文章编号:2095-6134(2018)04-0569-07

简 报

CCI:一种基于容器化的持续集成系统*

张兆晨†,罗铁坚

(中国科学院大学 计算机与控制学院,北京 101408)

(2017年3月10日收稿;2017年5月16日收修改稿)

Zhang Z C, Luo T J. CCI: a containerization-based continuous integration system[J]. Journal of University of Chinese Academy of Sciences, 2018,35(4):569-575.

摘 要 随着软件复杂度的增加和项目需求的变化,快速持续、高质量的实现应用交付成为开发的主要目标。传统软件构建方法缺乏团队的协同性和标准化的流程管理,往往需要投入大量的时间和成本来保障软件质量。为解决时间、质量和成本三者之间的矛盾,研究搭建一个基于Jenkins的持续集成构建系统,并利用Docker技术将持续集成平台容器化,标准化软件环境。实验表明容器化将搭建环境的复杂度降低90%以上,提高了应用测试环境的可移植性和多样性,大大提升软件开发效率。

关键词 软件构建;持续集成;容器化;DevOps

中图分类号:TP311.51 **文献标志码**:A **doi**:10.7523/j.issn.2095-6134.2018.04.021

CCI: a containerization-based continuous integration system

ZHANG Zhaochen, LUO Tiejian

(*School of Computer and Control Engineering, University of Chinese Academy of Sciences, Beijing* 101408, *China*)

Abstract With the increasing of software complexity and the changing of project requirements, it is the main consideration in software development to realize the rapid application delivery with high quality. However, traditional software-build methods lack team cooperation and standardized process management, and one needs to invest a lot of time and cost to ensure software quality. In order to solve this problem, we build a continuous integration system based on Jenkins and standardize the software environment by using Docker technology. The experimental results show that Docker reduces the complexity of building environment by more than 90%, improves the portability and diversity of the testing environment, and greatly increases the efficiency of software development.

Keywords software build; continuous integration; containerization; DevOps

随着软件项目规模逐渐扩大,市场需求不稳定,软件产品需要快速、持续、高质量的实现软件交付,从而更好地应对市场及用户的需求。软件构建是软件生命周期的关键环节,包含从源码编

* 中国科学院仪器设备共享管理系统优化项目(Y42901VED2)资助

† 通信作者,E-mail:zhangzhaochen14@mails.ucas.ac.cn

译到产品交付的整个过程。但缺乏规范化和标准化的软件构建流程往往成为影响软件交付的瓶颈所在,敏捷开发和DevOps思想理念的产生很好地解决了这一问题。

以迭代开发为特征的敏捷开发,灵活应对不断变化的需求,大大提高了开发效率。持续集成构建作为敏捷开发中的一项基本实践[1],将开发周期的集成阶段放到日常中,开发者每天都会集成代码,通过自动化的构建和测试尽早定位软件缺陷,降低开发风险。DevOps思想[2]是一组流程、技术和工具的统称,促进了开发、技术运营和质量保障部门之间的沟通协作,提供端到端的测试、交付和发布工作流。

各软件企业和相关研究人员对迭代式开发、持续集成等实践不断地研究尝试,促进了软件开发模式的发展,开发效率也在不断提升。虽然相关研究实践能够一定程度提高开发效率,但硬件资源的分配与限制是影响开发周期的另一个因素。本文的研究目的是将持续集成构建和容器技术相结合,应用在项目开发中,利用容器技术对硬件资源的分配特性,研究如何标准化环境从而重复、循环地交付软件价值。

本文总结国内外研究人员对敏捷开发中的持续集成实践和虚拟化容器技术的相关研究工作,分析传统持续集成系统的基本组成及其不足。提出将Docker技术应用在开发环节中,设计并实现一个容器化的持续集成构建系统(简称CCI),通过设计系统的一体化工作流方案,使各团队成员之间相互协作,优化软件开发流程。最后通过具体应用对系统平台进行分析和验证。

