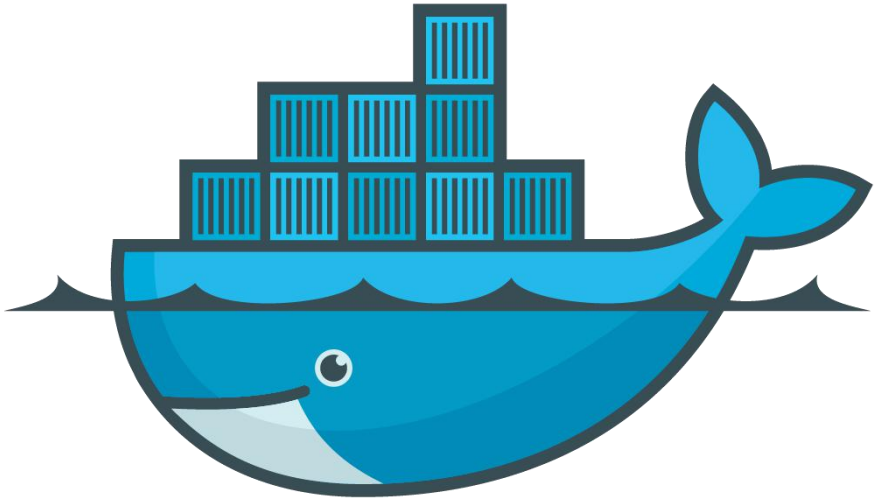
Docker 介绍



一、Docker案例介绍
二、Docker概念介绍
三、Docker与虚拟机对比
四、Docker应用场景
五、Docker的实用性
六、Docker基本原理
七、Docker基本命令介绍
八、Docker调度工具介绍
九、Docker最佳实践

一、Docker案例介绍

眼见为实——邮件服务器的搭建：

```
docker run −d −−name mail −−restart=always −p 443:443 −p
8081:80 −p 25:25 −p 110:110 −p587:587 −p995:995 −P −h
mail.example.com cema/iredmail:example  init 2
```

访问：
邮箱: https://localhost
管理后台：https://localhost/iredadmin
With login: postmaster@example.com
And password: password_pm

编辑(E) 查看(V) 历史(S) 书签(B) 工具(T) 帮助(H)

慧基础云 - 云端… × | (18) Roundcube… × | 智慧基础云BOSS × | Roundcube Webm… × | +

https://192.168.1.170/mail/ | 搜索

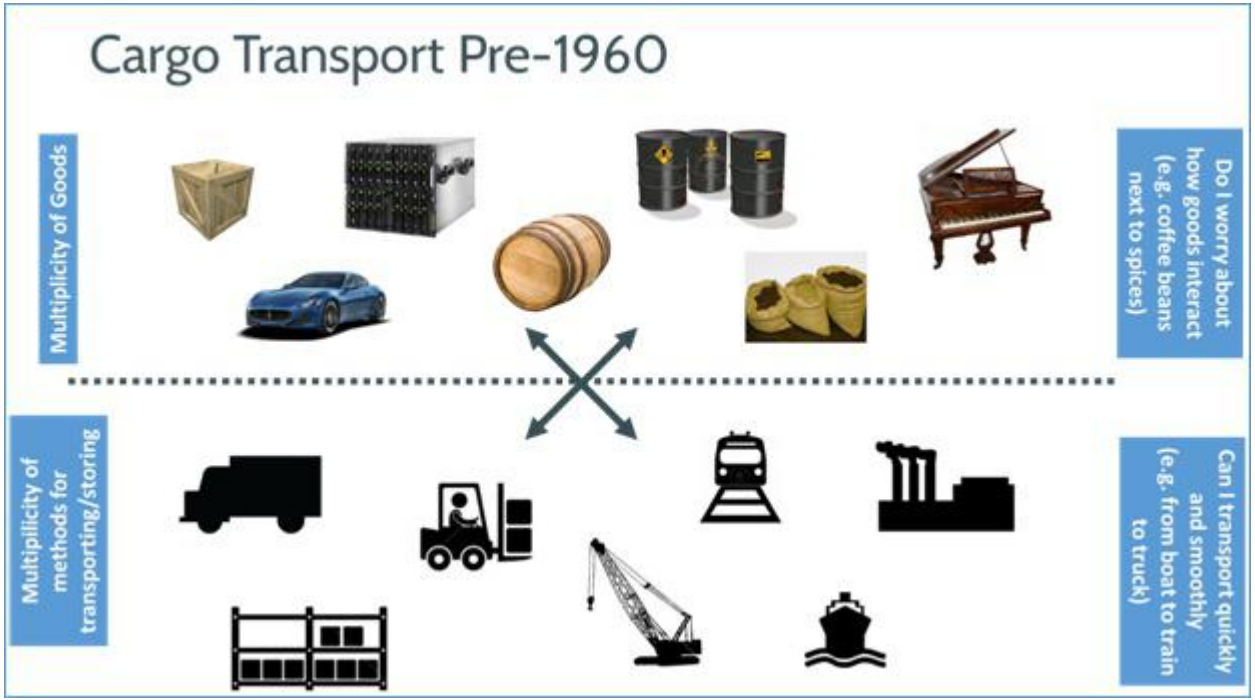
一下，你就知道 | Roundcube Webmail | Onecloud::ITSM | OA | small标题 | 鸿瑞云计算平台 | powerone | Amazon Web Servic… | ali云服务器 | 博客园

Roundcube Webmail

二、Docker概念介绍

什么是容器?
什么是Docker?

1960年之前的货运情况



通过集装箱的运货方式(1960年之后)

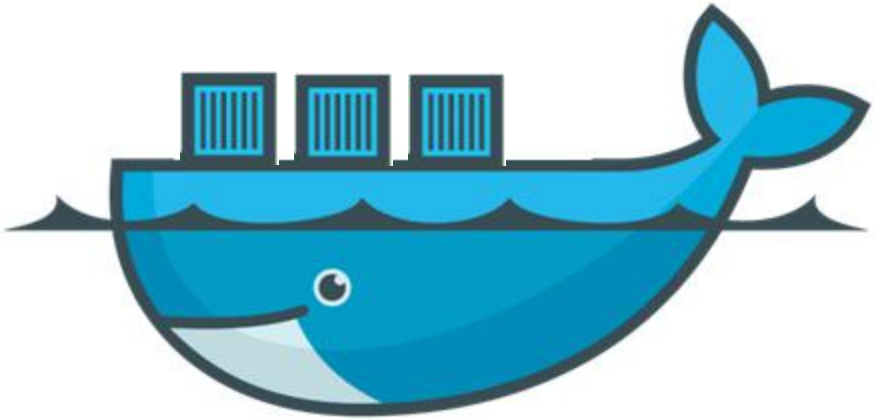
- Docker基于容器技术的轻量级虚拟化解决方案
- Docker是容器引擎，把Linux的cgroup、namespace等容器底层技术进行封装抽象，为用户提供了创建和管理容器的便捷界面（包括命令行和API）
- Docker 是一个开源项目，诞生于 2013 年初，基于 Google 公司推出的 Go 语言实现
- 微软，红帽Linux，IBM，Oracle等主流IT厂商已经在自己的产品里增加对Docker的支持。
- Google 每周启动超过20亿个容器进行业务服务，于上个世纪90年代已经开始大规模使用容器技术

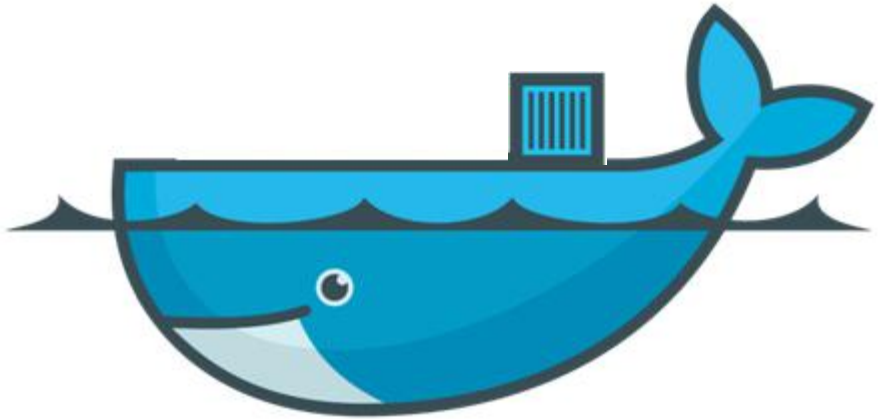
Docker主要功能特征

	物理容器	Docker
内容无关性	相同的集装箱可以容纳几乎任何类型的货物	可以封装任何有效负载及其依赖项
硬件无关性	同一标准的容器允许把货物从船上运输到火车、卡车上，直到运输到仓库，整个过程无需整理货物或打开容器	使用操作系统基元（例如：LXC）几乎可以在任何平台上运行——虚拟机、裸机、OpenStack、公共IaaS等，并且无需修改
内容隔离和交互	无需担心铁压在香蕉上，容器可以堆积运输	资源、网络和内容隔离，避免依赖

