基于 Docker 的 PaaS 平台建设①

王亚玲[1], 李春阳[1], 崔 蔚[1], 张 晶[2]

[1](国网信息通信产业有限公司, 北京 100031)
[2](北京中电普华信息技术有限公司, 北京 100192)

摘 要: 随着信息化时代的进步, 企业级系统越发庞大复杂, 系统级研发面临巨大的挑战. 从设计开发测试到实施, 有着种种的困难, 针对这一问题, 提供一套基于 Docker 的企业私有 PaaS 云的建设思路. 通过基于 Docker 的容器虚拟化、面向服务的分布式架构设计、基于 Docker 的服务发现、基于私有云的环境配置管理等技术手段, 构建针对企业级系统研发的 PaaS 平台. 实践应用表明, 基于 Docker 的 PaaS 平台, 能有效降低搭建研发环境的复杂度, 降低系统升级成本, 提高测试准确性, 从而有效地提升企业级系统研发效率.

关键词: 私有云; PAAS; Docker

Construction of Docker-Based PaaS

WANG Ya-Ling[1], LI Chun-Yang[1], CUI Wei[1], ZHANG Jing[2]

[1](State Grid Information & Telecommunication Industry Co. Ltd., Beijing 100031, China)
[2](Beijing China Power Information Technology Co. Ltd, Beijing 100192, China)

Abstract: The enterprise system becomes huger and more complicated with the increase of information building. And the developers must face this challenge. There are many difficulties to overcome with the work of design, development and testing. This paper provides a solution with PaaS based on Docker to solve this problem. There are many technologies in this solution, such as the VM technology with docker, SOA, service discovery technology with docker and configuration work base on private cloud. Using these technologies, we can manage the development and testing environment more smoothly, and also manage the architecture of the system and deployment more effectively.

Key words: private cloud; PAAS; Docker

面对信息化系统庞大、系统实现与部署复杂的问题, 各个开发商普遍的处理方法有以下两种模式: 其一, 通过改进开发模式来处理. 这种方式多见是以敏捷的开发模式, 应对系统的庞杂, 从小做起, 不断的修正应用系统设计以适应越来越多的信息建设需求. 其二, 利用企业级开发框架, 来适应庞大的业务系统建设. 常见的企业级框架有 J2EE 企业级框架, 以及基于 J2EE 的 SOA 平台分布式框架.

但是, 不论是从方式方法上, 还是从企业级架构上, 依然面对一系列开发中的实际问题, 如环境搭建及升级困难、团队协作困难、质量难以保证等.

随着云时代的来临, 解决面临的挑战似乎看到了希望. 从 SaaS 的概念的实现, 到 IaaS 的实现, 似乎从虚拟化的技术中, 可以找到一定的解决方案. 但是, 这仅仅是在生产环境的实施中找到了解决方案. 面对研发的相关问题, 依然找不到合适的答案.

在 2008 年, Google 公司发布了 Google App Engine, 亚马逊公司也在同时期发布了 EC2, S3, SimpleDB 等服务, 宣布了公有 PaaS 的来临. PaaS 的到来, 给研发人员带来了曙光[1].

然而, 对于企业级私有 PaaS 一直不是很成熟, 包括安全性、稳定性、可靠性方面让各个开发商依然持保留态度. 直到 Docker 的出现, 这一局面获得了全面的改观.

Docker 以其先进的设计理念, 非常高的安全性、稳定性、高效性, 让开发商敢于迈进企业私有云的时

① 收稿时间:2015-05-29;收到修改稿时间:2015-07-06

代. 包括谷歌、百度在内的众多 IT 巨头向 Docker 展开了怀抱.

1 技术背景

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 机器上, 也可以实现虚拟化. 容器是完全使用沙箱机制, 相互之间不会有任何接口. 几乎没有性能开销,可以很容易地在机器和数据中心中运行. 最重要的是,他们不依赖于任何语言、框架或包括系统. 应用被打包成 Docker Image 后, 部署和运维变的极其简单. 由于其基于 LXC 的轻量级虚拟化的特点, Docker 相比虚拟机最明显的特点就是启动快, 资源占用小. 因此它更适合构建隔离的标准化的运行环境、轻量级的 PaaS, 以及 2 一切可以横向扩展的应用.

1.1 基于 Docker 的虚拟化技术

传统非基于容器的 PaaS 云往往是基于硬件虚拟化的实现方式. 每个虚拟环境都虚拟出一个完整的操作系统, 在此基础上再配置相应的运行环境. 系统的维护性并不是很好, 因为每套虚拟系统都需要配置. 即使是可复制的配置环境, 也缺乏灵活性, 没有基于增量的扩展配置能力. 而 Docker 是基于容器设计的, 所以, 从结构上与非基于容器的虚拟化技术有着很大的不同[2], 具体如图 1 所示.

