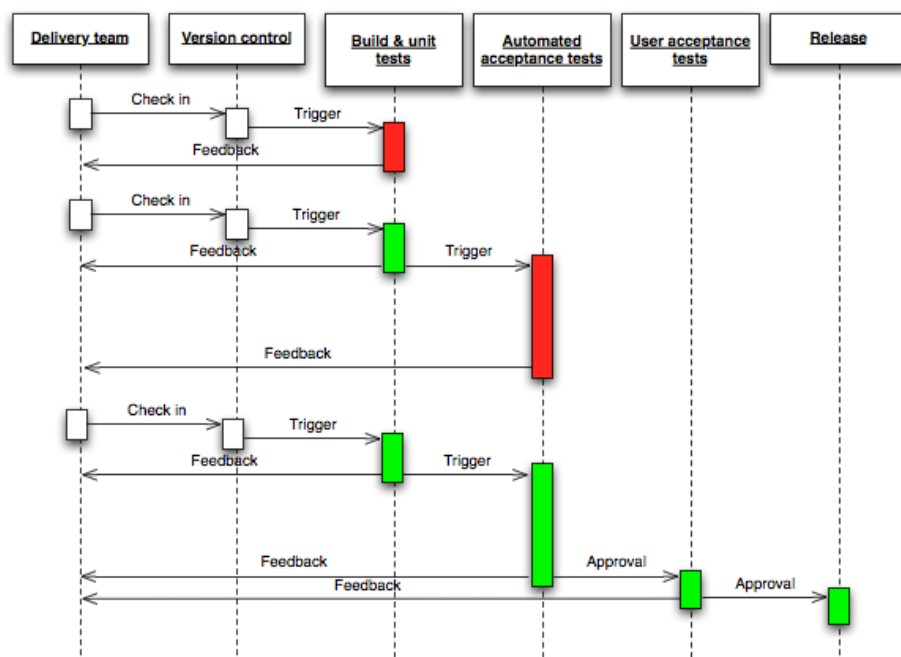


深入浅出Docker（四）：Docker的集成测试部署之道

1. 背景

敏捷开发已经流行了很长时间，如今有越来越多的企业开始践行敏捷开发所提倡的以人为中心、迭代、循序渐进的开发理念。在这样的场景下引入 Docker 技术，首要目的就是使用 Docker 提供的虚拟化方式，给开发团队建立一套可以复用的开发环境，让开发环境可以通过 Image 的形式分享给项目的所有开发成员，以简化开发环境的搭建。但是，在没有 Docker 技术之前就已经有类如 Vagrant 的开发环境分发技术，软件开发者一样可以创建类似需求的环境配置流程。所以在开发环境方面，Docker 技术的优势并不能很好的发挥出来。笔者认为 Docker 的优点在于可以简化 CI（持续集成）、CD（持续交付）的构建流程，让开发者把更多的精力用在开发上。

每家公司都有自己的开发技术栈，我们需要结合实际情况对其进行持续改进，优化自己的构建流程。当我们准备迈出第一步时，我们首先要确立一张构建蓝图，做到胸有成竹，这样接下来的事情才会很快实现。



这张时序图概括了目前敏捷开发流程的所有环节。结合以上时序图给出的蓝图框架，本文的重点是讲解引入 Docker 技术到每个环节中的实践经验。

2. 创建持续发布的团队

开发团队在引入 Docker 技术的时候，最大的问题是没有可遵循的业界标准。大家常常以最佳实践为口号，引入多种工具链，导致在使用 Docker 的过程中没有侧重点。涉及到 Docker 选型，又在工具学习上花费大量时间，而不是选用合适的工具以组建可持续发布产品的开发团队。基于这样的场景，我们可以把“简单易用”的原则作为评判标准，引入到 Docker 技术工具选型的参考中。开发团队在引入 Docker 技术的过程中，首先需要解决的是让团队成员尽快掌握 Docker 命令行的使用。在熟悉了 Docker 命令行之后，团队需要解决几个关键问题具体如下：

- 1) Base Image 的选择, 比如 [phusion-baseimage](#)
- 2) 配置管理 Docker 镜像的工具的选择, 比如 [Ansible](#)、[Chef](#)、[Puppet](#)
- 3) Host 主机系统的选择, 比如 [CoreOS](#)、[Atomic](#)、[Ubuntu](#)

