华章IT

云计算与虚拟化技术丛书

PEARSON　vmware PRESS

DevOps for VMware Administrators

DevOps实战

VMware管理员运维方法、工具及最佳实践

［美］小特雷弗 A. 罗伯茨（Trevor A. Roberts, Jr.）乔希·阿特韦尔（Josh Atwell）
埃格勒·西格勒（Egle Sigler）依弗·范·多恩（Yvo van Doorn）著
姚军 译

VMware资深专家撰写，是第一本写给VMware管理员的DevOps权威指南

既系统介绍DevOps的基础概念和流行的工具，又详细讲解改变管理系统和交付服务的方法，涵盖DevOps环境配置、维护、编排、管理的各个环节，包含大量实例

机械工业出版社
China Machine Press

云计算与虚拟化技术丛书

DevOps 实战：VMware 管理员运维方法、工具及最佳实践

DevOps for VMware Administrators

［美］ 小特雷弗 A. 罗伯茨（Trevor A. Roberts Jr.）乔希·阿特韦尔（Josh Atwell）
埃格勒·西格勒（Egle Sigler） 著
依弗·范·多恩（Yvo van Doorn）
姚军 译

HZ BOOKS
华章图书

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

DevOps 实战：VMware 管理员运维方法、工具及最佳实践 /（美）罗伯茨（Roberts, T. A.）等著；姚军译 . —北京：机械工业出版社，2016.1
（云计算与虚拟化技术丛书）
书名原文：DevOps for VMware Administrators

ISBN 978-7-111-52478-6

I. D…　II. ①罗…　②姚…　III. 虚拟处理机　IV. TP338

中国版本图书馆 CIP 数据核字（2015）第 303916 号

本书版权登记号：图字：01-2015-5789

Authorized translation from the English language edition, entitled *DevOps for VMware Administrators,* 9780133846478 by Trevor A. Roberts Jr., Josh Atwell, Egle Sigler, Yvo van Doorn, published by Pearson Education, Inc., Copyright © 2015.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Chinese simplified language edition published by Pearson Education Asia Ltd., and China Machine Press Copyright © 2016.

本书中文简体字版由 Pearson Education（培生教育出版集团）授权机械工业出版社在中华人民共和国境内（不包括中国台湾地区和中国香港、澳门特别行政区）独家出版发行。未经出版者书面许可，不得以任何方式抄袭、复制或节录本书中的任何部分。

本书封底贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

DevOps 实战
VMware 管理员运维方法、工具及最佳实践

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：关　敏　　责任校对：董纪丽
印　　刷：　　版　　次：2016 年 1 月第 1 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：17.25
书　　号：ISBN 978-7-111-52478-6　　定　　价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

版权所有 • 侵权必究
封底无防伪标均为盗版
本书法律顾问：北京大成律师事务所　韩光 / 邹晓东

The Translator's Words 译 者 序

互联网已经成为世界经济的焦点区域，越来越多的 Web 服务成为企业日常经营活动的核心，这对 IT 运营和开发都形成了巨大的挑战。

在传统的瀑布方法中，开发各阶段与运营之间是弱耦合的，开发阶段搜集的需求几乎不考虑运营，这也决定了在运营阶段发现的问题将成为久拖不决的痼疾，而且，传统方法无法适应 Web 服务频繁的更新换代。在此背景之下，以敏捷技术为代表的新型开发方法大行其道，同时，由于运营在业务中的地位越来越突出，IT 与运营之间的“竖井”必须打破，于是 DevOps 应运而生。

DevOps 的核心思想之一便是 IT 与运营团队的相互融合，运营人员参与开发全过程，指导开发人员生成运营友好的应用程序；开发人员参与运营环节，了解运营需求并及时消除代码中不利于运营的缺陷。在这种指导思想下，实现了开发环境、测试环境和生产环境的高相似度，可编程、自动化的全生命期配置维护管理，从而适应现代 Web 环境的高变动性以及高可用性、高可靠性要求。

虚拟化技术是现代云环境的基石，由于基础设施在很大程度上实现了软件化，自动化配置维护比以前更便于实现，而且硬件设施的安装、配置和维护也变成软件代码的一部分，可以像开发产品一样置于版本控制之下，因此，虚拟化技术与 DevOps 的有机整合成为令人期待的趋势。