1 相关工作

1.1 持续集成的相关研究

不少国内外研究人员对敏捷开发中的迭代式开发和持续集成等实践展开深入的研究,促进了软件开发模式的转变和发展。

Lai和Leu[3]总结敏捷开发和持续集成的优势,提出一种持续集成过程以有效降低Web应用的开发风险。Meyer[4]论述实现一个持续集成系统的关键要素包括版本控制系统和持续集成服务器。使用版本控制系统,改变了开发人员之间的交流和协作模式,使不同团队之间可以更好地相互合作[5]。Seth和Khare[6]讨论版本控制系统在软件开发中的重要作用和分布式版本控制系统Git的优势。

Jenkins是一个开源的持续集成服务器,支持分布式构建。Rai等[7]讨论Jenkins的背景和发展历史,提供安装和配置Jenkins的方法,同时对比分析其他相关持续集成工具的优缺点。Jenkins不仅提供丰富的插件支持,有可扩展的对象模型,这一特性使得可以定制化地开发插件以扩展Jenkins的功能。另外,Jenkins工具除可以实现持续集成外,还可以实现持续交付。Armenise[8]提出Jenkins不仅能够提供自动化的构建实现持续集成,还可以通过丰富的插件机制实现产品的发布、部署,从而实现软件的持续交付。

1.2 虚拟化容器技术的发展

可重复性实验在科学领域中得到越来越多的关注[9],为解决实验环境的可复用性和实验数据的可再现性,出现了两个主流的解决方案:工作流(Workflow)软件和虚拟机[10-11]。但是工作流软件往往不能覆盖每个研究人员的需求,虚拟机相比工作流软件而言是一个好的方法,但是它也有自身的一些限制。虚拟机虚拟出一个完整的操作系统实现环境的隔离,并在此基础上配置相应的环境,系统维护性和灵活性比较差且不易扩展[10],不容易实现环境的可复现性。

轻量容器技术Docker的出现,引起业界很大的反响,受到广泛关注。它是一个开源的应用容器引擎,与传统虚拟机的主要不同之处在于它共享宿主机的Linux内核,启动快、占用资源少。Docker容器的虚拟化是建立在主机的操作系统之上的,是系统级的虚拟化而不是针对硬件虚拟化,虚拟镜像的移植性比传统基于硬件的虚拟化移植性要强。这使它比传统虚拟机轻量且具有高性能[12],与传统虚拟机分钟级的启动速度相比,Docker启动速度达到秒级。传统的计算机能够同时运行几个虚拟机,但却能够同时运行上百个甚至更多Docker容器。Boettiger[10]论述Docker在可重复性研究中发挥的重要作用,它解决了计算重复性所面临的技术挑战。在集群、云环境等分布式系统中使用Docker技术可以使底层应用环境保持一致性,增加系统的可移植性[13]。另外,Docker在持续集成和软件测试领域应用也很广泛。例如,与性能测试工具结合,保证实验环境的可复用性和数据的可再现性[14]。

基于容器的虚拟化技术,利用增量的方式生成镜像环境,灵活性更高。根据Docker的特性,

将其应用到软件项目的生命周期(开发、测试、部署、维护),能有效地提高各个环节的效率。

2　持续集成系统组成

持续集成(CI)系统的核心价值在于自动化,服务器的每次集成都是通过自动化的构建来验证,包括自动编译、测试、部署等。一个传统的持续集成系统通常有 3 个组成部分(如图 1):实现代码托管的版本控制系统,执行集成构建的持续集成服务器和自动构建编译的工具。

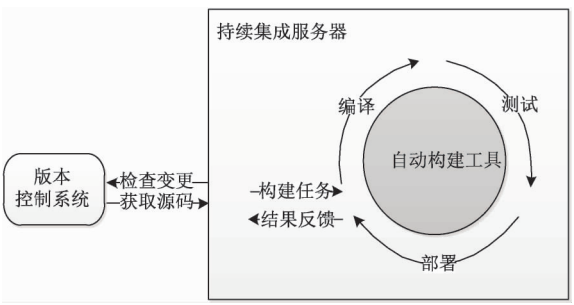

图 1　传统持续集成系统基本组成

Fig. 1　Basic components of traditional CI system

2.1　版本控制系统

版本控制系统在项目不同团队开发人员之间维护一个统一的代码库,实现项目的协同开发,保证项目的源代码处于有序的管理中,方便项目开发人员随时获取和提交变更。在持续集成系统中,版本控制系统起到的主要作用是源代码的管理。