Base Image 包括了操作系统命令行和类库的最小集合，一旦启用，所有应用都需要以它为基础创建应用镜像。Ubuntu 作为官方使用的默认版本，是目前最易用的版本，但系统没有经过优化，可以考虑使用第三方有划过的版本，比如 [phusion-baseimage](#)。对于选择 RHEL、CentOS 分支的 Base Image，提供安全框架 SELinux 的使用、块级存储文件系统 devicemapper 等技术，这些特性是不能和 Ubuntu 分支通用的。另外需要注意的是，使用的操作系统分支不同，其裁

剪系统的方法也完全不同，所以大家在选择操作系统时一定要慎重。

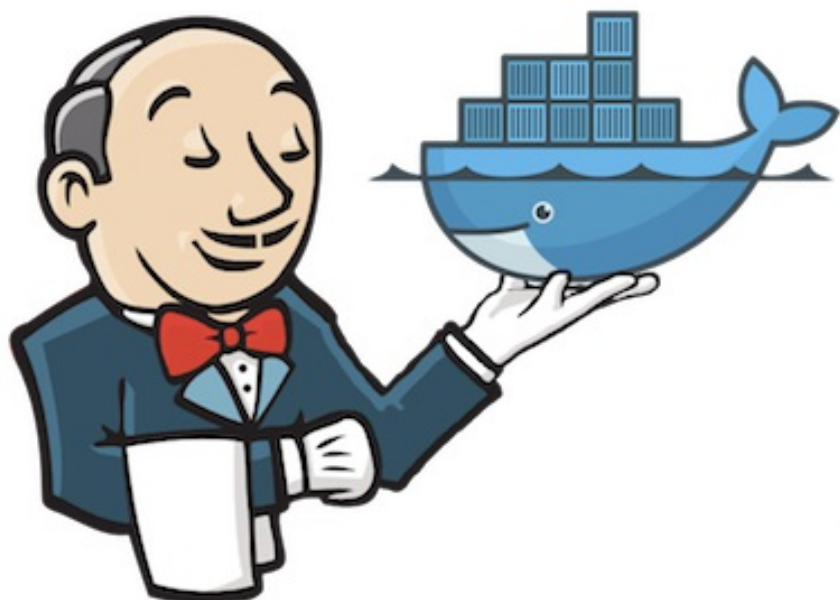
配置管理Docker镜像的工具主要用于基于Dockerfile创建Image的配置管理。我们需要结合开发团队的现状，选择一款团队熟悉的工具作为通用工具。配置工具有很多种选择，其中[Ansible](#)作为后起之秀，在配置管理的使用中体验非常简单易用，推荐大家参考使用。

Host主机系统是Docker后台进程的运行环境。从开发角度来看，它就是一台普通的单机OS系统，我们仅部署Docker后台进程以及集群工具，所以希望Host主机系统的开销越小越好。这里推荐给大家的Host主机系统是[CoreOS](#)，它是目前开销最小的主机系统。另外，还有红帽的开源[Atomic](#)主机系统，有基于[Fedora](#)、[CentOS](#)、[RHEL](#)多个版本的分支选择，也是不错的候选对象。另外一种情况是选择最小安装操作系统，自己定制Host主机系统。如果你的团队有这个实力，可以考虑自己定制这样的系统。

3. 持续集成的构建系统

当开发团队把代码提交到Git应用仓库的那一刻，我相信所有的开发者都希望有一个系统能帮助他们把这个应用程序部署到应用服务器上，以节省不必要的人工成本。但是，复杂的应用部署场景，让这个想法实现起来并不简单。

首先，我们需要有一个支持Docker的构建系统，这里推荐[Jenkins](#)。它的主要特点是项目开源、方便定制、使用简单。Jenkins可以方便的安装各种第三方插件，从而方便快捷的集成第三方的应用。



通过Jenkins系统的Job触发机制，我们可以方便的创建各种类型的集成Job用例。但缺乏统一标准的Job用例使用方法，会导致项目Job用例使用的混乱，难于管理维护。这也让开发团队无法充分利用好集成系统的优势，当然这也不是我们期望的结果。所以，敏捷实践方法提出了一个可以持续交付的概念 [Deployment Pipeline](#)（管道部署）。通过Docker技术，我们可以很方便的理解并实施这个方法。