	物理容器	Docker
自动化	标准的接口使其易于实现自动化装卸、搬运等	运行、启动、停止、提交、搜索等都有标准的操作，非常适合devops：CI、CD、自动扩展、混合云
高效	无需打开或修改，可以在起始两地快速地移动/运输	轻量级，可以进行快速移动和操作
职责分离	托运人担心盒子内部、承运人担心盒子外部	开发人员担心代码，运营人员担心基础设施

docker容器,软件运行的单元（例如tomcat、mysql软件）



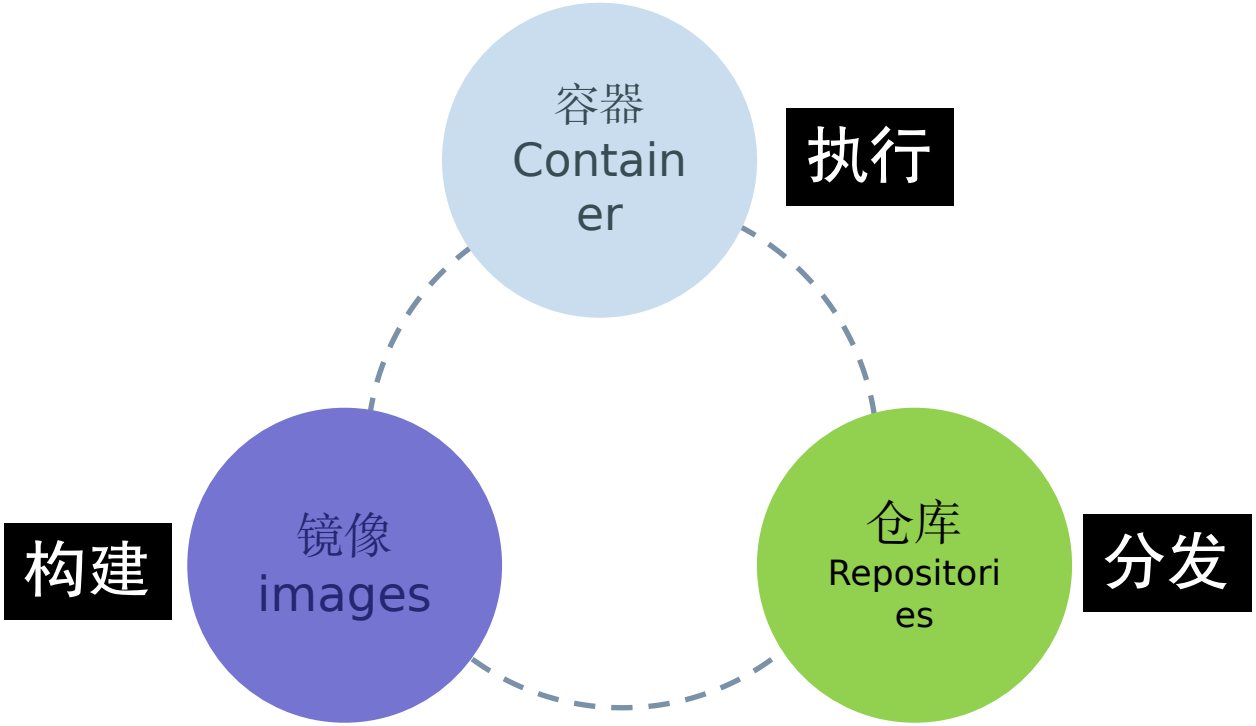


容器介绍

Container（容器）技术其实早在多年前就出现了。从2005年的Solaris Containers到2008年LXC 0.1版本的推出。再到后来的Google推出开源的容器管理工具lmctfy，也将近经历了10年的发展。它仅仅只是一个虚拟化的技术，相比KVM，XEM没有太多的优势。

直到2013年，Docker的出现。才代表着容器技术一个新的时代的来临。

从技术角度看，传统容器只解决了容器执行（run）问题，而Docker定义了一套容器构建（build）分发（ship）执行（run）



Client

Docker pull

Docker run

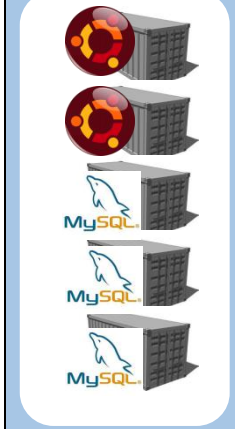
Docker build

Docker push

Docker_host1

Docker engine

容器



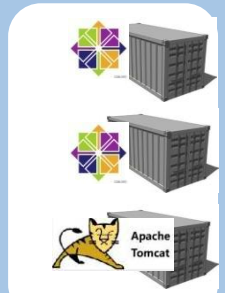
镜像

Registry(仓库)

Docker_host 2

Docker engine

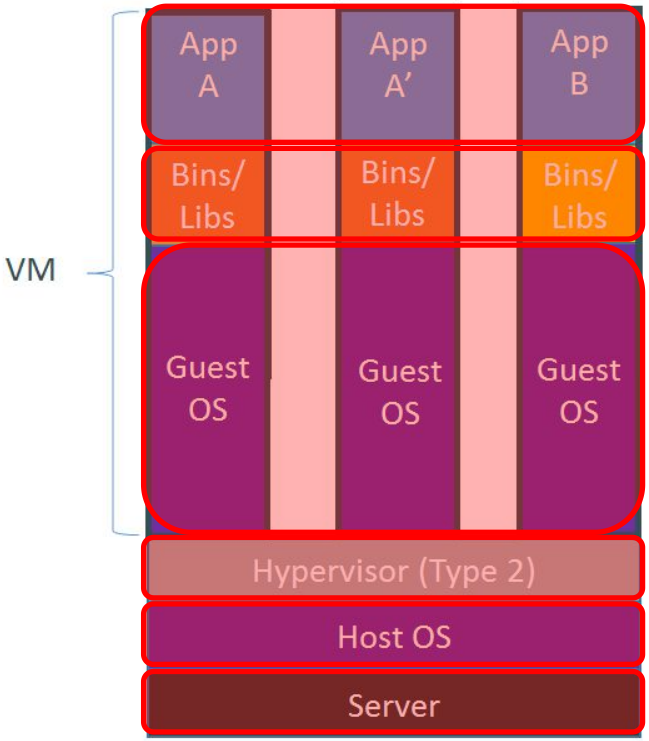
容器



镜像

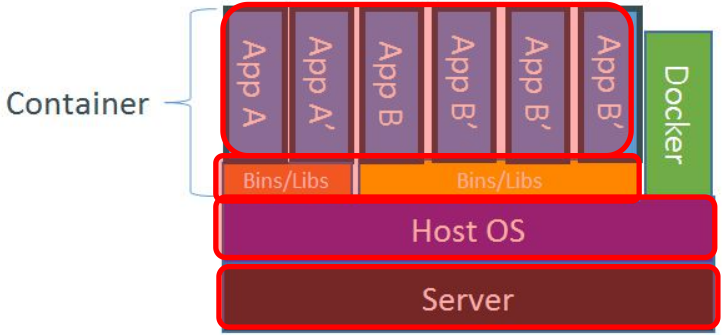
三、Docker与虚拟机对比

虚拟机和Docker有什么区别?



Containers are isolated, but share OS and, where appropriate, bins/libraries

…result is significantly faster deployment, much less overhead, easier migration, faster restart



Docker--轻量级虚拟化容器技术

作为一种轻量级的虚拟化方式，Docker在运行应用上跟传统的虚拟机方式相比具有显著优势：

Docker容器很快，启动和停止可以在秒级实现，这相比传统的虚拟机方式要快得多。

Docker容器对系统资源需求很少，一台主机上可以同时运行数千个Docker容器。

Docker通过类似Git的操作来方便用户获取、分发和更新应用镜像，指令简明，学习成本较低。

Docker--轻量级虚拟化容器技术

Docker通过Dockerfile配置文件来支持灵活的自动化创建和部署机制，提高工作效率。

Docker容器除了运行其中的应用之外，基本不消耗额外的系统资源，保证应用性能的同时，尽量减小系统开销。传统虚拟机方式运行N个不同的应用就要启动N个虚拟机（每个虚拟机需要单独分配独占的内存、磁盘等资源），而Docker只需要启动N个隔离的容器，并将应用放到容器内即可。

特点对比

	Docker	虚拟机
启动速度	秒级	分钟级
复杂度	基于内核的namespace技术，对现有基础设施的侵入较少	部署复杂度较高，并且很多基础设施不兼容
执行性能	在内核中实现，所以性能几乎与原生一致	对比内核级实现，性能较差
可控性	依赖简单，与进程无本质区别	依赖复杂，并且存在跨部门问题
体积	与业务代码发布版本大小相当MB级别	GB级别
并发性	可以启动几百几千个容器	最多几十个虚拟机
资源利用率	高	低

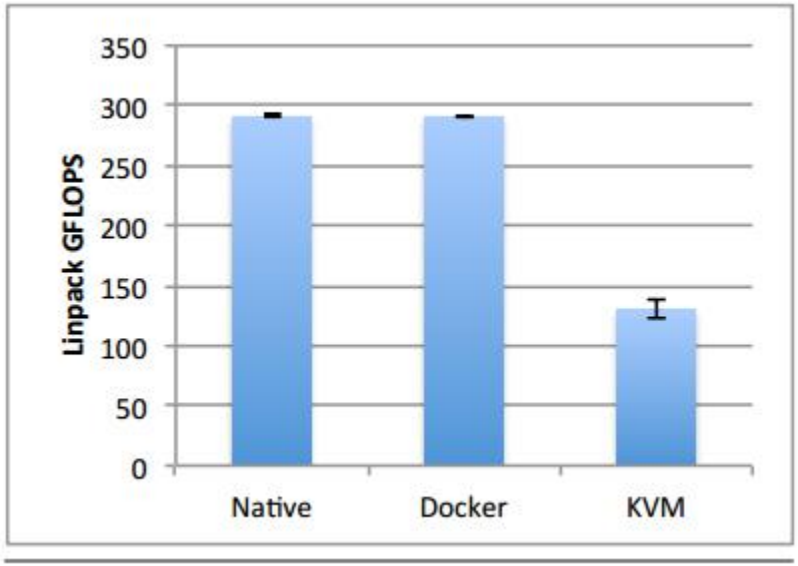
性能对比

以下的数据均是在IBM x3650 M4服务器测得，其主要的硬件参数是：

（1）2颗英特尔xeon E5-2655处理器，主频2.4-3.0 GHz。每颗处理器有8个核，因此总共有16个核。

（2）256 GB RAM.