2.2　持续集成服务器

持续集成服务器在整个系统中起到至关重要的作用,连通各个部分的基本组件,是实现过程自动化的关键。当开发者提交代码到版本库中,服务器轮询到变更,便会触发自动构建工具,进入到后续的自动化构建过程。这一自动化过程使开发人员能够快速收到 Bug 和相关故障的反馈通知,快速修复软件质量的缺陷,从而及时交付软件价值。

2.3　自动编译构建工具

在自动化编译构建工具引入之前,项目人员需要进行一系列手动的过程,下载源码、编译、测试、部署。手动构建不仅浪费大量的人力进行重复性的工作,而且容易引入人为因素导致的错误。

自动化的编译构建工具对于持续集成系统是不可或缺的,是实现系统自动化的前提。持续集成服务器集成构建工具完成构建任务,实现项目的自动化构建、测试、部署流程。

2.4　存在的问题

在传统的持续集成系统中,基本解放人力,实现了自动化,但仍然存在一些不足之处。

1)环境的搭建、维护比较复杂且耗时。持续集成环境、开发测试环境的搭建需要在本地完成种类繁多的软件安装和配置,是一件非常复杂且耗时的工作,且环境移植性比较差。

2)开发环境、测试环境和生产环境不一致。维护不一致的环境,影响软件的交付效率。另一方面容易导致在生产环境中暴露出开发测试环境没有出现的错误。

3)集成构建过程触发单元测试、集成测试,但没有考虑其他方面的测试,不方便测试人员开展工作。

3　CCI 容器化实现

3.1　系统架构

在本文的容器化实现中,采用 Github 作为版本控制系统,Jenkins 工具作为持续集成服务器。与传统集成平台相比,将安装在宿主机本地或虚拟机中的集成环境进行容器化,将集成服务器主从节点分别放在不同的容器环境中。利用 Jenkins 的分布式特性,将 Jenkins 主服务器(Master 节点)和 Jenkins 从服务器(Slave 节点)分别放在 Docker 容器中,CCI 系统架构如图 2 所示。

Jenkins 主机器负责任务(job)的配置及相关调配,从版本控制系统中获取代码变更来触发构建,根据对构建任务的相关配置,将任务分发到持续集成从机器中执行构建。可以根据项目需要,配置不同的从机器环境执行构建,保证任务构建测试环境的可扩展性。将配置好的集成构建环境生成镜像保存到镜像库中,保证构建环境的快速可再现性和可移植性。

3.2　具体实现

3.2.1　Jenkins 服务容器化

在 CCI 系统的实现中,关键是将集成构建环境进行容器化,利用 Docker 容器技术快速生成软件构建环境。为了生成 Jenkins 容器,Docker 提供两种创建方式:

1)利用现有的镜像库生成,使用 docker run 命令快速创建 Jenkins 容器。在 Docker 命令中将容器内部的 8080 端口重定向到宿主机的 8080 端口,可以通过主机端口访问容器内部 Jenkins 服务。