Jenkins的管道部署把部署的流程形象化成为一个长长的管道，每间隔一小段会有一个节点，也就是Job，完成这个Job工作后可以进入下一个环节。形式如下：

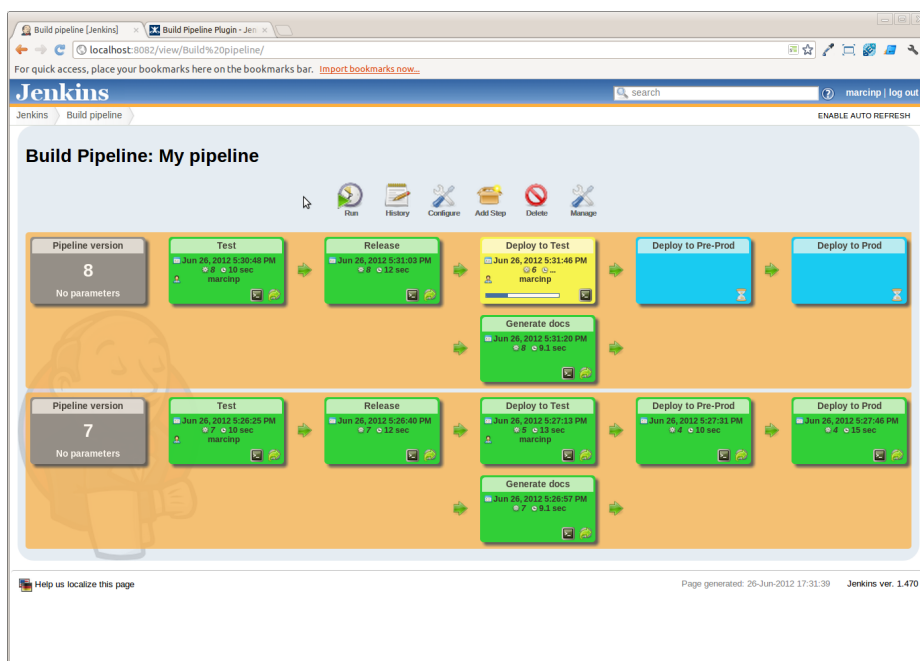


image source: google image search

大家看到上图中的每一块面板在引入Docker技术之后，就可以使用Docker把任务模块化，然后做成有针对性的Image用来跑需要的任务。每一个任务Image的创建工作又可以在开发者自己的环境中完成，类似的场景可以参考下图：

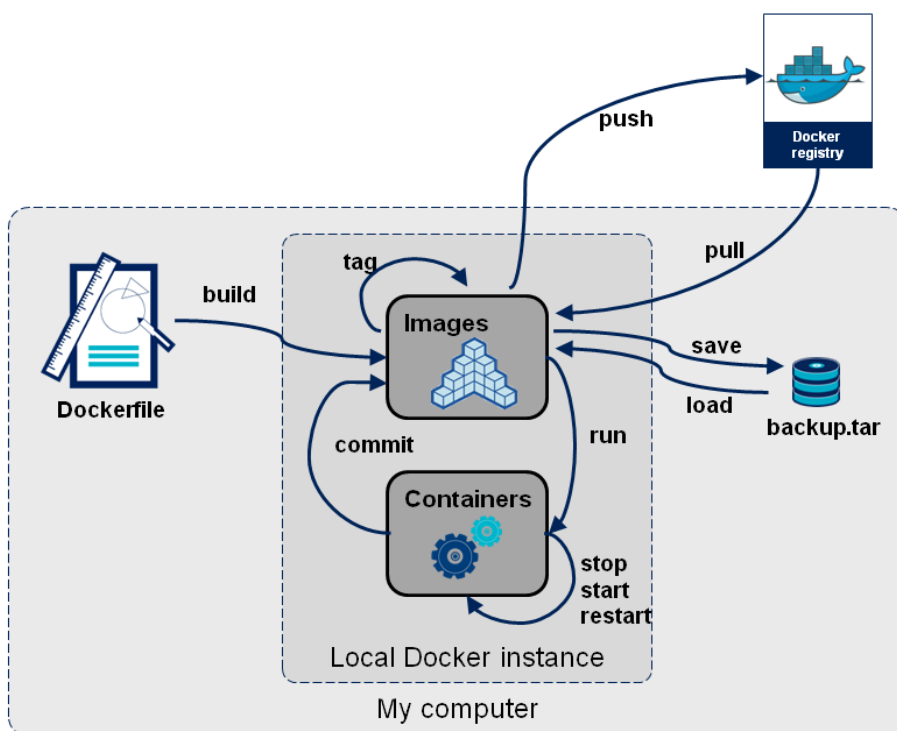


image source: google image search

所以，使用Docker之后，任务的模块化很自然地定义出来。通过管道图，可以查看每一步的执行时间。开发者也可以针对任务的需要，为每一个任务定义严格的性能标准，已作为之后测试工作的参考基础。