本书由 VMware 的资深专家编著，系统介绍了 DevOps 的基础概念和流行的工具，这些工具包括最流行的第三方工具（如 Vagrant、Chef、Ansible、Razor、Docker、Microsoft PowerShell 等）和 VMware 自身提供的持续集成、交付和部署产品（如 VMware vRealize Automation），涵盖了 DevOps 环境配置、维护、编排、管理的各个环节，书中介绍的大量例子可以帮助读者快速了解工具的概念、使用以及如何与 VMware 虚拟化环境结合。对于想要试水 DevOps 的读者来说，这是一本不可多得的参考书。DevOps 是一个快速变化的领域，有

了这本入门书籍，读者有望跟上它的脚步，并在实践中不断提高，建设属于自己的理想世界。在翻译过程中，我们也深深感受到短短十年之间技术和理念变化带来的震撼。由于译者水平所限，书中难免存在不足，希望读者谅解并提出宝贵意见。

本书的翻译工作主要由姚军完成，徐锋、刘建林、陈志勇、宁懿、白龙、陈美娜、谢志雄、方翊等也为翻译工作做出了贡献，在此衷心感谢机械工业出版社的编辑关敏老师和其他有关人员为本书所提的宝贵意见。

译者

HZ BOOKS
华章图书

Preface 前　言

什么是 DevOps？是可以从供应商那里买到，解决所有 IT 问题的产品吗？是分析师用来引起 CIO 注意的行业流行词吗？虽然 IT 社区对 DevOps 的介绍接近于大肆追捧，但那更多的是因为 DevOps 确实能够带来好处，而不仅仅只是行业的广告宣传。

DevOps 这一术语指的是一组帮助各种规模的组织更快地从 IT 投资中获得价值的方法、理念和工具。这个词的确切含义是什么？

想象一下，在你的组织中为了将软件项目从概念阶段、软件开发一直推进到生产部署，需要多少时间和过程？这个过程越长，IT 组织向整个公司展示价值所需的时间就越长。由于技术无处不在，客户期待 IT 服务的交付像移动应用商店那么容易。他们不愿意为了一项功能的实现而等待数年，对客户的要求反应迟钝的公司难以获得长期的成功。

DevOps 如何解决客户交付速度问题？例如，配置管理技术可以避免服务器配置漂移，加速在线购买新服务器处理客户请求快速增长的过程。持续集成可以确保自动化测试在开发者提交源代码时进行。这只是本书所讨论技术的两个例子。

网络规模 IT 组织（如 Etsy、Netflix 和 Amazon Web Services）被视为 DevOps 的典范。但是，Gene Kim 的 DevOps 企业峰会参与者的数量证明 DevOps 也能给传统 IT 组织带来价值。

所以，做好思想准备，DevOps 正在来临。好消息是，你可以为所在 IT 组织 DevOps 行动的成功做出贡献。本书的目标不仅是介绍 DevOps 的概要思路，还将提供 DevOps 工具和技术的实例。

关于本书

在我们的经验中，DevOps 的概念和工具可以显著地改进 IT 运营。虽然 Amazon 和 Rackspace 等大型 IT 组织已经在它们的环境中实施 DevOps 并取得成效，但是许多企业级 IT 组织对 DevOps 实践仍处于熟悉阶段。

本书的目标是为读者提供上述 IT 组织获得成功所借助的 DevOps 工具的实操示例。

本书的读者

本书是为具备 VMware vSphere 虚拟化管理程序（hypervisor）和 Linux 操作系统使用经验的系统管理员所写。我们将循序渐进地介绍 DevOps 从业者所使用软件解决方案的使用方法，每章都提供后续研究所需的额外资源。

本书内容

本书介绍的主题从虚拟化专业人士如何获得 DevOps 实践知识的概述开始，然后讨论 DevOps 从业人员使用的各种工具。

第 1 章讨论 DevOps 的概念，包括这一术语的定义以及 DevOps 相关实践有助于 IT 组织成功的原因。

第 2 章介绍 DevOps 从业人员使用的一些流行工具。第 3 章准备建立测试环境，以使用本书中的示例代码。

第 4 ~ 6 章介绍 Puppet 配置管理解决方案，包括简介、多层次应用部署，以及 Puppet 与 VMware vSphere 服务器和虚拟机管理集成的介绍。

第 7 ~ 9 章介绍 Chef 配置管理解决方案，包括简介、常见系统管理任务，以及 Chef 和 VMware vSphere 环境管理集成的介绍。

第 10 章和第 11 章介绍 Ansible 配置管理和编排解决方案，包括这种技术和各种应用程序部署的基本知识。

第 12 章介绍 PowerShell 预期状态配置（PowerShell Desired State Configuration，DSC）的基础知识，包括 Microsoft Windows PowerShell 这一新功能的架构和主要用例。为了阐述 DSC 的基本功能、解释组成该功能的不同组件，提供了样板代码。