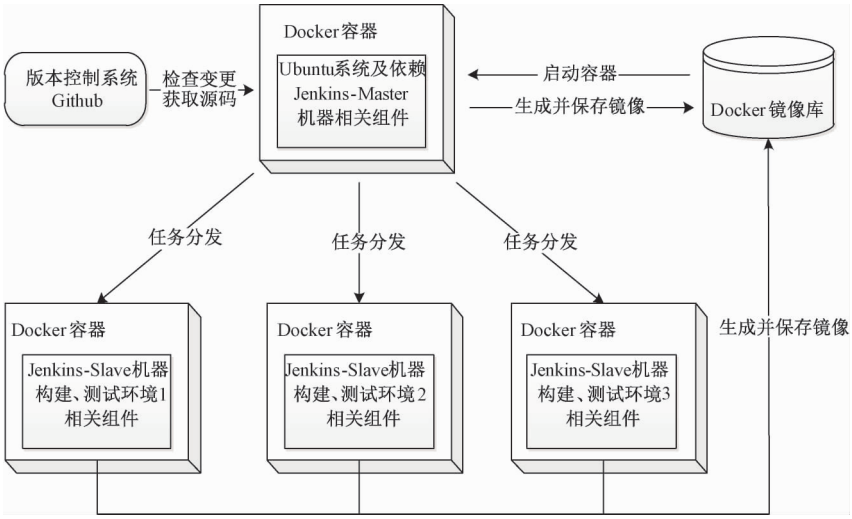

图 2 CCI 系统架构

Fig. 2 Architecture of CCI system

docker run-d-p 8080:8080-name zzcjenkins jenkins:latest

运行 docker run 命令时,如果本地没有镜像,则会默认从远端镜像仓库中拉取镜像,将镜像缓存到本地,再启动基于镜像创建的容器。

2)使用 Dockerfile 自定义的构建 Jenkins 镜像环境。Dockerfile 包含创建镜像所需要的全部指令,使用 docker build 命令来执行命令并创建镜像,自底向上地打包软件及其环境。

两种方式各有优缺点。利用现有镜像库方式生成容器,能够快速获取现有镜像生成可运行的容器环境,但是不方便构建自定义的环境。而通过 Dockerfile 的方式,可以减少镜像和容器的创建过程以简化部署,方便系统的升级。另一方面,在具体项目应用中,开发测试人员和运维人员可以利用 Dockerfile 文档来沟通软件项目的执行环境,促进了不同开发团队之间环境的统一。

使用 Dockerfile 构建拥有更高的灵活性和可维护性,本文的 Jenkins 容器环境的实现利用 Dockerfile 文档自定义构建 Jenkins 环境。在 Dockerfile 文档中实现 Jenkins 及其插件、版本控制服务 git 及其他相关依赖软件的安装配置。

Jenkins 容器构建完成之后,需要进行相关任务配置以运行构建任务。为了解决 Docker 中数据的安全性问题,保证项目数据的安全,我们不在容器中存放任何与项目相关的数据。Jenkins 在运行构建时,相关配置和数据都放在 $ JENKINS_HOME 下,通过 Volume 技术将 Jenkins 的运行目录挂载到宿主机的/var/lib/docker 目录下。

3.2.2 构建配置从机器

对于一个持续集成工具而言,Jenkins 的分布式特性,给我们带来很大的便利。在 Jenkins 分布式架构中,主机器主要负责任务的自动触发及调度分配,实际的构建任务在从机器上执行,保证干净的构建环境和整体性能。我们把从节点的环境也放到容器中,与主机器相似,从机器容器中也不能存放项目数据,利用 Volume 技术将容器环境中的目录挂载到宿主机。

本文中,Jenkins 从机器负责任务的构建和测试,从机器通过 Docker 技术虚拟出不同的构建环境执行主机器上的构建任务。通过这种方式构建出不同的从机器环境,使构建、测试环境多样化,更好地保证软件质量。本文通过第一种方式执行 docker run 命令直接在镜像库中获取基础镜像,使用 docker exec 命令进入到容器内部进行相应的环境安装配置操作。

3.2.3 任务的分发

Jenkins 分布式构建机制使同一项目或代码能够在不同的环境或系统中进行编译、部署,主机器和从机器的关联通信主要有 3 种方式,通过 ssh 启动连接、通过 java web 启动连接、使用命令行方式启动连接。

针对容器环境,在本文中采用 SSH 机制连接不同机器的通信,减少配置人员的工作量。

3.3 系统工作流

对于一个持续集成系统平台而言,关键是根据项目需求进行相关的集成配置,将流程进行自动化。从开发人员提交代码的那一刻,中间过程

通过自动化实现,而无需人工参与。

本文设计了 CCI 系统的主要工作流(如图 3 所示)。工作流中的主要参与人员有开发者、测试人员、部署人员,系统组成要素有版本控制系统 Github、Jenkins 服务器节点(Docker 容器)、Jenkins 从服务器节点(Docker 容器)和 Docker 镜像库。