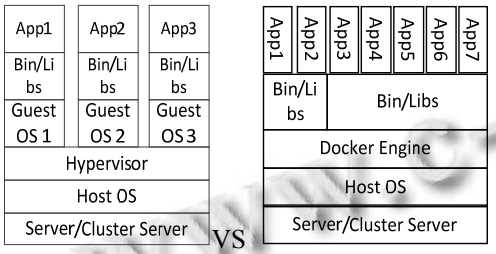

图 1 虚拟化结构对比

从图中, 我们可以看到, 基于 Docker 的虚拟化结构有如下特点[3]:

首先, 基于 Docker 的虚拟化是建立在操作系统之上, 是系统级的虚拟, 并没有针对硬件虚拟. 这样就保证了虚拟镜像的移植性比传统的基于硬件的移植性更强.

其次, 基于 Docker 的虚拟化是基于容器的. 通过共享底层虚拟的容器, 可以更加便捷地虚拟出独立的应用系统. 一方面, 在物理资源的占用上, 保证了每个虚拟镜像只保留变化的部分, 大大减少了虚拟镜像消耗的空间. 另一方面, 针对容器的虚拟化, 可以充分地利用增量发布的能力, 通过继承的方式制作镜像, 灵活性更高.

1.2 基于 Docker 的服务发现技术

基于 Docker 的服务发现技术有很多种实现, 针对企业级系统研发有几个设计要求: ①服务分布式地部署在各个物理环境中, 所以, 要能够有效地发现分布式的服务; ②由于企业级系统足够庞大导致服务部署的物理环境足够庞大, 所以, 服务发现的引擎本身要能够集群部署.

这里我们采用 Docker + Etcd + Haproxy 的技术组合来实现服务发现, 如图 2 所示.

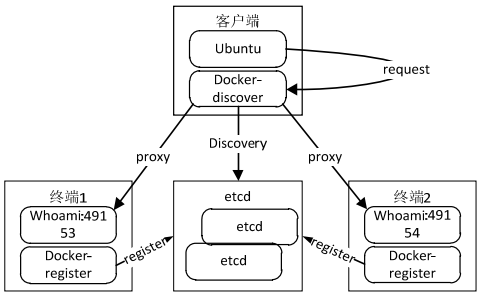

图 2 基于 Docker 的服务发现

基于 Docker、 Etcd 和 Haproxy 的服务发现机制, 与 SmartStack 很像, 我们拥有一个服务注册组件(etcd), 一个注册伙伴进程(docker-register), 一个发现伙伴进程(docker-discover), 一些终端服务(Whoami)以及一些消费者(ubuntu/curl). 注册和发现组件在容器中独立运行, 负责监听端口或者连接到其他主机的端口, 不需要编写容器植入性代码. 可以实现服务注册集群, 以及分布式的终端部署集成.

2 基于私有PaaS平台的系统研发需求

基于私有 PaaS 云, 结合系统研发过程中面临的问题, 构建适应于企业级系统研发的 PaaS 平台有以下几方面的需求[4]:

在开发环境搭建上, 平台应提供快速的环境搭建方法, 解决传统环境搭建困难耗时、工程师难以快速上手的问题;

在质量保证的方面, 应保证测试环境搭建的便捷性及与研发环境的一致性, 解决由于实施的原因以及

硬件资源的限制而带来的测试问题;

在部署实施上, 传统研发模式对系统运行环境要求十分苛刻, 平台应提供合理规划的运行环境, 解决运维工作开展困难的问题;

在系统维护工作中, 由于系统复杂, 对工程师的水平要求也很高, 维护困难, 成本很高, 平台应提供对应的解决方案;

在系统的升级改造工作中, 平台应提供灵活的升级策略, 解决系统升级中牵一发动全身的状况;

在工程的结构上, 传统业务系统研发功能过于集中, 互相影响, 移植性很差. 这不仅仅是开发设计的缺陷问题, 也是实际的硬件资源限制导致的, 平台应提供对应的解决策略;

在团队之间的协同配合上, 平台应提供松耦合的研发机制, 解决团队间工作彼此依赖的问题.