在测试中是通过运算Linpack程序来获得计算能力数据的。结果如下图所示：



Docker相对虚拟机不足之处

1.资源隔离方面不如虚拟机，docker是利用cgroup实现资源限制的，只能限制资源消耗的最大值，而不能隔绝其他程序占用自己的资源。
2.安全性问题。docker目前并不能分辨具体执行指令的用户，只要一个用户拥有执行docker的权限，那么他就可以对docker的容器进行所有操作，不管该容器是否是由该用户创建。比如A和B都拥有执行docker的权限，由于docker的server端并不会具体判断docker client是由哪个用户发起的，A可以删除B创建的容器，存在一定的安全风险。
3.docker目前还在版本的快速更新中，细节功能调整比较大。一些核心模块依赖于高版本内核，存在版本兼容问题

结论

有些激进的言论声称Docker将是现有虚拟机技术的终结者，个人觉得此言论有些浮夸了。Docker是面向应用的，其终极目标是构建PAAS平台，而现有虚拟机主要目的是提供一个灵活的计算资源池，是面向架构的，其终极目标是构建一个IAAS，或者是SDDC(Software Defined Data Center软件定义的数据中心)。

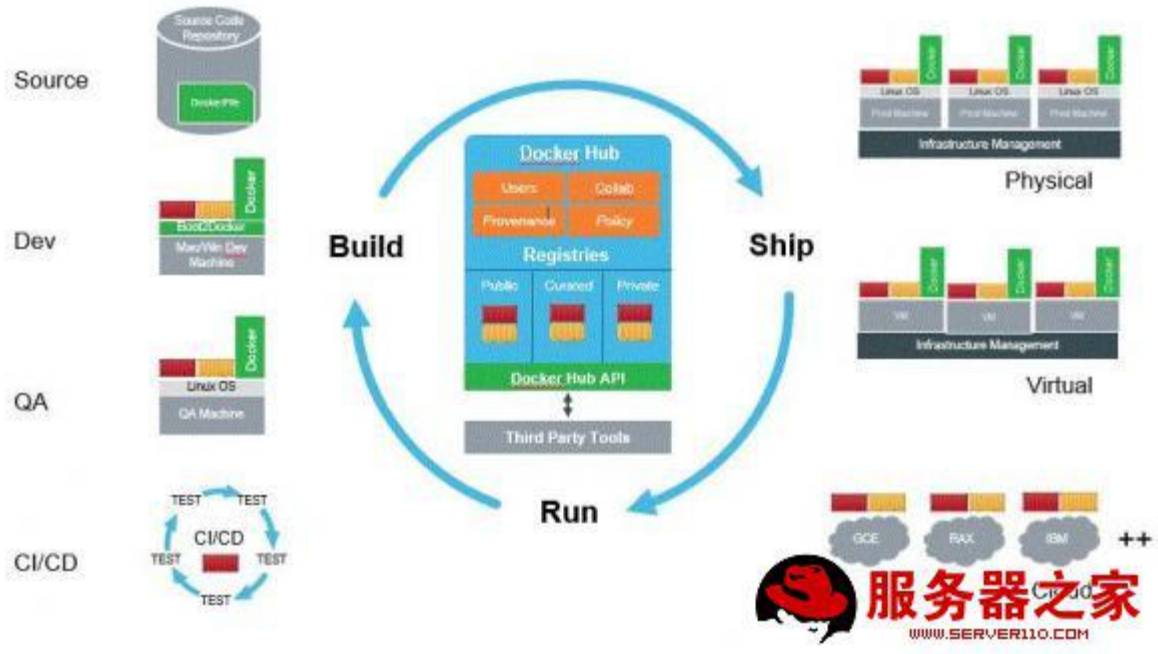
并且，两者相辅相成。Docker的老东家dotCloud的PAAS服务便基于Amazon的AWS服务，因此，虚拟机是Docker的土壤，而Docker则向用户展现了业务。

四、Docker应用场景

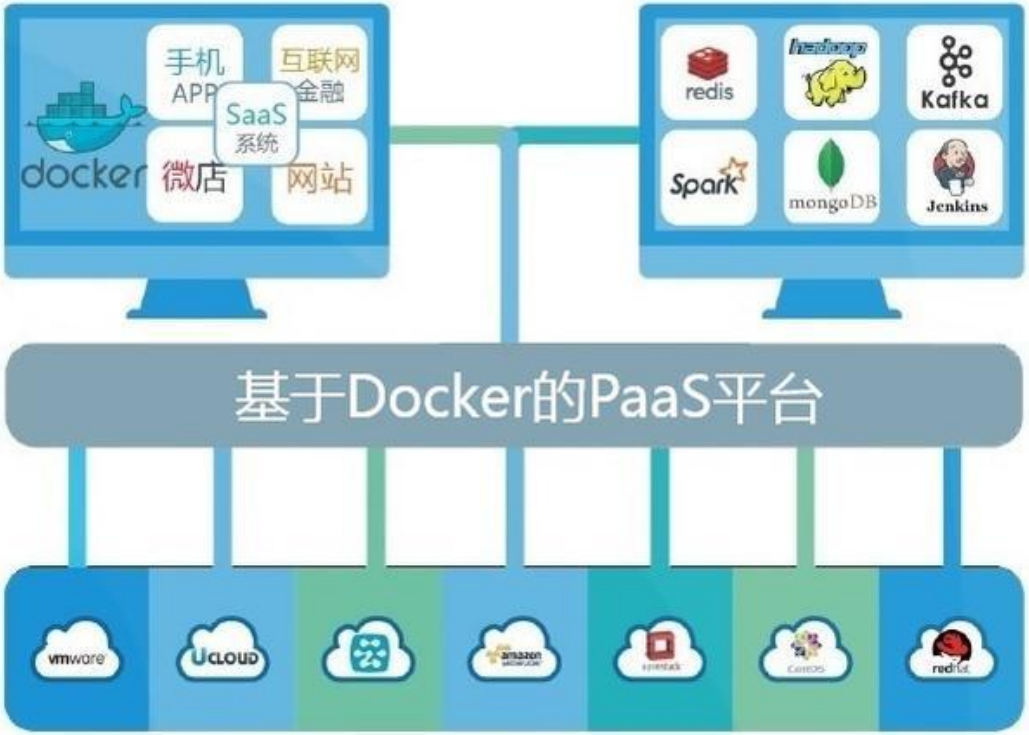
对应用进行自动打包和部署
创建轻量、私有的**PAAS**环境
自动化测试和持续整合与部署
部署和扩展**Web**应用、数据库和后端服务

1、对应用进行自动打包和部署(Automating the packaging and deployment of applications)

Docker对于应用依赖封装完整，同一镜像可重复的在测试、集成、生产等环境部署，做到“一次构建，处处运行”，适用于持续集成、持续部署流程。

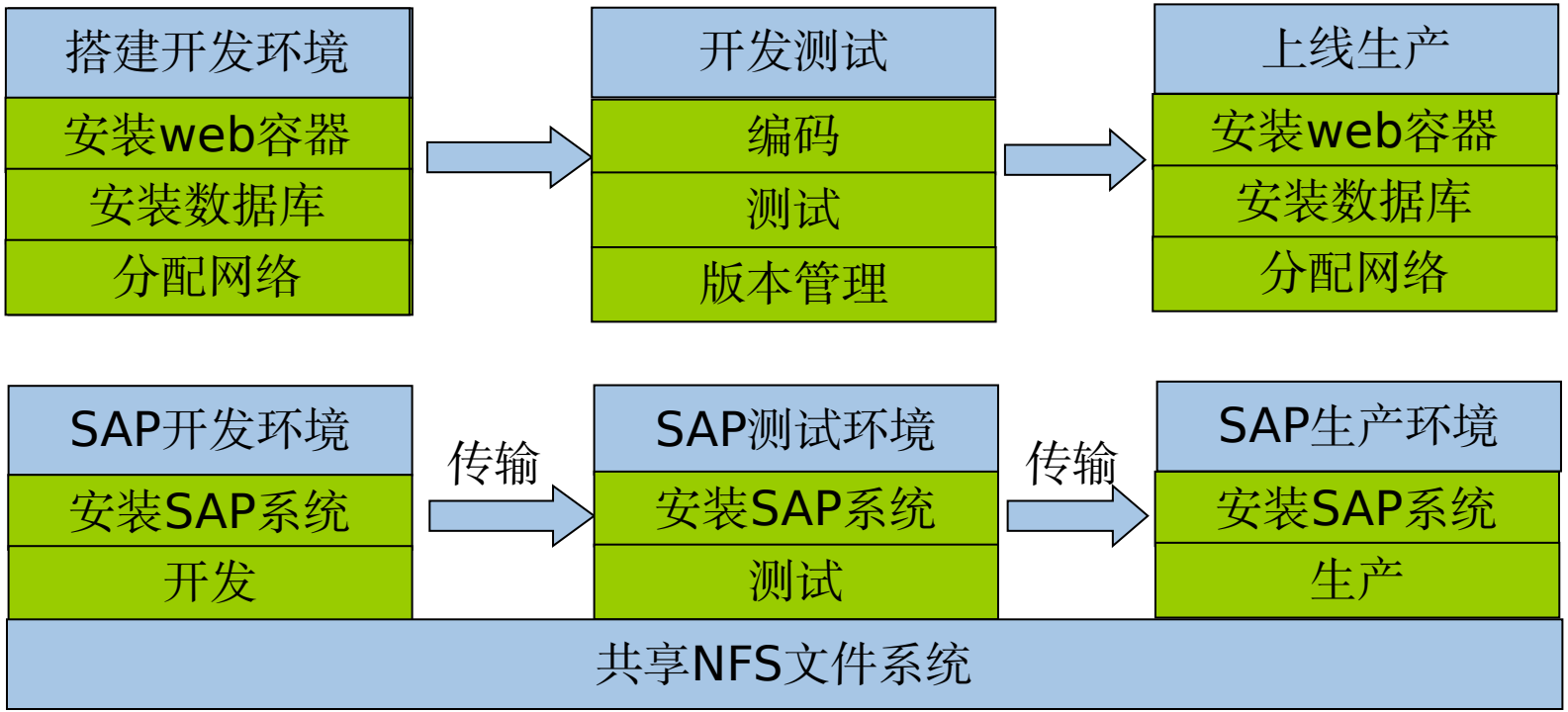


2、创建轻量、私有的PAAS环境(Creation of lightweight, private PAAS environments)