4.最佳的发布环境

应用经过测试，接下来我们需要把它发布到测试环境和生产环境。这个阶段中如何更合理地使用Docker也是一个难点，开发团队需要考虑如何打造一个可伸缩扩展的分发环境。其实，这个环境就是基于Docker的私有云，更进一步我们可能期望的是提供API接口的PaaS云服务。为了构建此PaaS服务，这里推荐几款非常热门的工具方便大家参考，通过这些工具可以定制出企业私有的PaaS服务。

1) Apache Mesos + marathon

Apache Mesos系统是一套资源管理调度集群系统，生产环境使用它可以实现应用集群。此系统是由Twitter发起的Apache开源项目。在这个集群系统里，我们可以使用Zookeeper开启3个Mesos master服务，当3个Mesos master通过zookeeper交换信息后会选出Leader服务，这时发给其它两台Slave Mesos Master上的请求会转发到Mesos master Leader服务。Mesos slave服务器在开启后会把内存、存储空间和CPU 资源信息发给Mesos master。Mesos是一个框架，在设计它的时候只是为了用它执行Job来做数据分析。它并不能运行一个比如Web服务Nginx这样长时间运行的服务，所以我们需要借助marathon来支持这个需求。marathon有自己的REST API，我们可以创建如下的配置文件Docker.json：

```
{
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "libmesos/ubuntu"
    }
  },
  "id": "ubuntu",
  "instances": "1",
  "cpus": "0.5",
  "mem": "512",
  "uris": [],
  "cmd": "while sleep 10; do date -u +%T; done"
}
```

然后调用

```
curl -X POST -H "Content-Type: application/json" http://<master>:8080/v2/apps -d@Docker.json
```

我们就可以创建一个Web服务在Mesos集群上。对于Marathon的具体案例，可以[参考官方案例](#)。

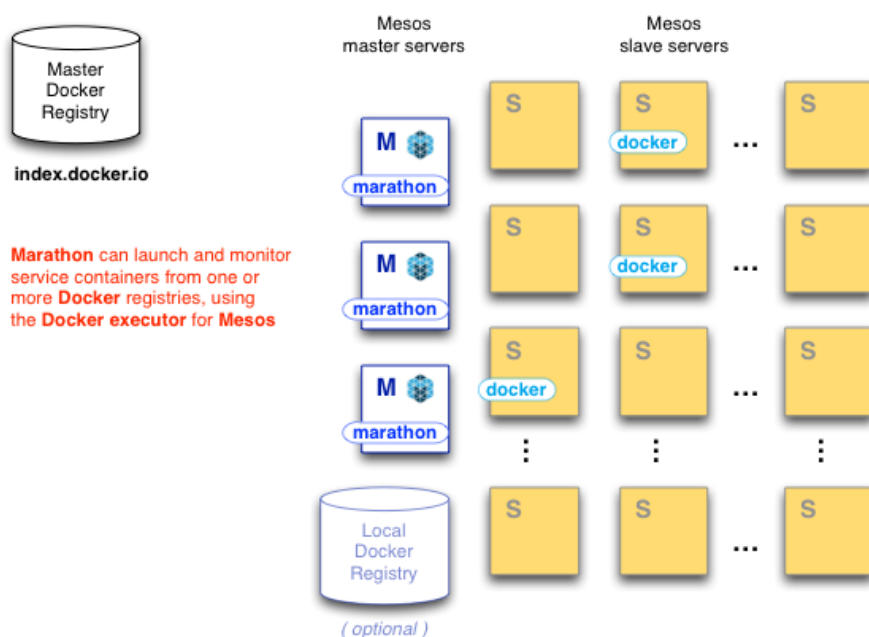


image source: google image search

2) Google Kubernetes

Google的一个容器集群管理工具，它提出两个概念：

1. Pods，每个Pod是一个容器的集合并部署在同一台主机上，共享IP地址和存储空间，比如Apache，Redis之类分为一组容器集合。
2. Labels，提供服务标签，方便Pod容器之间的调用协作。

通过官方[架构设计](#)文档的介绍，可以详细的了解每个组件的设计思想。这是目前业界唯一在生产环境部署经验的基础上

The diagram illustrates the Kubernetes architecture, divided into Master components and Minion components.