第 13 章探索 VMware 管理员在其环境中实施 PowerShell DSC 的方法。本章包括专门针对 VMware 管理员（可能不是 Windows 系统管理员）使用 DSC 提供额外价值及能力的用例。本章讨论了不同的方法，相应地强调和讨论了每种方法的建议和局限。

第 14 章讨论对企业 IT 组织来说相对新颖的一种应用程序部署范型：Linux 容器的使用。本章用实操示例讨论 Docker 容器管理系统的基础知识。

第 15 章进一步讨论 Linux 容器，介绍 Google Kubernetes，这是一种在数据中心大规模管理容器的开源工具。

第 16 章描述如何安装、配置和使用 Razor——一种全生命期自动配给工具，组合了安装、

服务器管理和配置工具。

第 16 章详细介绍 Razor 的所有关键概念和组件，首先描述 Razor 的工作原理和入门使用方法。一旦了解了 Razor 的概念和结合 DevOps 工具用于自动化配给的方法，你就能够发现 Razor 的不同功能组件。最后，本章介绍了 Razor 的最优安装和配置方法。

第 17 章介绍 Elasticsearch、Logstash 和 Kibana（ELK）栈。这些工具都可以单独使用，但是结合使用可以成为日志管理的完美组合。本章单独介绍每一种工具，以及如何组合它们、最大限度地利用它们的能力提升日志管理的效率。

第 18 章介绍用于持续集成的 Jenkins，讨论如何在代码提交到源代码库之后自动交付。

第 19 章讨论 VMware 自身的 DevOps 倡议，包括 VMware vRealize Automation 与 DevOps 工具的集成，以及新的 VMware vRealize Code Stream 解决方案。

HZ BOOKS
华章图书

致　谢 *Acknowledgements*

许多人对本书给予了帮助，我要感谢他们在任务完成中对我们的直接和间接影响：

感谢 Gene Kim 在忙于自己的著作（《 The DevOps Handbook 》）和 DevOps 企业峰会规划工作时抽出时间，指导本书的内容和写作过程的多个方面。

感谢 Nick Weaver 通过 Razor 方面的作品向 VMware 社区介绍 Puppet，开启了我的 DevOps 之旅。

感谢 VMware 出版社的 Joan Murray，他的有力支持推动了本书的写作。

感谢 Kelsey Hightower 在 Linux 容器及其大规模协调方面提供的专业知识。

感谢 Aaron Sweemer 提供了 VMware 内部的联络人，和本书的读者分享公司的 DevOps 愿景。

感谢我的合著者，感谢他们对我所领导的这一书籍项目的耐心和持续支持。

感谢 Scott Lowe、Nick Silkey 和 Matt Oswalt 为本书内容提供的宝贵反馈。

——Trevor Roberts，Jr.

我要感谢在写作我自己的那部分内容时给我提供帮助的几个人。感谢 Don Jones、Steven Murawski 和 Alan Renouf 在我寻求 VMware 管理员可能从 PowerShell DSC 得到的益处时提供的重要指导。没有他们的深刻见解和观点，我可能仍然在实验室中苦苦思索。还要感谢 Trevor Roberts，Jr. 邀请我参加这个项目。最后，我要感谢 VMware 社区的大力支持和对本书的兴趣。希望你们和我一样喜欢这本书。

——Josh Atwell

感谢开源社区，没有你们，我就不能拥有这么出色和令人惊异的工具。

——Egle Sigler

首先，我要感谢 Trevor Roberts，Jr. 给我参与本书创作的机会。感谢《 Promise Theory: Principles and Applications》的合著者 Mark Burgess，Mark 在书中介绍了当今配置管理背后的科学知识，其中许多都是我们日常使用的。最后，我对 Chef 的每个人都心存感激，在 Chef 社区中，我才能迸发出许多灵感。

——Yvo van Doorn

HZ BOOKS
华章图书

关于作者 *About the Authors*

Trevor Roberts，Jr. 是 VMware 公司的高级技术市场经理。Trevor 拥有 CCIE 数据中心认证，是 VMware 数据中心设计和管理集中化认证高级专家。业余时间，Trevor 在 http://www.VMTrooper.com 通过 vBrownBag Professional OpenStack 和 Professional VMware 播客以及 Twitter（@VMTrooper）分享对数据中心技术的认识。他对 IT 社区的贡献得到公认，被授予 VMware vExpert、Cisco Data Center Champion 和 EMC Elect 的称号。