3、自动化测试和持续整合与部署(Automated testing and continuous integration/deployment)

传统型软件开发、测试、上线过程



传统型软件开发、测试、上线过程不足之处

1、资源利用效率低

2、单物理机多应用无法有效隔离（进程空间，cpu资源，磁盘）

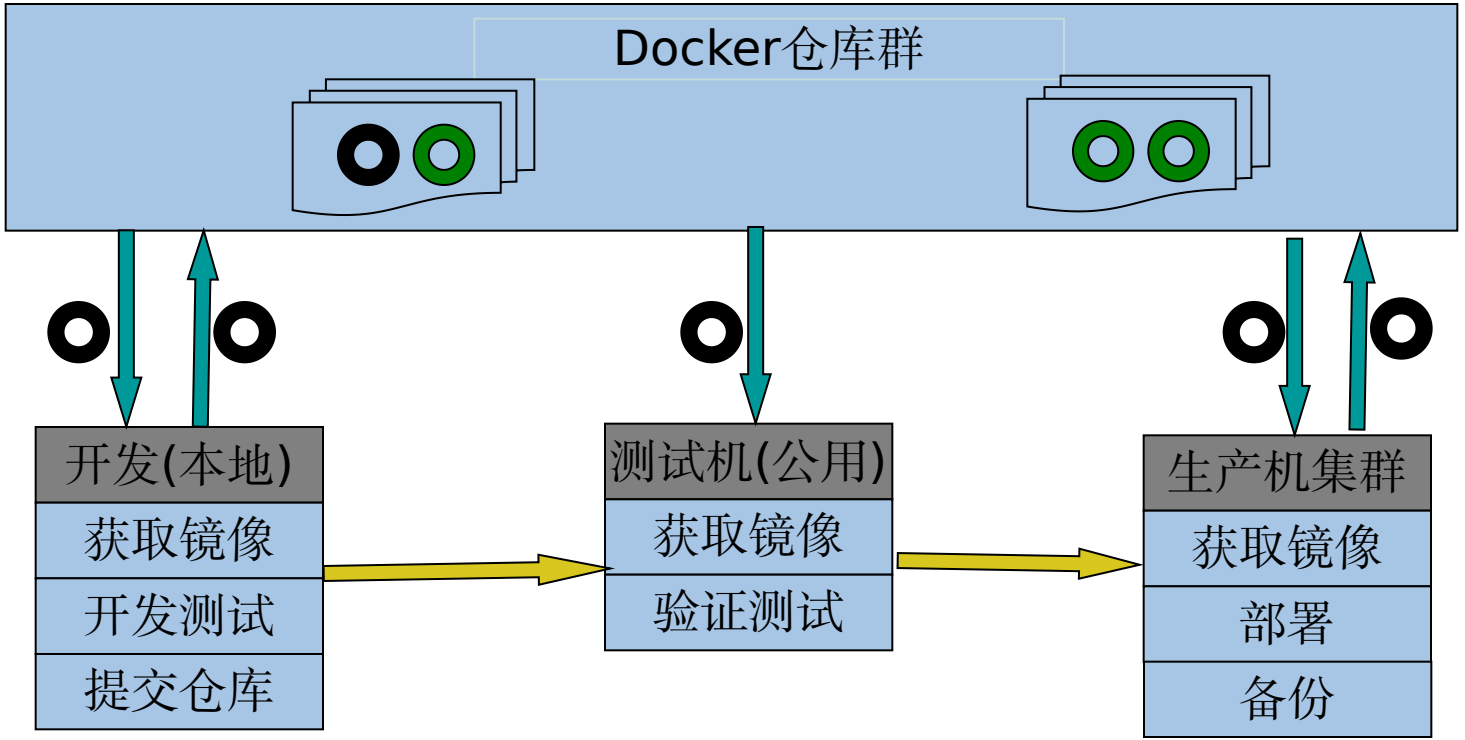
3、运维部署不便

4、测试、版本管理复杂

5、迁移成本高

6、传统虚拟机，空间占用大，启动慢，管理复杂

以Docker为单位的开发部署流程设计



- 以docker为单位的开发测试部署流程,简化了环境搭建的步骤,提高了资源利用效率和开发测试部署的速度,降低了迁移的成本
- 更快速的交付和部署。使用Docker，开发人员可以使用镜像来快速构建一套标准的开发环境；开发完成之后，测试和运维人员可以直接使用相同环境来部署代码。
- Docker可以快速创建和删除容器，实现快速迭代，大量节约开发、测试、部署的时间。并且，各个步骤都有明确的配置和操作，整个过程全程可见，使团队更容易理解应用的创建和工作过程。

4、部署和扩展Web应用、数据库和后端服务(Deploying and scaling web apps, databases and backend services)

案例：微博红包

羊年春晚Docker集群成功的为1.02亿用户刷微博、抢红包提供了可靠的服务。

微博平台Docker集群的规模情况：

Docker集群规模达到1000+节点

QPS（每秒查询率）峰值达到800K/s

4个9的服务SLA达到150ms

共覆盖23个核心服务

春晚共调度近300节点完成动态扩容

结论

由此可见，Docker的目的是让用户用简单的“集装箱”方式，快速的部署大量的、标准化的应用运行环境，所以，只要是这类的需求，Docker都比较适合。

五、Docker的实用性

成熟度、适用性
Docker能在企业里面用么?

1、稳定性
2、可管理性
3、业务高可用和可恢复能力

稳定性

从稳定性上看，Docker在2015年6月10日发布了1.0版本，把该版本称为一个“里程碑”，并声称“1.0的发布表明在质量、功能完整、后台兼容和API 稳定性方面已经提升了一个级别，达到企业 IT 标准”。

但在此之前，dotCloud一直警告用户“不要在生产环境中运行 Docker ”，在RHEL 7中，Docker的版本为0.11.1，这是1.0发布前的RC版本，虽然红帽会将之后的Docker更新和补丁修复更新到0.11版本中，目前Docker的版本是1.9版本。

但是，企业客户在使用这样一个较新的软件版本时，仍需承担不小的稳定性风险的。而在很多企业客户的软件版本选择规范上，都有“需采用已经发布超过半年的稳定版本”的要求。

可管理性

可管理性方面，企业的IT运维人员需要所使用的软件具有很好的可视化管理能力，并且具有可行的监控手段。

Docker目前的集中化管理主要有DockerUI 、Dockland、Shipyard 等Docker的主要作用是应用的发布和运行，但是，看起来Shipyard在Application的管理上还很粗糙，并且，整个管理思路并不是以应用为中心的，这可能会给企业在集中管理Docker的时候，带来了一定的“麻烦”。

而监控的主要目的是快速了解系统、运行的建康状况，对风险状态进行告警，这方面，Docker较为缺乏，还需要企业针对相关环境进行定制化的监控实现。

业务高可用和可恢复性

在企业中任何一个业务都是需要高度可用的，因此，企业业务平台都要考虑三个事情：本地高可用、数据备份、远程灾难恢复。

当然，在使用Docker的时候，也许需要从另一个角度考虑问题，在Docker的应用场景中，提倡“无状态”应用，也就是说，业务数据仅在数据层进行存储，而业务层不关注任何数据。业务层的高可用就可以通过快速的重新部署来实现，数据层仍然采用传统模式，或者借助于传统的方式实现高可用和可恢复性。但这需要时间进行方案摸索和验证，其可行性和可靠性需要时间来去证明。

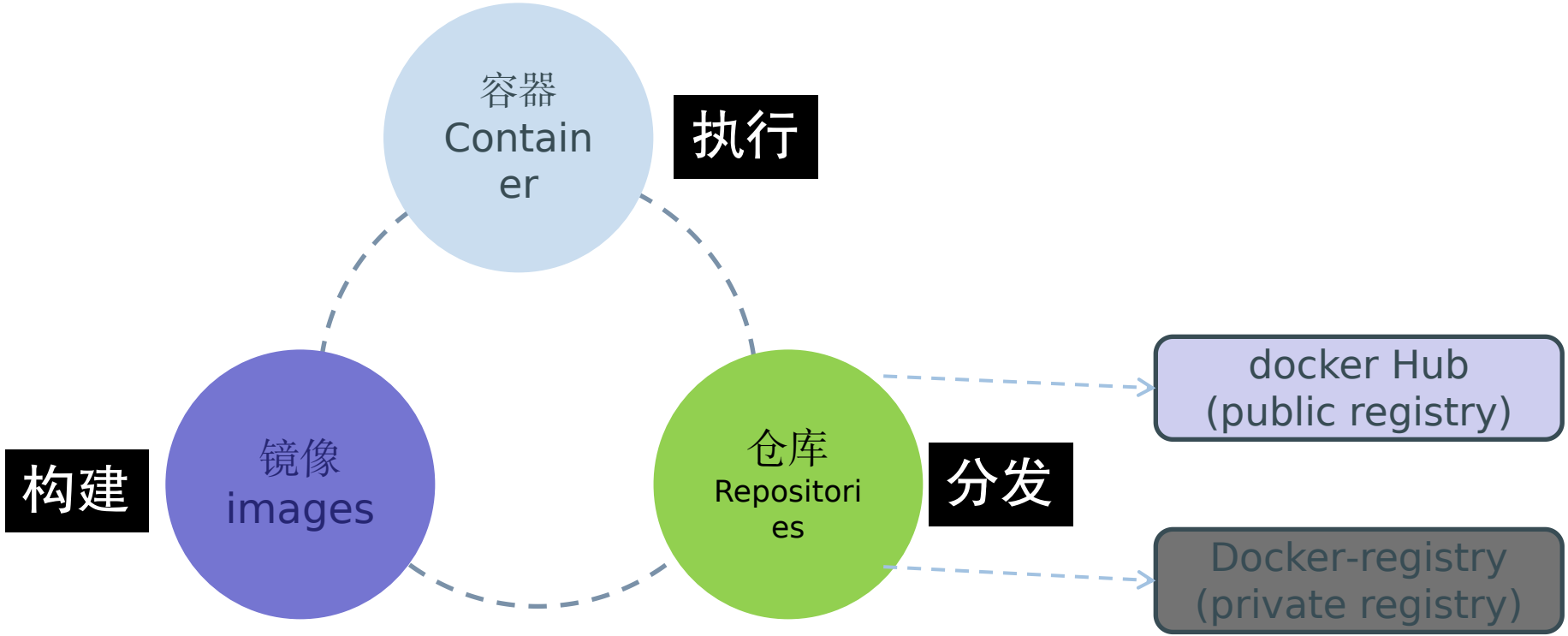
综上所述，Docker到大规模的企业环境应用还有不少的路要走，但是，它所带来的便利性仍然不可小视，这将是革命性的改变，“不足”换种说法就是“机会”，这需要大量了解企业业务的合作伙伴围绕Docker推出相应的解决方案，而Docker的开放性给这种努力带来了极大的便利性。