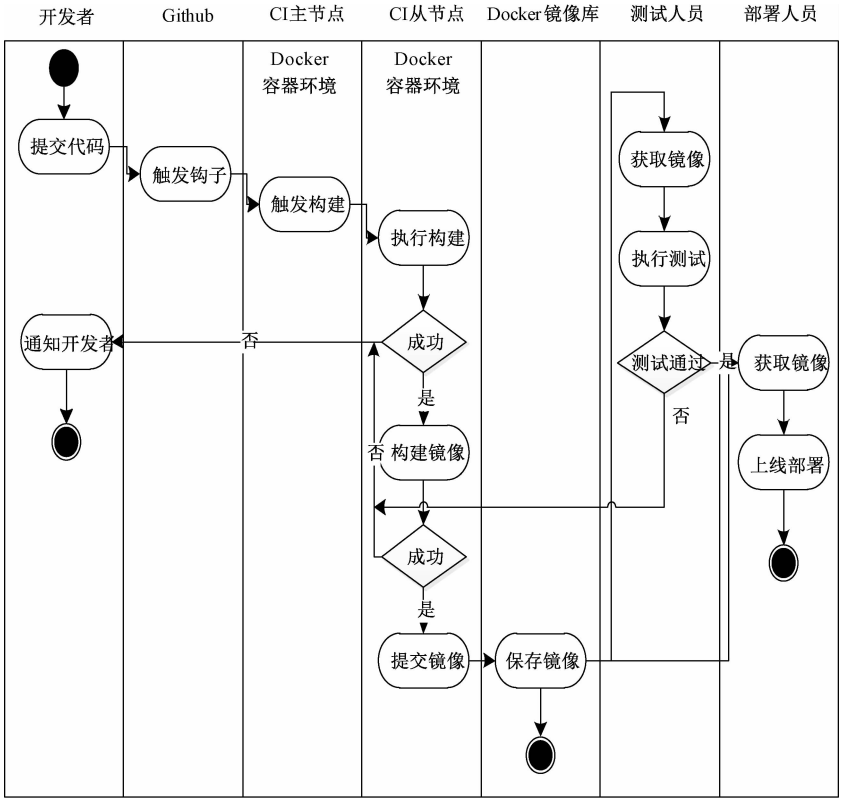

图 3　CCI 系统工作流

Fig. 3　Flow chart of CCI system

版本控制系统与容器中的持续集成主服务器 Jenkins 之间配置了自动触发的钩子,开发人员将代码提交到版本控制系统 Github 上之后,自动触发集成服务器进入到构建过程。Jenkins 服务器的主节点轮询版本控制系统,发现变更,将任务分发到 Jenkins 从机器节点上执行构建。在构建过程中,配置相关的测试脚本,执行自动化的单元测试。若项目构建成功,则将容器环境生成镜像,保存到 Docker 私有镜像库中。测试人员和运维部署人员在私有镜像库获取镜像,进行测试环境和线上环境的搭建,继而执行其他相关操作。通过共享镜像的方式,统一不同团队之间的环境,有利于快速生成软件价值,实现持续交付。

3.4　优点分析

传统的持续集成平台是将 Jenkins 安装在主机或虚拟机上,本文通过利用 Docker 将持续集成平台进行容器化,可以得到以下几点好处。

1)Jenkins 服务器的容器化提高系统平台的可重用性和可移植性。

2)将 Jenkins 从节点容器化,提供多样的构建环境和测试环境。

3)维护一致的开发、测试和部署环境,减少因环境不一致所造成的时间成本。

4　系统验证

4.1　实验环境

本文的基础实验环境主要有 3 个,分别为宿主机环境、CCI 系统容器环境和虚拟机环境,各环境操作系统及主要配置如下:

1)宿主机的操作系统采用 ubuntu14.04 版本,分别安装了 docker、git、jdk、vmware 软件环境;

2)CCI 系统包括 3 个容器环境,分别是 Jenkins 容器环境和 2 个从节点容器构建环境(slave1-ubuntu,slave2-centos);

3)在 vmware 虚拟机环境中,生成 3 台虚拟机,分别作为 Jenkins 服务器机器和 2 个从节点机器环境。

4.2 目的及方法

本实验主要有两个验证目标:验证 CCI 容器系统及本文所提出的工作流方案的可行性;验证系统资源占用少、快速构建部署的优势。

为了验证系统平台在软件开发各阶段中发挥的重要作用,我们在容器化的 Jenkins 系统中执行相关集成构建任务测试其可行性。容器内部的 8080 端口重定向到宿主机的 8080 端口,通过宿主机端口访问容器内部 Jenkins 服务,系统界面图如图 4 所示。