Josh Atwell 是 SolidFire 的云架构师，专注于 VMware 和自动化解决方案。10 年多的努力使他可以用少量代码通过各种自动化工具来完成自己的工作。Josh 已经有了两个儿子，2015 年年初，他和妻子 Stephanie 又生了一个女儿。他住在北卡罗来纳州的罗利，享受着和家人在一起的时间，他还喜欢高尔夫、有声读物和新的波本威士忌。Josh 是虚拟化社区的活跃分子，是 CIPTUG、VMUG 和 UCS 等技术用户组的领导人，而且还和其他人一起合作，准备通过 vBrownBag 播客和虚拟设计大师竞赛追求专业上的发展。Josh 还经常发表公开演讲，是 Mastering vSphrer 系列丛书的作者。他从不吝啬发表意见，在 vtesseract.com 上撰写博客，在 Twitter（@Josh_Atwell）上也是三句话不离本行。

Egle Sigler（@eglute, anystacker.com）现为 Rackspace 的首席架构师。她在职业生涯初期是一位软件开发人员，至今仍有着所有编写、测试和部署代码的人所具有的弱点，因为她有机会从事所有这类工作。Egle 的梦想是有朝一日，编写、测试和部署代码将成为无缝、轻松的过程，完全没有缺陷和挫折。Egle 坚信，知识应该共享，并通过撰写本书、发表讲话和会议上的探讨以及博客努力实践。

Yvo van Doorn 有 10 多年的系统管理经验。在职业生涯初期，他人工构建和配置“裸”服务器。在同辈人中，Yvo 成为配置管理和虚拟化的冠军。加入 Chef 之前，他在将西雅图一家小型技术公司的整个生产系统迁移到虚拟化平台时亲身见证了 VMware 产品的威力。他坚信 DevOps 所带来的文化变迁。在不忙于传播 Chef 的福音时，他可能会享受醉人的 IPA 啤酒，探索好的户外运动或者继承自己的荷兰传统，一边吃着高达干酪，一边观看橙衣军团丢掉世界杯。Yvo 和妻子及黑色的赖伯犬一起住在华盛顿州西雅图市。

About the Reviewers 关于评审人员

Scott Lowe，VCDX 39，在 IT 行业工作了 20 多年，目前是 VMware 的工程架构师，专注于网络虚拟化、开源和云计算的汇聚点。他还花时间参与了许多 DevOps 相关的产品和项目。

Randall“Nick”F. Silkey，Jr. 是 The Rackspace Cloud 的高级系统工程师。他热心于基础架构自动化和发行工程，喜欢组织得克萨斯州奥斯汀的几个专业技术组织。Nick 还在当地及全国性会议上发表关于持续集成及运营工程的演讲。工作之余，Nick 享受着和妻子 Wendy 及 3 个孩子在一起的时光。

Matt Oswalt 是一位全能的技术迷，目前主要关注在联网工作中引入自动化工具和方法。他进入 IT 界时在一家大型零售连锁企业担任应用程序开发人员。之后，他担任网络基础设施方面的顾问达 4 年之久。现在，他将这两方面的技能结合起来，建立更灵活、有弹性的网络基础设施。他常常参加自动化和 DevOps 社区，帮助推动网络自动化和 SDN 方面的交流。他经常在 keepingitclassless.net 上的个人博客以及 Twitter（@Mierdin）发表关于这一领域及传统基础设施的作品。

关于贡献者 *About the Contributing Author*

Chris Sexsmith 是本书的贡献者。Chris 在过去 4 年内担任 VMware 全球卓越中心的员工解决方案架构师，主要专注于自动化、DevOps 和云管理技术。Chris 住在加拿大不列颠哥伦比亚省的温哥华，在那里攻读 MBA，并尽可能地将其余时间用于观看冰球比赛。Chris 和他的团队领导 LiVefire 项目，专注于软件定义数据中心（SDDC）内专家和合作伙伴解决方案的实现。