而对于企业来说，Docker对开发、测试团队带来的便利性非常巨大，而开发、测试环境对之上所讨论到的缺点并不关注，所以，在开发、测试团队大胆的推广、使用Docker无疑能够获得极大的收益。

六、Docker基本原理

基本概念及基本原理

基本概念（三大核心）


容器
Contain
er
执行
镜像
images
构建
仓库
Repositori
es
分发
docker Hub
(public registry)
Docker-registry
(private registry)

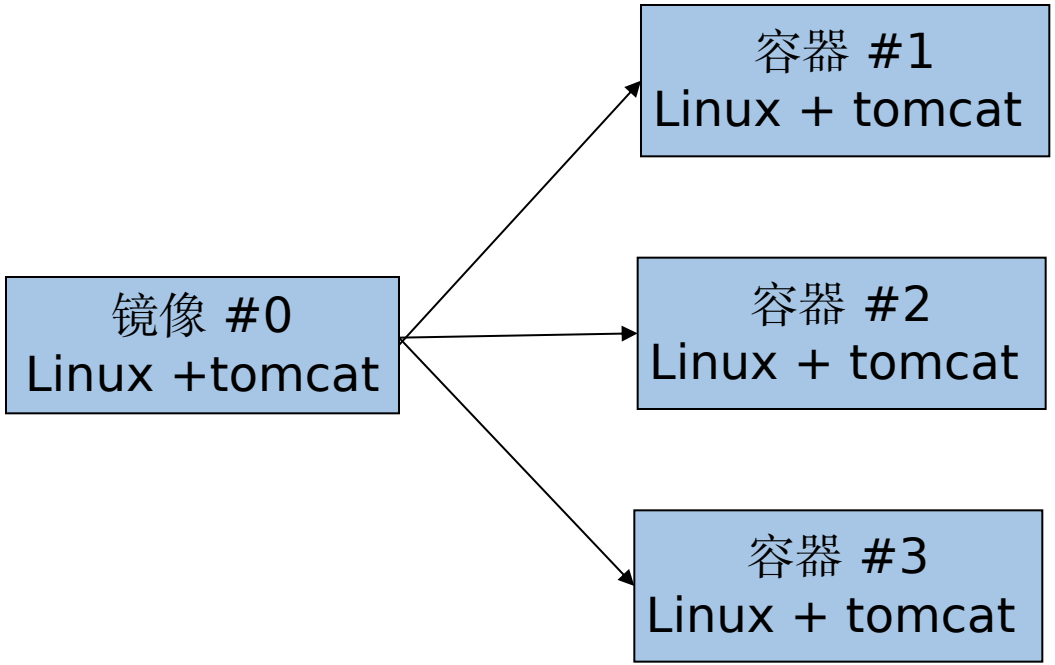
镜像

- Docker 的镜像类似虚拟机的模板，但是更轻量
- 例如：一个镜像可以包含一个完整的 Linux 操作系统环境，里面仅安装了 Tomcat或用户需要的其它应用程序
- 镜像可以用来创建容器

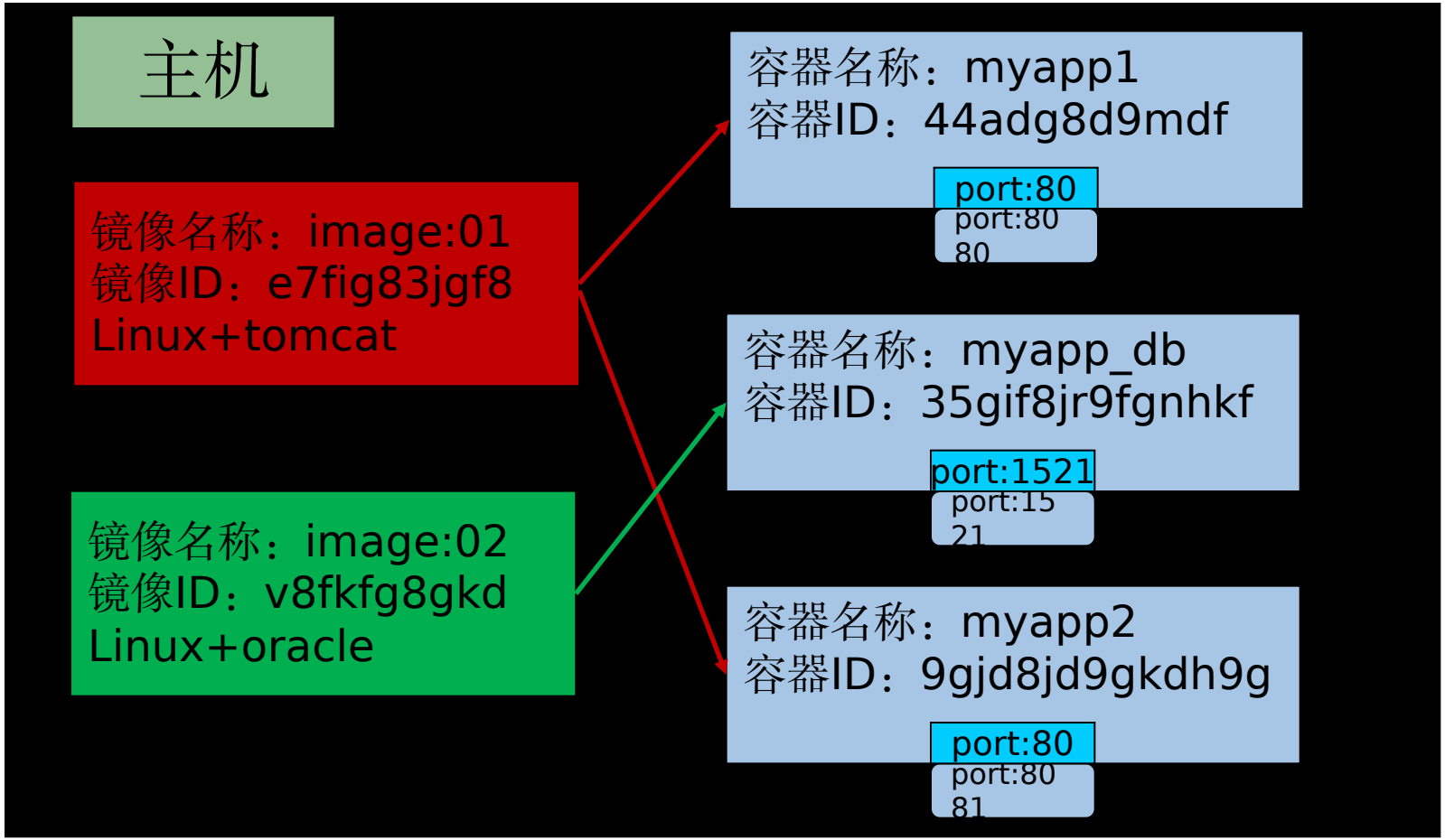
容器

- 等同于从模板中创建虚拟机
- 容器是从镜像创建的运行实例。它可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台。
- 可以把容器看做是一个简易版的 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。

从同一个镜像启动多个容器

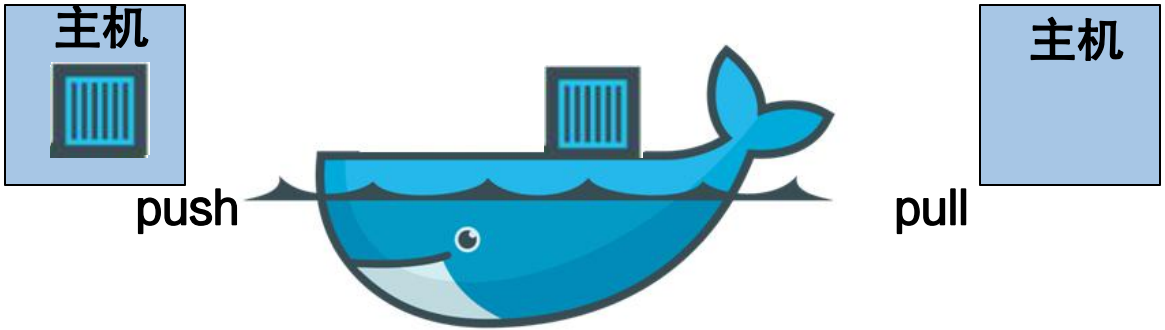


容器端口映射


主机
镜像名称：image:01
镜像ID：e7fig83jgf8
Linux+tomcat
镜像名称：image:02
镜像ID：v8fkfg8gkd
Linux+oracle
容器名称：myapp1
容器ID：44adg8d9mdf
port:80
port:80
80
容器名称：myapp_db
容器ID：35gif8jr9fgnhkf
port:1521
port:15
21
容器名称：myapp2
容器ID：9gjd8jd9gkdh9g
port:80
port:80
81