3 基于Docker的Paas平台在企业级系统研发中的应用研究

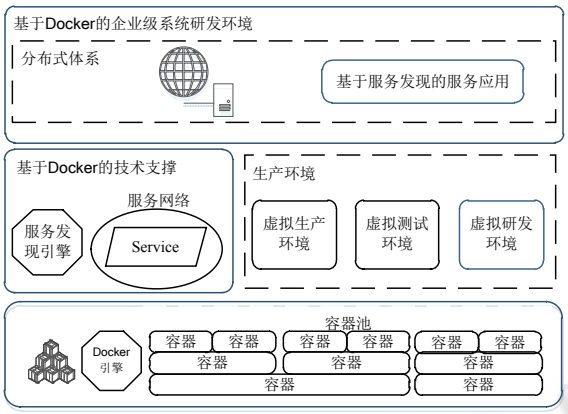

图 3 基于 Docker 的企业级系统研发架构

如图 3, 企业级系统研发环境架构主要分为基础支撑的 Docker 虚拟化容器池、基于 Docker 的服务发现、生产环境、测试环境、研发环境以及顶层的基于分布式服务的应用架构[5]. 下边分别针对各部分进行说明.

3.1 面向服务的分布式系统架构

随着分布式技术的发展, 越来越多的面向服务的分布式系统架构涌现. 这些架构的技术成熟度已经非常高, 系统的稳定性和性能也十分突出. 而且, 随着硬件水平的提高, 各种分布式技术所带来的网络开销、内存开销、运算开销几乎都可以忽略不计. 系统研发人员可以更加信赖分布式技术, 更加关注业务系统的实现.

长期的系统研发实践证明优秀的分布式设计, 可以有效地完成系统解耦的工作, 而提高系统的可维护性和灵活性. 研发团队或个人能够投入更多精力去关心业务本身, 从而提高系统的质量. 也正是因为业务功能的元素化, 维护内容的专一, 在维护过程中对单个服务功能的维护也就越发容易, 维护成本也越发低.

3.2 基于服务发现的系统自升级环境

基于服务发现技术, 可以实现分布式协同研发的效果. 在项目研发过程中, 通过系统模块化、服务元素化的设计, 把研发单元划分的足够独立. 不同的服务独立的运行在虚拟容器中, 使得研发人员可以专注地对单个服务进行研发工作. 在研发初期, 没有实现具体服务, 而又拥有依赖关系的情况下, 可以部署伪实现的服务, 从而提供系统集成. 而在研发任务完成后, 可以通过服务发现技术, 热部署已经完成的功能服务, 不需要重启系统. 这样, 业务系统模块可以在无感知的情况下享用部署后的应用服务, 可以在无感知的情况下进行系统集成和升级, 减少研发过程中的依赖, 实现分布式协同研发的效果[6].

在前文中提到的研发过程的系统升级难、分布式协同研发难的问题, 都可以通过服务发现的技术得到解决.

3.3 基于 Docker 的开发环境虚拟化

针对开发环境搭建及维护困难的问题, 可以通过 Docker 虚拟化研发环境并生成镜像的方式解决.

在基于 Docker 的开发环境结构(如图 4)中, 通过把研发环境虚拟成多个镜像来管理. 如图中虚拟的应用服务器、数据库服务器、其他中间件服务器; 同时还以增量的方式虚拟出拥有 JDK 和其他依赖系统环境的基础容器镜像、拥有各种开发人员需要的 IDE 或者开发平台的高级容器镜像、研发人员个性化的开发环境容器镜像.

这样的结构设计, 充分保证了开发环境的独立性; 也充分利用了容器的增量性, 定制了针对项目通用的基础容器; 同时充分地考虑到研发人员的个性化需求, 提供其自行设定的环境容器.

在环境升级的需求上, 可以通过升级容器, 达到只变动一处就能修改所有环境的效果. 而且, Docker 能够通过手动或自动感知容器变化的方式升级. 这为

环境的升级带来了非常大的便利.

前文中提到的企业级系统研发中遇到的开发环境搭建难的问题, 在这里得到了有效的解决. 只要制定了统一、标准的制式环境镜像, 研发人员只需执行一个简单的脚本, 快速地构建自己的开发环境.

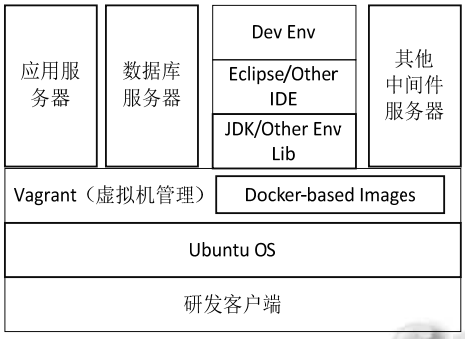

图 4 基于 Docker 的开发环境结构

3.4 基于 Docker 的测试环境持续集成

基于 Docker 的虚拟化技术, 可以利用有限的资源, 虚拟出足够的测试环境, 大幅降低硬件资源的消耗成本.

