UnionFS（联合文件系统）等技术实现：

**namespace（命名空间）**：命名空间是 Linux 内核一个强大的特性。每个容器都有自己单独的命名空间，运行在其中的应用都像是在独立的操作系统中运行一样。命名空间保证了容器之间彼此互不影响。docker实际上一个进程容器，它通过namespace实现了进程和进程所使用的资源的隔离。使不同的进程之间彼此不可见。

Docker用到的一些命名空间有：

pid命名空间：用于隔离进程，容器都有自己独立的进程表和1号进程；

net命名空间：用于管理网络，容器有自己独立的networkinfo；

ipc命名空间：用于访问IPC资源（IPC:InterProcess Communication）；

mnt命名空间：用于管理挂载点，每个容器都有自己唯一的目录挂载；

uts命名空间：用于隔离内核和版本标识（UTS:UnixTimeProcess System），每个容器都有独立的hostname和domain。

**cgroup（控制组）**：是 Linux 内核的一个特性，主要用来对共享资源进行隔离、限制、审计等。只有能控制分配到容器的资源，才能避免当多个容器同时运行时的对系统资源的竞争。控制组技术最早是由 Google 的程序员 2006 年起提出，Linux 内核自 2.6.24 开始支持。控制组可以提供对容器的内存、CPU、磁盘 IO 等资源的限制和审计管理。

**UnionFS（联合文件系统）**：Union文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对 文件系统的修改作为一次提交来一层层的叠加，同时可以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)。Union 文件系统是 Docker 镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。另外，不同 Docker 容器就可以共享一些基础的文件系统层，同时再加上自己独有的改动层，大大提高了存储的效率。Docker 中使用的 AUFS（AnotherUnionFS）就是一种 Union FS。 AUFS 支持为每一个成员目录（类似 Git 的分支）设定只读（readonly）、读写（readwrite）和写出（whiteout-able）权限, 同时 AUFS 里有一个类似分层的概念, 对只读权限的分支可以逻辑上进行增量地修改(不影响只读部分的)。Docker 目前支持的 Union 文件系统种类包括 AUFS, btrfs, vfs 和 DeviceMapper。

1. 虚拟机与容器的对比



1. 基本命令使用：

* 查找镜像： docker search
* 获取镜像： docker pull
* 查看本地镜像： docker images
* 创建容器： docker run -dit
* 连接容器： docker attach / docker exec
* 停止容器： docker stop
* 查看容器： docker ps -a
* 提交新镜像： docker commit
* 导出镜像： docker save
* 导入镜像：docker load
* 删除容器： docker rm
* 删除镜像： docker rmi images

1. 应用场景

### 简化配置

这是Docker公司宣传的Docker的主要使用场景。虚拟机的最大好处是能在你的硬件设施上运行各种配置不一样的平台（软件、系统），Docker在降低额外开销的情况下提供了同样的功能。它能让你将运行环境和配置放在代码中然后部署，同一个Docker的配置可以在不同的环境中使用，这样就降低了硬件要求和应用环境之间耦合度。

### 代码流水线（Code Pipeline）管理

前一个场景对于管理代码的流水线起到了很大的帮助。代码从开发者的机器到最终在生产环境上的部署，需要经过很多的中间环境。而每一个中间环境都有自己微小的差别，Docker给应用提供了一个从开发到上线均一致的环境，让代码的流水线变得简单不少。

### 提高开发效率

这就带来了一些额外的好处：Docker能提升开发者的开发效率。如果你想看一个详细一点的例子，可以参考Aater在[DevOpsDays Austin 2014](http://www.slideshare.net/Flux7Labs/using-docker-to-improve-web-developer-productivity-dev-opsdays-austin-may-5)大会或者是DockerCon上的演讲。  
  
不同的开发环境中，我们都想把两件事做好。一是我们想让开发环境尽量贴近生产环境，二是我们想快速搭建开发环境。  
  
理想状态中，要达到第一个目标，我们需要将每一个服务都跑在独立的虚拟机中以便监控生产环境中服务的运行状态。然而，我们却不想每次都需要网络连接，每次重新编译的时候远程连接上去特别麻烦。这就是Docker做的特别好的地方，开发环境的机器通常内存比较小，之前使用虚拟的时候，我们经常需要为开发环境的机器加内存，而现在Docker可以轻易的让几十个服务在Docker中跑起来。

### 隔离应用

有很多种原因会让你选择在一个机器上运行不同的应用，比如之前提到的提高开发效率的场景等。  
我们经常需要考虑两点，一是因为要降低成本而进行服务器整合，二是将一个整体式的应用拆分成松耦合的单个服务（译者注：微服务架构）。

### 整合服务器

正如通过虚拟机来整合多个应用，Docker隔离应用的能力使得Docker可以整合多个服务器以降低成本。由于没有多个操作系统的内存占用，以及能在多个实例之间共享没有使用的内存，Docker可以比虚拟机提供更好的服务器整合解决方案。

### 调试能力

Docker提供了很多的工具，这些工具不一定只是针对容器，但是却适用于容器。它们提供了很多的功能，包括可以为容器设置检查点、设置版本和查看两个容器之间的差别，这些特性可以帮助调试Bug。

### 多租户环境

另外一个Docker有意思的使用场景是在多租户的应用中，它可以避免关键应用的重写。我们一个特别的关于这个场景的例子是为IoT（译者注：物联网）的应用开发一个快速、易用的多租户环境。这种多租户的基本代码非常复杂，很难处理，重新规划这样一个应用不但消耗时间，也浪费金钱。  
  
使用Docker，可以为每一个租户的应用层的多个实例创建隔离的环境，这不仅简单而且成本低廉，当然这一切得益于Docker环境的启动速度和其高效的diff命令。

### 快速部署

在虚拟机之前，引入新的硬件资源需要消耗几天的时间。虚拟化技术（Virtualization）将这个时间缩短到了分钟级别。而Docker通过为进程仅仅创建一个容器而无需启动一个操作系统，再次将这个过程缩短到了秒级。这正是Google和Facebook都看重的特性。  
  
你可以在数据中心创建销毁资源而无需担心重新启动带来的开销。通常数据中心的资源利用率只有30%，通过使用Docker并进行有效的资源分配可以提高资源的利用率。