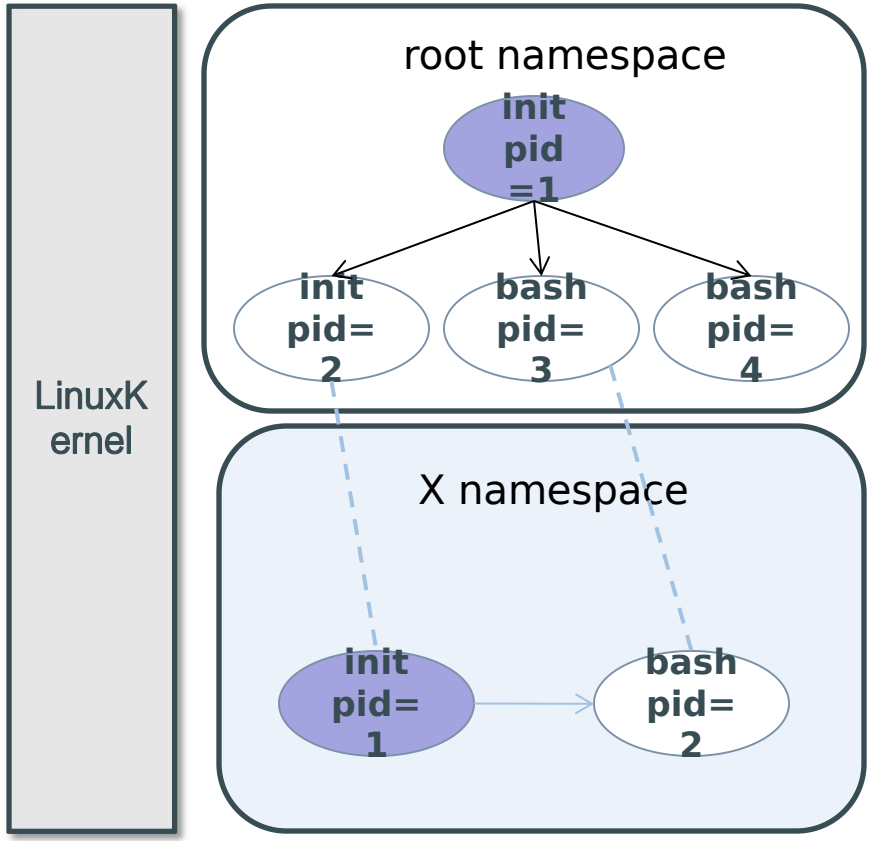
仓库及仓库注册服务器

- 仓库是集中存放镜像文件的场所
- 仓库注册服务器上往往存放着多个仓库，每个仓库中又包含了多个镜像，每个镜像有不同的标签
- 仓库分为公开仓库（Public）和私有仓库（Private）两种形式
- push 镜像到仓库,从仓库pull下镜像



容器Namespace资源隔离

- pid – 进程
- net – 网络
- ipc – 消息
- mnt – 文件系统
- uts – 分时
- user – 用户

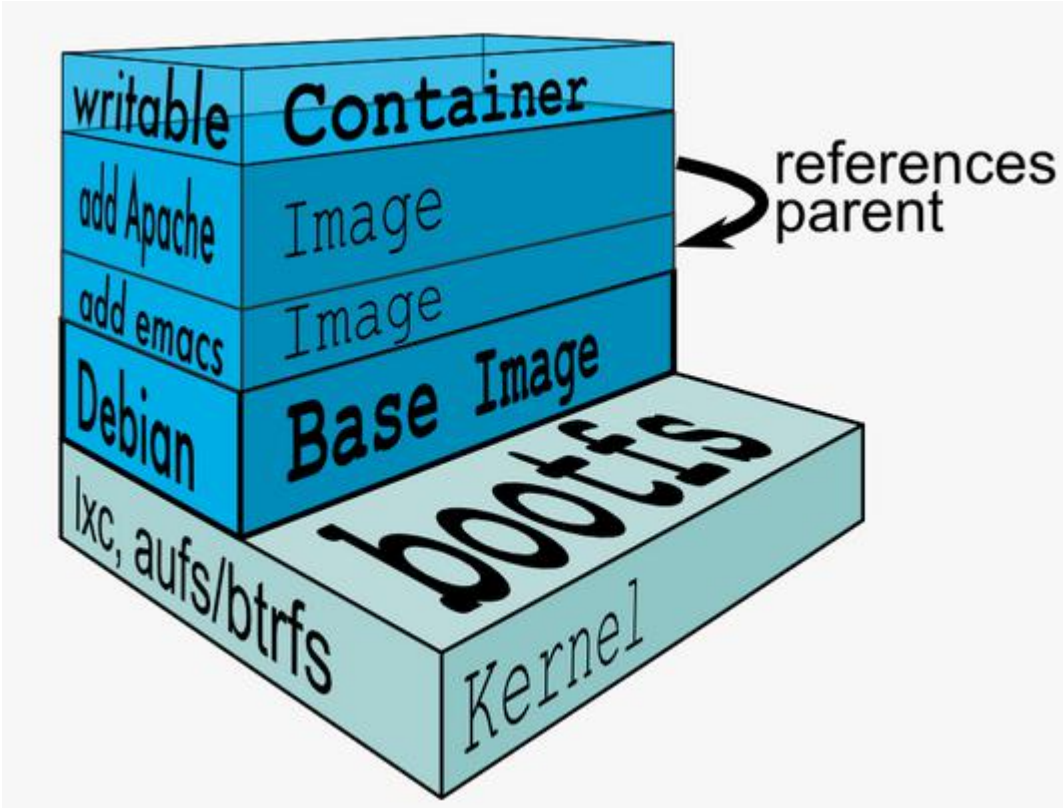


文件系统是如何工作的

Docker镜像被存储在一系列的只读层。当我们开启一个容器，Docker读取只读镜像并添加一个读写层在顶部。如果正在运行的容器修改了现有的文件，该文件将被拷贝出底层的只读层到最顶层的读写层。在读写层中的旧版本文件隐藏于该文件之下，但并没有被不破坏 – 它仍然存在于镜像以下。当Docker的容器被删除，然后重新启动镜像时，将开启一个没有任何更改的新的容器 – 这些更改会丢失。此只读层及在顶部的读写层的组合被Docker称为 Union File System（联合文件系统）。

Docker AUFS特性

- Docker镜像位于bootfs之上
- 每一层镜像的下面一层称为其父镜像(父子关系)
- 第一层镜像为Base Image
- 容器在最顶层
- 其下的所有层都为readonly
- Docker将readonly的FS层称作"image"



七、Docker基本命令介绍

Docker怎么操作?

基本命令

下载image

$docker pull image_name

列出镜像列表;

$docker images

在容器中运行"echo"命令，输出"hello word"

$ docker run image_name **echo** "hello word“

基本命令

列出当前所有正在运行的container

$docker ps

利用dockerfile建立新的镜像

$docker build -t image_name Dockerfile_path

发布docker镜像

$docker **push** new_image_name

操作案例

1、新建dockerfile文件
2、使用dockerfile创建新镜像
3、新建新容器

```
ubuntu@ubuntu:~$ mkdir dockerfile
ubuntu@ubuntu:~$ cd dockerfile
ubuntu@ubuntu:~/dockerfile$ vi dockerfile
```

容器介绍

```
ubuntu@ubuntu:~$ mkdir dockerfile
ubuntu@ubuntu:~$ cd dockerfile
ubuntu@ubuntu:~/dockerfile$ vi dockerfile
```

脚本介绍

基于ubuntu12.04，先来一个更新，然后安装nginx、zip、curl，配置nginx，下载2048代码，解压再放到指定位置，删除原始文件，抛出80端口，最后是执行命令。

```
FROM ubuntu:12.04
RUN apt-get update
RUN apt-get install -y nginx zip curl
RUN echo "daemon off;" >> /etc/nginx/nginx.conf
RUN curl -o /usr/share/nginx/www/master.zip -L
https://codeload.github.com/gabrielecirulli/2048/zip/master
RUN cd /usr/share/nginx/www/ && unzip master.zip && mv 2048-master/* .
&& rm -rf 2048-master master.zip
EXPOSE 80
CMD ["/usr/sbin/nginx", "-c", "/etc/nginx/nginx.conf"]
```

执行脚本

```
ubuntu@ubuntu:~/dockerfile$ sudo docker build -t two-eight .
Sending build context to Docker daemon 2.048 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:12.04
 ---> 5c97af892079
Step 1 : RUN apt-get update
 ---> Using cache
 ---> c327c23fca5c
Step 2 : RUN apt-get install -y nginx zip curl
 ---> Using cache
 ---> 672d58dcb0a3
Step 3 : RUN echo "daemon off;" >> /etc/nginx/nginx.conf
 ---> Using cache
 ---> dc4b8f42854c
```

```
Step 4 : RUN curl -o /usr/share/nginx/www/master.zip -L
https://codeload.github.com/gabrielecirulli/2048/zip/master
 ---> Using cache
 ---> a73692e9d747
Step 5 : RUN cd /usr/share/nginx/www/ && unzip master.zip && mv 2048-
master/* . && rm -rf 2048-master master.zip
 ---> Using cache
 ---> c454b6cfda95
Step 6 : EXPOSE 80
 ---> Using cache
 ---> 90a36ee1a682
Step 7 : CMD /usr/sbin/nginx -c /etc/nginx/nginx.conf
 ---> Using cache
 ---> ab656c34b790
Successfully built ab656c34b790
```

新建容器

```
ubuntu@ubuntu:~/dockerfile$ sudo docker run -d -p 8016:80 two-eight
F85f1d378c3aa3d50a4ae4643ef149f7d6650aa880cf4f3183733427c19333
af
```

查看容器

```
ubuntu@ubuntu:~/dockerfile$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                    NAMES
f85f1d378c3a        two-eight:latest    "/usr/sbin/nginx -c    12 seconds ago      Up 11
seconds       0.0.0.0:8016->80/tcp          happy_brattain
083e0b625f1d        two-eight:v5        "nginx -g 'daemon of   5 days ago          Up 5
days          0.0.0.0:8015->80/tcp          hungry_kirch
05f67d439202        two-eight:v4        "nginx -g 'daemon of   5 days ago          Up 5
days          0.0.0.0:8014->80/tcp          berserk_kirch
e22e3622bc03        mytwo:latest        "/usr/sbin/nginx -c    5 days ago          Up 5
days          0.0.0.0:8013->80/tcp          adoring_lumiere
372f7bf88fce        tomcat7:latest      "/usr/share/tomcat7/   7 weeks ago         Up 7
weeks         9000/tcp, 0.0.0.0:8120->8080/tcp   ecstatic_tesla
46efd96497df        mytwo:latest        "/usr/sbin/nginx -c    11 weeks ago        Up 11
weeks         0.0.0.0:8012->80/tcp          admiring_hodgkin
4d750597e720        mytwo:latest        "/usr/sbin/nginx -c    11 weeks ago        Up 11
weeks         0.0.0.0:8011->80/tcp          goofy_poincare
```

八、Docker调度工具介绍

Docker怎么才能用好?