在测试环境搭建过程中, 可以使用与开发环境相同的应用服务器、数据库服务器、中间件服务器的镜像, 达到开发环境和测试环境同步的目的. 而且, 针对业务应用服务器, 可以采用持续构建技术持续构建独立的业务系统.

在面向服务的分布式架构下, 测试可以针对单个服务进行功能性测试以及性能测试等.

测试人员也可以建立自己的自动化测试环境, 保持环境的独立性. 这与研发人员的环境维护是类似的.

3.5 基于 Docker 的生产环境管理

上文提到, 在生产环境的实施过程中, 经常会遇到单个环境部署着多个应用或服务的混乱情况; 也经常遇到系统升级困难的情况.

基于 Docker 的虚拟化技术的部署环境的独立性, 以及基于 Docker 的服务发现机制带来的系统升级便捷性, 能有效地解决在生产环境实施管理以及运维管理中遇到的问题.

4 基于Docker的PaaS平台搭建与使用

4.1 基于 Docker 的研发环境搭建

首先, 利用 Docker, 配置一个开发环境. 利用容器增量的特性, 配置出针对不同用户(包括开发人员、测试人员、版本管理人员等)的开发环境, 并通过 Docker build 命令制作出镜像[7].

然后, 通过 vagrant 技术, 把 Docker 的虚拟镜像虚拟成系统. 基于不同的操作系统, 都可以通过 Vagrant 虚拟并运行 Docker 容器. 通过这样的特性, 减少了对开发环境的限制, 不局限于 Docker 所依赖的 64 位的 Linux 操作系统.

其次, 针对容器和本地主机的端口映射, 通过 Vagrant 的配置文件开放容器的端口:

```
config.vm.network "forwarded_port",
guest: 8080, host: 8080
```

这样, 就解决了虚拟系统之间的集成问题. 研发环境搭建效果如图 5 所示.

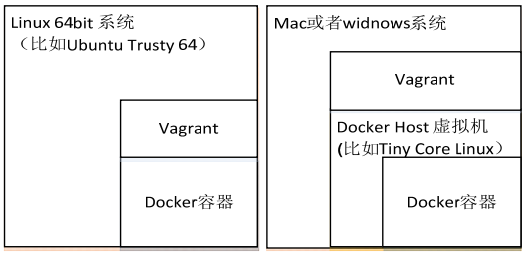

图 5 基于 docker 的研发环境搭建效果

最后, 根据实际应用情况完善虚拟结构, 即同时运行多个虚拟容器. 在 Vagrant 的配置中, 配置三台虚拟机, 一台是应用服务器, 一台是数据库服务器, 一台是开发机.

```
Vagrant.configure("2") do |config|
  config.vm.define "appserver" do |a|
  …
  end
  config.vm.define "dbserver" do |a|
  …
  end
  config.vm.define "devenv" do |a|
  …
  end
end
```

4.2 Docker 的自感知升级

Docker 提供的基础环境变化的感知功能可以通过如下命令实现[8]:

```
LATEST=`docker inspect --format "{{.Id}}" $IMAGE`
```

RUNNING=`docker inspect --format "{{.Image}}" $im`

通过对两个语句的执行结构进行比较, 就能获知当前的镜像是否为最新的结果.

此外, Docker 提供镜像的升级命令:

docker run itech/docker-unattended-upgrade

利用以上的核心指令, 就可以编写自动检查和升级的脚本. 在实际应用中, 通过编写一个 cron 任务, 执行对应的版本检查和升级脚本, 实现虚拟环境的自动感知和升级(如图 6):

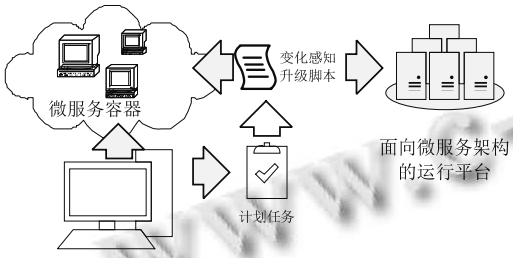

图 6　镜像升级流程图

4.3 基于 Docker 的服务发现

这里采用 Docker + Etcd + Haproxy 的技术组合实现基于 Docker 的服务发现.

在上文中提到, 把服务注册在服务发现的 Etcd 集群里, 在终端, 通过服务发现进行服务转发, 以代理的形式把最终的服务展现在终端, 从而达到服务调用的目的.

当服务发生变化的时候, 服务发现引擎 Etcd 会管理其注册的服务, 进行服务注册、注销、管理等响应操作, 并不需要改变服务终端. 这种代理和隔离的机制实现了服务发现.