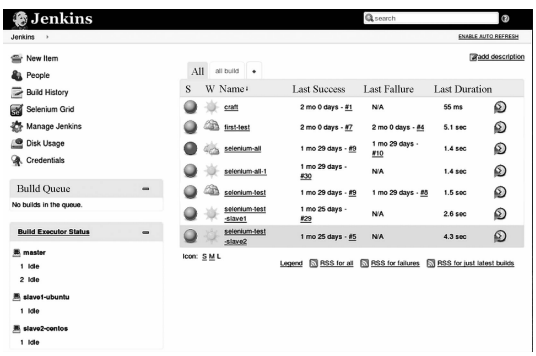

图 4 容器内的 Jenkins 服务

Fig. 4 Jenkins services in Docker container

我们利用此系统平台分别执行基于 Tomcat 的 Web 项目和对 Web 应用进行测试的 Selenium 测试脚本。测试脚本目的是测试应用的系统平台兼容性,对中国科学院仪器设备共享管理(V3.0)系统[15]的镜像进行测试,分别在容器中的两个不同操作系统环境(slave1-ubuntu,slave2-centos)中执行测试构建。

另一方面,为了验证 CCI 系统资源占用少、快速构建部署的优势,我们分别在虚拟机和 Docker 环境中安装配置了基于 Jenkins 的持续集成构建环境,记录安装部署的时间耗时。

4.3 结果分析

实践应用表明,基于 Docker 的 Jenkins 持续集成平台给开发活动中的各阶段都带来了便利。我们在系统上执行的基于 Tomcat 的 Web 应用程序,从源码的获取到测试构建,能快速地得到应用的部署结果,并将结果反馈给开发人员,极大地方便了开发活动。针对应用的测试而言,Selenium 测试脚本在两台从节点容器环境上执行构建的时间花费如图 5 所示,可以通过系统日志观察到在不同操作系统环境上的测试构建趋势和执行状态,方便测试人员定位兼容性错误。

从开发人员的角度来说,开发者可以利用

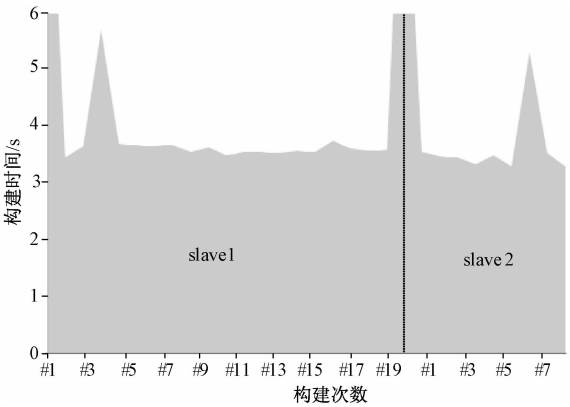

图 5 Selenium 测试脚本的执行结果

Fig. 5 Selenium script execution results

Docker 快速搭建开发环境分发给团队成员,并利用容器化 CCI 持续集成系统平台,快速得到项目的反馈,提高开发效率。从测试人员出发,利用容器化 CCI 持续集成平台,可以快速构建出不同的测试环境,通过分析在不同测试环境下得到的测试结果对软件质量进行评估。对于运维人员来说,能够快速搭建与生产环境一致的平台对系统进行评估和维护。

为了验证容器化系统相较于本地或虚拟机环境下的优势,分别在虚拟机和 Docker 环境中安装配置基于 Jenkins 的持续集成构建环境。根据实际安装测试,在虚拟机或本地宿主机上搭建持续集成系统,往往需要耗费 1 ~ 2 天的时间,需要分别安装底层操作系统、Jenkins 服务器及相关依赖和 Slave 从节点机器环境等各种复杂的配置。而用 Docker 在 10 s 左右就能快速生成一个可运行的持续集成容器环境,基于 docker 构建环境的时间与虚拟机相比几乎可以忽略不计,搭建环境的效率提升 90% 以上,系统启动时间的效率提高大约 80% 。表 1 通过各个指标对比 CCI 系统与虚拟机持续集成系统。