HZ BOOKS
华章图书

Contents 目　录

译者序
前言
致谢
关于作者
关于评审人员
关于贡献者

第一部分　DevOps 概述

第 1 章　DevOps 简介 ····· 2
1.1　DevOps 原则概述 ····· 2
1.2　采用系统思维 ····· 3
1.2.1　改变团队的互动方式 ····· 4
1.2.2　改变基础设施部署方法 ····· 5
1.2.3　改变软件开发和部署方法 ····· 6
1.2.4　经常收集和响应有用的系统反馈并相应调整 ····· 6
1.3　增进 DevOps 知识和技能 ····· 6
1.4　小结 ····· 7
参考文献 ····· 7

第 2 章　DevOps 工具 ····· 8
2.1　为成功而组织：看板 ····· 8
2.2　服务器部署 ····· 11
2.3　配置管理 ····· 11
2.4　持续集成 ····· 12
2.5　日志分析 ····· 12
2.6　小结 ····· 12
参考文献 ····· 12

第 3 章　建立 DevOps 配置管理测试环境 ····· 13
3.1　用 AutoLab 进行环境配给 ····· 13
3.2　用 Vagrant 进行环境配给 ····· 14
3.3　用 Packer 创建映像 ····· 18
3.4　管理源代码 ····· 18
3.5　小结 ····· 24
参考文献 ····· 24

第二部分　Puppet

第 4 章　Puppet 简介 ····· 26
4.1　Puppet 架构 ····· 26
4.1.1　独立部署 ····· 27
4.1.2　主机 – 代理部署 ····· 27
4.2　准备 Puppet 测试实验室 ····· 28

HZ BOOKS 华章图书

4.3 Puppet 资源 ····· 29
4.4 Puppet 清单 ····· 30
4.5 Puppet 模块 ····· 35
4.5.1 Puppet Forge ····· 37
4.5.2 创建第一个 Puppet 模块 ····· 37
4.5.3 Puppet 模块初始化清单（init.pp）····· 38
4.5.4 模板 ····· 39
4.5.5 使用 Puppet 模块 ····· 42
4.5.6 最后一步：版本控制提交 ····· 42
4.6 小结 ····· 42
参考文献 ····· 42

第 5 章 Puppet 系统管理任务 ····· 43
5.1 用数据分离优化 Web 层 ····· 43
5.1.1 参数类（params.pp）····· 45
5.1.2 Hiera ····· 48
5.1.3 节点分类 ····· 51
5.2 应用层 ····· 51
5.3 数据库层 ····· 53
5.4 实施生产建议措施 ····· 53
5.5 部署应用程序环境 ····· 54
5.6 小结 ····· 54
参考文献 ····· 54

第 6 章 用 Puppet 进行 VMware vSphere 管理 ····· 55
6.1 Puppet 的 VMware vSphere 云配给器 ····· 55
6.1.1 准备 VM 模板 ····· 55
6.1.2 准备 Puppet 主服务器 ····· 56
6.2 VMware 的管理模块 ····· 58
6.3 小结 ····· 63
参考文献 ····· 63

第三部分 Chef

第 7 章 Chef 简介 ····· 66
7.1 什么是 Chef ····· 66
7.2 Chef 的核心思想 ····· 67
7.2.1 食谱的顺序 ····· 67
7.2.2 幂等性 ····· 67
7.2.3 基于 API 的服务器 ····· 67
7.2.4 客户端进行所有搜集工作 ····· 68
7.2.5 测试驱动基础设施 ····· 68
7.3 Chef 术语 ····· 68
7.3.1 食谱 ····· 68
7.3.2 烹调书 ····· 68
7.3.3 属性 ····· 68
7.3.4 角色 ····· 68
7.3.5 运行列表 ····· 69
7.3.6 资源 ····· 69
7.3.7 环境 ····· 69
7.4 托管 Chef 和 Chef Server 之间的差别 ····· 69
7.4.1 托管 Chef ····· 69
7.4.2 Chef Server ····· 69
7.5 ChefDK 简介 ····· 70
7.5.1 ChefDK 是什么 ····· 70
7.5.2 安装 ChefDK ····· 70
7.6 使用 Knife ····· 72