很多人将Docker等同于Container，其实这是不对的，就像传统的集装箱运输体系一样，集装箱只是其中一个最核心的部件。用它来代表整个以集装箱为核心的运输体系。那么Docker其实就是以容器为核心的IT交付与运行体系。
它包括了Docker Engine（容器的运行管理）
Docker Registry（容器的分发管理）
以及相关的一系列的API接口。
包括Docker Machine，Swarm，Compose。
所以可以看做是一套以容器为核心的创建，分发，和运行的标准化体系。

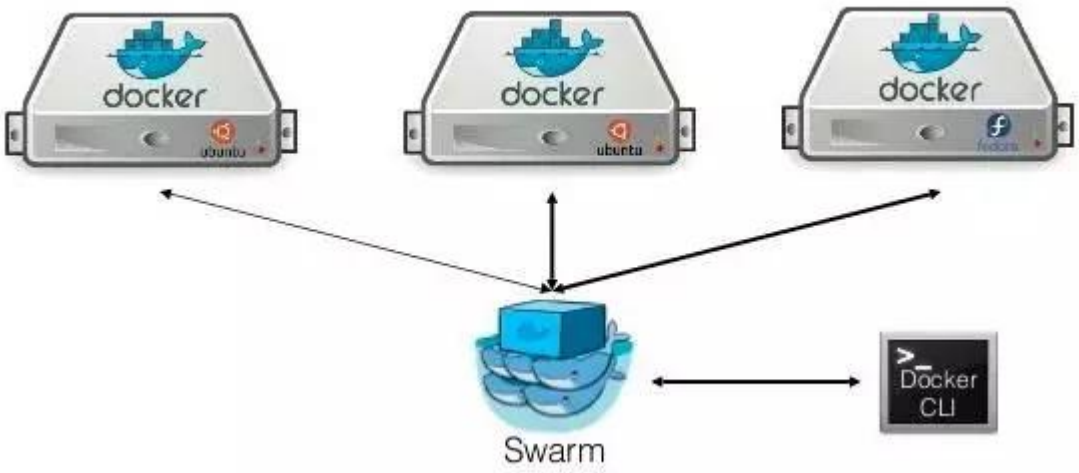
容器调度简介

容器调度工具的主要任务就是负责在最合适的主机上启动容器，并且将它们关联起来。它必须能够通过自动的故障转移（fail-overs）来处理错误，并且当一个实例不足以处理/计算数据时，它能够扩展容器来解决问题。

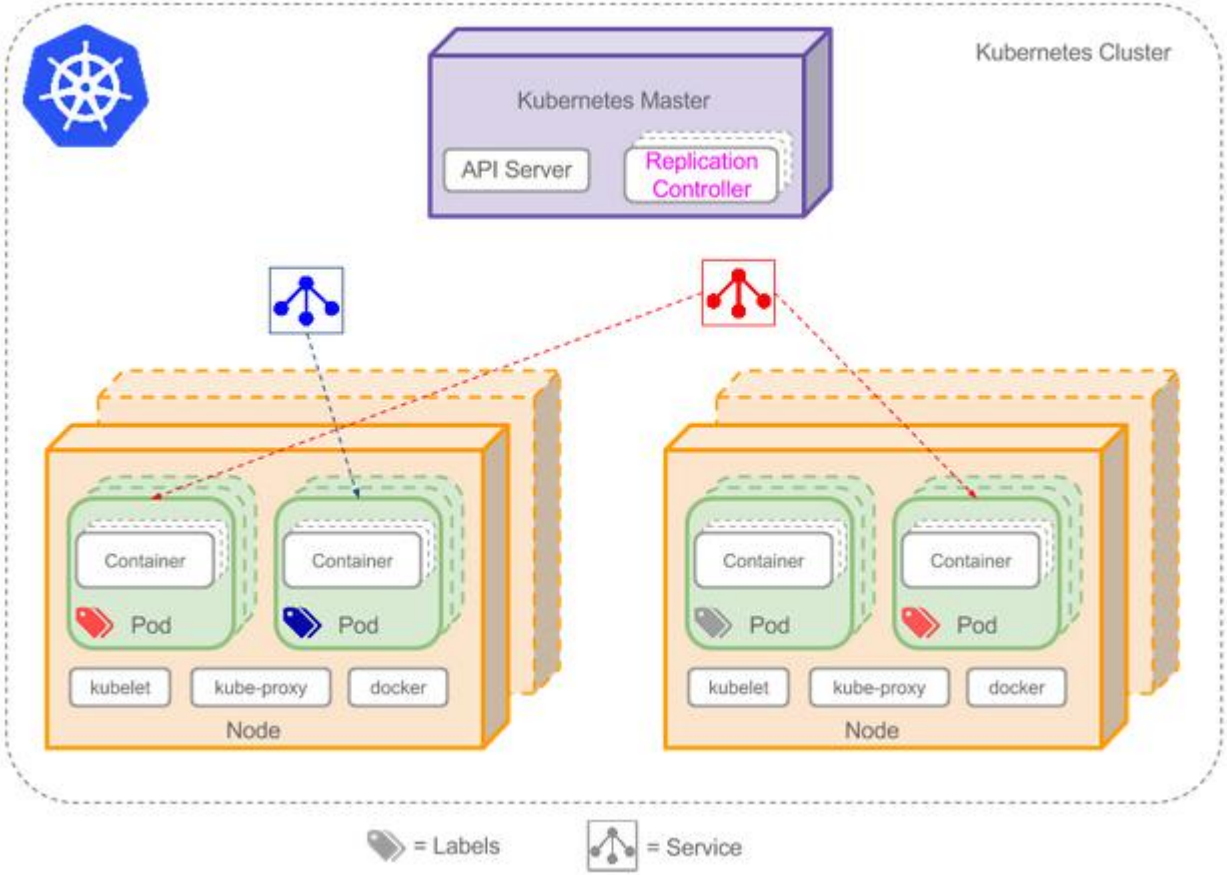
四个主流容器调度框架：Docker Swarm、Apache Mesos（running the Marathon framework）and Google Kubernetes。CoreOS Fleet