Master Components:

- kubectfg (user commands):** Interacts with the **authorization authentication** component.
- authorization authentication:** Connects to the **REST (pods, services, rep. controllers)** component.
- REST (pods, services, rep. controllers):** The central component that interacts with the **scheduling actuator**, **Scheduler**, **replication controller**, **kubelet info service**, and **Distributed Watchable Storage (implemented via etcd)**.
- scheduling actuator:** Connects to the **Scheduler**.
- Scheduler:** Connects to the **REST** component.
- replication controller:** Connects to the **REST** component.
- kubelet info service:** Connects to the **REST** component and the **kubelet** in Minions.
- Distributed Watchable Storage (implemented via etcd):** Connects to the **REST** component and the **kubelet** in Minions.

Minion Components:

- Internet:** Connects to the **Firewall**.
- Firewall:** Connects to the **Proxy** in Minions.
- docker:** Connects to the **kubelet** and **cAdvisor**.
- kubelet:** Connects to the **REST** component, **Distributed Watchable Storage**, **cAdvisor**, and **Proxy**.
- cAdvisor:** Connects to the **kubelet** and **Proxy**.
- Proxy:** Connects to the **kubelet** and **containers**.
- Pod:** Contains **containers**.

Master components Colocated, or spread across machines, as dictated by cluster size.

3) [Panamax](#)

在琳琅满目的集群管理工具面前，如何管理单机的Docker容器也是一个需要解决问题。因为Docker占用内存小，在单机服务器上部署成百上千个容器也不足为奇。Panamax提供人性化的Web管理界面用来安装软件让部署变得更简单。并且，Panamax还提供丰富的[容器模板](#)，让在线创建服务成为可能。比如到DigitalOcean申请一台主机，安装一套Panamax启动为后台服务。然后通过Panamax Web界面安装Nginx、Mysql、Redis等服务镜像，这样可以快速搭建生产环境的应用场景。所有的操作都是在Web界面上完成，开发者只需要关注开发本身即可。

在琳琅满目的集群管理工具面前，如何管理单机的Docker容器也是一个需要解决问题。因为Docker占用内存小，在单机服务器上部署成百上千个容器也不足为奇。Panamax提供人性化的Web管理界面用来安装软件让部署变得更简单。并且，Panamax还提供丰富的[容器模板](#)，让在线创建服务成为可能。比如到DigitalOcean申请一台主机，安装一套Panamax启动为后台服务。然后通过Panamax Web界面安装Nginx、Mysql、Redis等服务镜像，这样可以快速搭建生产环境的应用场景。所有的操作都是在Web界面上完成，开发者只需要关注开发本身即可。

在琳琅满目的集群管理工具面前，如何管理单机的Docker容器也是一个需要解决的问题。因为Docker占用内存小，在单机服务器上部署成百上千个容器也不足为奇。Panamax提供人性化的Web管理界面用来安装软件让部署变得更简单。并且，Panamax还提供丰富的[容器模板](#)，让在线创建服务成为可能。比如到DigitalOcean申请一台主机，安装一套Panamax启动为后台服务。然后通过Panamax Web界面安装Nginx、Mysql、Redis等服务镜像，这样可以快速搭建生产环境的应用场景。所有的操作都是在Web界面上完成，开发者只需要关注开发本身即可。

CoreOS Host:



CenturyLink

Manage / Dashboard / Applications /

Socialize!

Deployed to: CoreOS Local
[Documentation](#)
[Access your application](#)

★ Save as Template ↻ Rebuild App ✕ Delete App

Application Services  

WORKERS

redis_latest

etl

+ Add a Service

API

postgres

api

+ Add a Service

SUPPORT

erbit

mongo

+ Add a Service

UI

ui

+ Add a Service

Add a Category

+

CoreOS Journal - Application Activity Log Show Full Activity Log

5. 结论

Docker的集成部署方案，是一套灵活简单的工具集解决方案。它克服了之前集群工具复杂、难用的困境，使用统一的Docker应用容器的概念部署 软件应用。通过引入Docker技术，开发团队在面对复杂的生产环境中，可以结合自己团队的实际情况，定制出适合自己基础架构的配套软件发布方案。

6. 作者简介

肖德时, Red Hat Engineering Service/HSS 内部工具组Team Leader. Nodejs开源项目nodejs-cantas Lead Developer。擅长企业内部工具的设计以及实现。开源课程Rails Starter的发起人。rubygem: lazy_high_charts的Maintainer。twitter账号：xds2000，邮箱：xiaods@gmail.com

7. 下期预告

Docker在生产环境的应用实践已经给大家介绍，下期我将给大家介绍如何基于Docker快速构建开发环境，敬请期待！