在具体实现过程中, 首先运行一个服务发现服务:

docker run -d --name etcd -p 4001:4001 -p 7001:7001 coreos/etcd

然后, 注册服务到 Etcd:

HOST_IP=$(hostname --all-ip-addresses | awk '{print $1}')

ETCD_HOST=w.x.y.z:4001

docker run --name docker-register -d -e HOST_IP=$HOST_IP -e ETCD_HOST=$ETCD_HOST -v /var/run/docker.sock:/var/run/docker.sock -t jwilder/docker-register

最后在终端, 运行服务发现:

ETCD_HOST=w.x.y.z:4001

docker run -d --net host --name docker-discover -e ETCD_HOST=$ETCD_HOST -p 127.0.0.1:1936:1936 -t jwilder/docker-discover

以上就是服务发现的核心实现. 通过拓展和完善这套机制, 就可以设计出基于 Docker 的服务发现架构.

4.4 基于 Docker 的 PaaS 平台使用

通过建立基于统一应用开发平台(SG-UAP)的 PaaS 平台验证基于 Docker 的 PaaS 平台的可行性. 基于 SG-UAP 的 PaaS 平台包括服务化的集成开发工具和开发环境、基于 SG-UAP 的业务应用运维和测试环境.

业务应用开发者通过平台的 Web 界面选择业务系统涉及的应用服务, 根据开发者制定的服务需求, 服务交互层发现相应的平台服务并通知服务管理者, 服务管理者根据服务器状态选择合适的结点提供服务, 并通过 web 界面返还服务提供者的相关服务信息. 开发者在获取到这些信息后, 通过平台提供的支持服务和 Web API 进行应用开发, 并在开发完成后把应用发布到平台上.

通过实际应用表明, 基于 Docker 的 PaaS 平台, 对研发、测试以及实施都带来了极大的便利, 能有效地提高研发效率.

表 1　基于 Docker 的 PaaS 平台带来的性能提升

对比项	传统研发	基于 Docker 的 PaaS 平台
环境搭建	2-3 人/天	0.2 人/天
测试	与研发环境不一致导致测试重复率高、效率低下	测试重复率降低 95%
系统升级	需要停止服务器	无感知升级

5　结语

本文研究并实现了一套基于 Docker 的私有 PaaS 云的实现. 这套基于 Docker 的私有 PaaS 云是一个整体的解决方案. 它在研发、测试环节上节省了硬件资源的使用, 解决了现有硬件环境下的资源不足的问题, 使得各个应用的部署环境更加独立. 在系统架构上, 采用了更加深化的面向服务架构设计, 配合服务发现机制, 使得系统更加灵活, 维护更加便捷, 团队协同更加顺畅. 在生产环境实施上, 基于容器的虚拟化技术让系统的兼容性、移植性更高; 基于服务的设计, 让

系统的升级更加平滑; 基于服务发现的技术, 让系统鲁棒性更好, 系统更加稳定. 下一步的工作, 是建设一套完善的基于私有云的分布式服务设计规范, 让开发者有所遵循的依据. 而且, 也要针对目前基于 Docker 的私有云技术, 设计一套更加体系化的开发模式, 结合迭代、敏捷的开发模式, 让企业架构的研发不论是从技术上还是从模式上都更加体系化.

参考文献

1 Rimal BP, Choi E, Lumb I. A taxonomy and survey of cloud computing systems. Proc. of the 2009 Fifth International Joint Conference on INC, IMS and IDC (NCM09). Washington, DC. IEEE Comp Society. 2009. 44–51.

2 Ye K, Jiang X, Ye D, et al. Two optimization mechanisms to improve the isolation property of server consolidation in virtualized multi-core server. Proc. of the 12th IEEE International Conference on High Performance Computing and Communications(HPCC*10). Piscataway. IEEE. 2010. 281–288.

3 LXC: Linux container tools. http://www.ibm.com/developerworks/library/l-lxc-containers/. [2009-02-03].

4 IBM 虚拟化与云计算小组.虚拟化与云计算.北京:电子工业出版社,2010.

5 Biederman E, Networx L. Multiple instances of the global linux namespaces. Proc. of the Linux Symposium. 2006.

6 罗军舟,金嘉晖,宋爱波,东方.云计算:体系架构与关键技术.通信学报,2011,32(7):3–21.

7 杨保华,戴玉剑,曹亚仑.Docker 技术入门与实践.北京:机械工业出版社,2014.

8 Turnbull J,李兆海,刘斌,巨震,译.第一本 Docker 书.北京:人民邮电出版社,2014.