表 1 容器化 CI 和虚拟机 CI 的对比

Table 1 Comparison between CCI and virtual machine CI

	CCI 系统	虚拟机 CI 系统
环境搭建	15 ~ 20 s	1 ~ 2 d
启动时间	2 ~ 5 s	30 ~ 50 s
测试效率	环境可复用,重复率低	重复率高
移植性	5 s 完成环境的分发	不易移植

基于 Docker 的持续集成平台,在环境搭建耗时上远远优于传统的持续集成系统的搭建,只需要在秒级的时间里快速启动一个镜像就得

到一个持续集成环境。Docker 占用系统资源少,启动速度快,这点是虚拟机没有办法比拟的。在测试效率方面,容器化保证开发测试环境的统一和环境的重用性,从而减少测试重复率,提高测试效率。

5　结语

本文利用 Docker 实现一个容器化的 CCI 持续集成系统,提出代码管理、持续集成和持续交付的一体化工作流方案。将搭建环境的复杂度降低九成,确保环境的一致性和可移植性,有效促进开发、测试和运维团队之间的协作,能够大大提升软件开发效率。目前本文实现的集成系统还没有实现容器的管理和部署编排工具,缺乏可视化的用户管理界面。在下一步的研究工作中,将此系统框架迁移到私有云环境上进行实验,将云平台与容器技术相结合,在云环境下进一步验证系统的可靠性,并深入研究容器相关管理工具的应用,进一步开发可视化的管理界面。

参考文献

[1] Martin R C. Agile software development: principles, patterns, and practices[M]. New Jersey: Prentice Hall PTR, 2003.

[2] Erich F, Amrit C, Daneva M. A mapping study on cooperation between information system development and operations[C]//International Conference on Product-Focused Software Process Improvement. Springer International Publishing, 2014: 277-280.

[3] Lai S T, Leu F Y. Applying continuous integration for reducing web applications development risks[C]//Broadband and Wireless Computing, Communication and Applications (BWCCA), 2015 10th International Conference on. IEEE, 2015: 386-391.

[4] Meyer M. Continuous Integration and Its Tools[J]. IEEE Software, 2014, 31(3): 14-16.

[5] Vasilescu B, Van Schuylenburg S, Wulms J, et al. Continuous integration in a social-coding world: empirical evidence from GitHub[C]// Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on. IEEE, 2014: 401-405.

[6] Seth N, Khare R. ACI (automated Continuous Integration) using Jenkins: key for successful embedded software development[C]// Recent Advances in Engineering & Computational Sciences (RAECS), 2015 2nd International Conference on. IEEE, 2015: 1-6.

[7] Rai P, Dhir S, Garg A. A prologue of JENKINS with comparative scrutiny of various software integration tools[C]// Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on. IEEE, 2015: 201-205.

[8] Armenise V. Continuous delivery with Jenkins: Jenkins solutions to implement continuous delivery[C]// Release Engineering(RELENG), 2015 IEEE/ACM 3rd International Workshop on. IEEE, 2015: 24-27.

[9] Peng R D. Reproducible research in computational science [J]. Science, 2011, 334(6 060): 1 226-1 227.

[10] Boettiger C. An introduction to Docker for reproducible research[J]. ACM SIGOPS Operating Systems Review, 2015, 49(1): 71-79.

[11] Howe B. Virtual appliances, cloud computing, and reproducible research[J]. Computing in Science & Engineering, 2012, 14(4): 36-41.

[12] Merkel D. Docker: lightweight linux containers for consistent development and deployment[J]. Linux Journal, 2014, 2014 (239): 2.

[13] Zheng C, Thain D. Integrating containers into workflows: a case study using Makeflow, Work Queue, and Docker[C]// Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing. ACM, 2015: 31-38.

[14] Ferme V, Pautasso C. Integrating faban with docker for performance benchmarking[C]// Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering. ACM, 2016: 129-130.

[15] Zhang Z C, Luo T J, Yang L, et. al. Performance enhancement for IBBS@CAS[J]. Journal of University of Chinese Academy of Sciences, 2016,33(5):693-702.