Docker Swarm是一个由Docker开发的调度框架。由Docker自身开发的好处之一就是标准Docker API的使用。

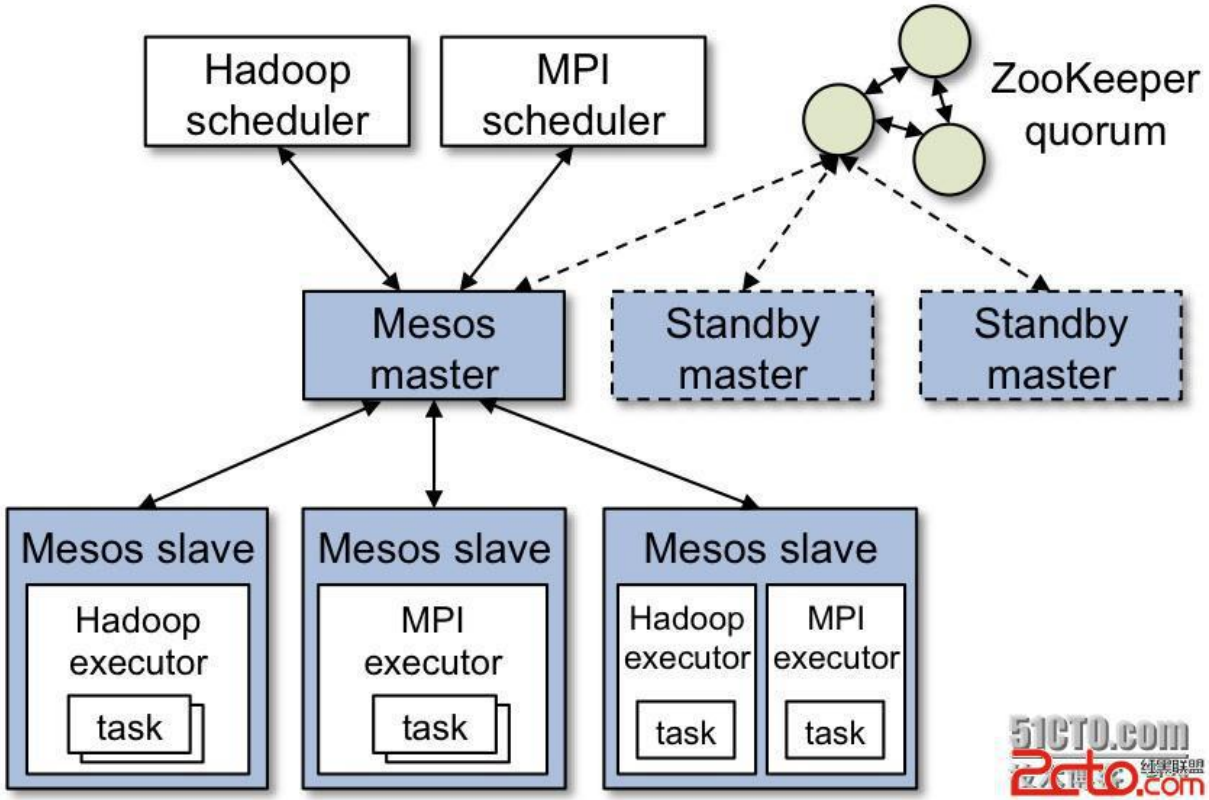
。



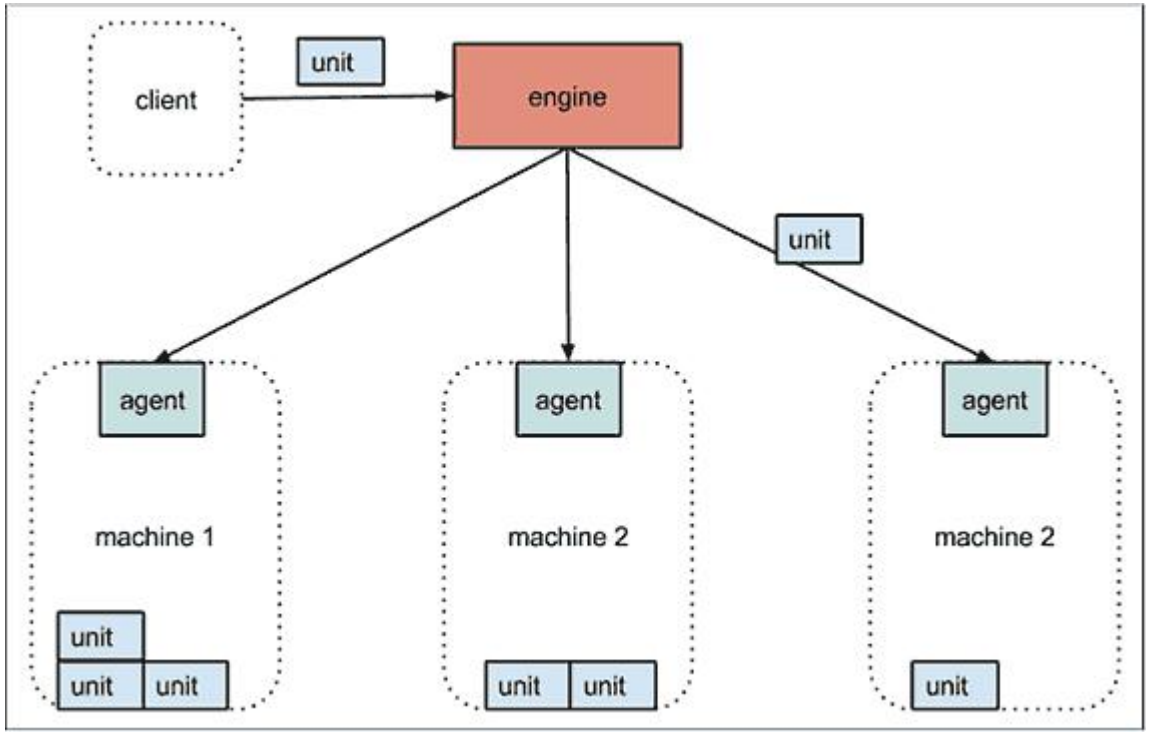
Kubernetes是一个Docker容器的编排系统，它使用label和pod的概念来将容器换分为逻辑单元。Pods是同地协作（co-located）容器的集合，这些容器被共同部署和调度，形成了一个服务，这是Kubernetes和其他两个框架的主要区别。相比于基于相似度的容器调度方式（就像Swarm和Mesos），这个方法简化了对集群的管理。


Kubernetes Cluster
Kubernetes Master
API Server
Replication Controller
Container
Container
Pod
Pod
kubelet
kube-proxy
docker
Node
Container
Container
Pod
Pod
kubelet
kube-proxy
docker
Node
= Labels
= Service

Mesos的目的就是建立一个高效可扩展的系统，并且这个系统能够支持很多各种各样的框架，不管是现在的还是未来的框架，它都能支持。



Fleet是一个来自CoreOS的集群管理工具，自诩为低级别的集群引擎，也就意味着，它可支持从基础层到高层解决方案如Kubernetes。



总结比较

1. Swarm的优点和缺点都是使用标准的Docker接口，使用简单，容易集成到现有系统，但是更困难支持更复杂的调度，比如以定制接口方式定义的调度。
2.Fleet是低层次且相当简单的管理指挥层，能作为运行高级别管理工具如Kubernetes 的基础。
3.Kubernetes 是自成体系的管理工具，有自己的服务发现和复制，需要对现有应用的重新设计，但是能支持失败冗余和扩展系统。Kubernetes是由谷歌的Borg容器管理工具简化的开源版本 。

总结比较

4.Mesos是低级别 battle-hardened调度器，支持几种容器管理框架如Marathon, Kubernetes, and Swarm，现在Kubernetes和 Mesos稳定性超过Swarm，在扩展性方面，Mesos已经被证明支持超大规模的系统，比如数百数千台主机，但是，如果你需要小的集群，比如少于一打数量的节点服务器数量，Mesos也许过于复杂了。

Mesos则由开源社区在谷歌公开出的技术原理上开发出来，但可以适应更大规模的分布式集群系统。

九、Docker最佳实践

介绍JAVA项目如何通过Docker实现持续部署（只需简单四步），**全程无需运维人员参与**即：

开发人员通过git push上传代码

经Git和Jenkins配合

自动完成程序部署

发布

其他变通的方案，把代码放在宿主机上，让容器通过卷组映射来读取。这种方法不建议的原因是，**将代码拆分出容器，这违背了Docker的集装箱原则。**从货运工人角度考虑，整体才是最经济的。这样，也才能实现真正意义的容器级迁移。 或者说，**容器时代，抛弃过去文件分发的思想**。

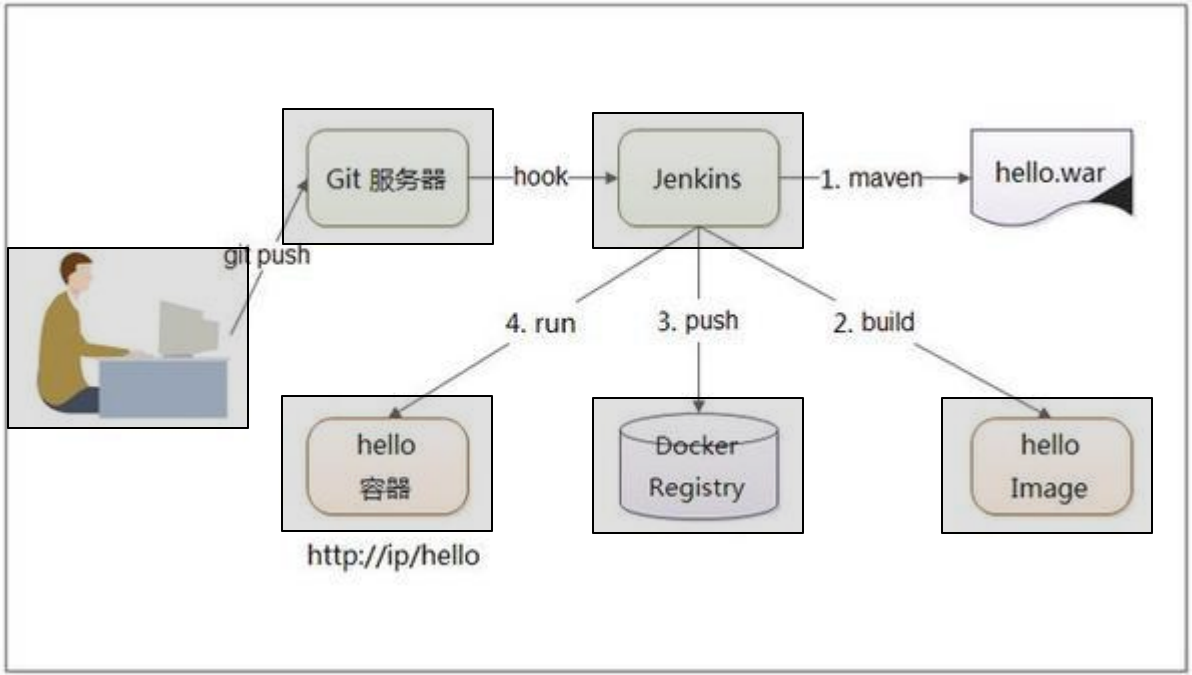
容器即进程。我们采用上述方案做Docker持续部署的原因和意义，也在于此。**容器的生命周期，应该远远短于虚拟机，容器出现问题，应该是立即杀掉，而不是试图恢复。**

知识点介绍

GIT作为开源代码库以及版本控制系统，Github拥有140多万开发者用户。随着越来越多的应用程序转移到了云上，Github已经成为了管理软件开发以及**发现已有代码**的首选方法。

Jenkins是基于Java开发的一种继续集成（Continuous integration）工具，简称CI。
它倡导团队开发成员必须经常集成他们的工作，甚至每天都可能发生多次集成。而每次的集成都是通过自动化的构建来验证，包括自动编译、发布和测试，从而尽快地发现集成错误，让团队能够更快的开发内聚的软件。

图示JAVA项目如何通过Docker实现持续部署即： **开发人员通过git push上传代码，经Git和Jenkins配合，自动完成程序部署、发布，全程无需运维人员参与。**



开发和运维的有效隔离

一个IT系统应该包含如下几个层次:

- 应用程序
- 运行时平台（bin/framework/lib）
- 操作系统
- 硬件（基础设施）

开发人员的主要工作是应用程序的编码、构建、测试和发布，涉及应用程序和运行时平台这两层。而运维人员的工作则涉及从硬件、操作系统到运行时平台的安装、配置、运行监控、升级和优化等工作。docker提供了一种运行时环境,隔离了上层应用于下层操作系统和硬件的关联,使得术业有专攻。

Docker目前在着以下几个缺点

Docker是基于Linux 64bit的，无法在32bit的linux/Windows/unix环境下使用

LXC是基于cgroup等linux kernel功能的，因此container的guest系统只能是linux base的

隔离性相比KVM之类的虚拟化方案还是有些欠缺，所有container公用一部分的运行库

网络管理相对简单，主要是基于namespace隔离

cgroup的cpu和cpuset提供的cpu功能相比KVM的等虚拟化方案相比难以度量(所以dotcloud主要是按内存收费)

docker对disk的管理比较有限

container随着用户进程的停止而销毁，container中的log等用户数据不便收集