HZBOOKS 华章图书

7.7 创建第一个“你好，世界”的 Chef 食谱 …… 73
7.8 小结 …… 76

第 8 章 使用 Chef 完成系统管理任务 …… 77

8.1 注册托管 Chef …… 78
8.2 社区烹调书 …… 81
8.3 设置系统管理 …… 81
8.3.1 准备 / 设置系统管理任务 1：管理时间 …… 82
8.3.2 准备 / 设置系统管理任务 2：管理根密码 …… 83
8.4 配置虚拟客户机 …… 84
8.5 系统管理任务 …… 86
8.6 管理根密码 …… 89
8.6.1 创建两个环境文件 …… 89
8.6.2 将环境文件上传到托管 Chef 组织 …… 90
8.6.3 为每个服务器分配一个环境 …… 91
8.6.4 修改每个服务器的运行列表，以运行 Managedroot 烹调书 …… 91
8.6.5 对节点应用更改 …… 92
8.6.6 校验实施的策略 …… 93
8.7 小结 …… 94
参考文献 …… 94

第 9 章 用 Chef 管理 VMware vSphere …… 95

9.1 Knife 插件 …… 96
9.1.1 knife-vsphere 入门 …… 97
9.1.2 配置 knife.rb 文件 …… 97
9.1.3 校验配置 …… 99
9.1.4 组合 …… 99
9.2 Chef 配给 …… 101
9.2.1 Chef 配给架构 …… 102
9.2.2 Chef 配给入门 …… 102
9.2.3 启动某些节点 …… 103
9.3 小结 …… 105

第四部分 Ansible

第 10 章 Ansible 简介 …… 108

10.1 Ansible 架构 …… 108
10.2 准备 Ansible 测试实验室 …… 109
10.3 Ansible 组 …… 110
10.4 Ansible 临时命令执行 …… 110
10.4.1 Ping 模块 …… 111
10.4.2 Command 模块 …… 111
10.4.3 User 模块 …… 111
10.4.4 Setup 模块 …… 112
10.5 Ansible 剧本 …… 112
10.6 Ansible 角色 …… 117
10.7 Ansible Galaxy …… 121
10.8 小结 …… 121
参考文献 …… 121

第 11 章 Ansible 系统管理任务 …… 122

11.1 Web 服务器部署 …… 122
11.2 应用层 …… 123
11.3 数据库层 …… 124
11.4 角色结构优化 …… 126
11.5 VMware 资源管理 …… 128

HZ BOOKS 华章图书

11.6 小结 …… 132
参考文献 …… 132

第五部分 PowerShell

第 12 章 PowerShell 预期状态配置简介 …… 134
12.1 什么是 PowerShell DSC …… 134
12.2 PowerShell DSC 需求 …… 135
12.3 PowerShell DSC 组件 …… 136
12.3.1 原生命令集 …… 136
12.3.2 托管对象格式文件 …… 136
12.3.3 本地配置管理器 …… 137
12.4 PowerShell DSC 配置 …… 138
12.5 PowerShell DSC 模式 …… 140
12.5.1 本地推送模式 …… 140
12.5.2 远程推送模式 …… 140
12.5.3 拉取模式 …… 141
12.6 PowerShell DSC 资源 …… 142
12.7 小结 …… 144
参考文献 …… 144

第 13 章 PowerShell DSC 实施策略 …… 145
13.1 PowerShell DSC 在 VMware 环境中的用例 …… 145
13.2 用 PowerCLI 进行脚本化 VM 部署 …… 146
13.3 在 VM 模板中加入 PowerShell DSC …… 148
13.4 对新 VM 实施 PowerShell DSC 配置所面临的挑战 …… 148
13.4.1 PowerCLI Invoke-VMscript …… 149
13.4.2 PowerCLI Copy-VMGuestFile …… 150
13.5 经验教训总结 …… 151
13.6 未来 PowerShell DSC 在 VMware 环境中的用例 …… 151
13.7 小结 …… 152
参考文献 …… 152

第六部分 利用容器进行应用程序部署

第 14 章 Docker 应用容器简介 …… 154
14.1 什么是应用程序 …… 154
14.1.1 隐藏的复杂性 …… 154
14.1.2 依赖性和配置冲突 …… 155
14.2 Linux 容器 …… 155
14.2.1 控制组 …… 155
14.2.2 命名空间 …… 156
14.2.3 容器管理 …… 157
14.3 使用 Docker …… 157
14.3.1 安装 Docker …… 157
14.3.2 Docker 守护进程 …… 158
14.3.3 Docker 客户端 …… 158
14.3.4 Docker 索引 …… 158
14.3.5 运行 Docker 容器 …… 158
14.3.6 列出运行的容器 …… 159
14.3.7 连接到运行的容器 …… 159
14.3.8 构建和分发 Docker 容器 …… 161
14.3.9 Dockerfile …… 161
14.3.10 Docker Hub …… 162

HZBOOKS 华章图书

14.3.11 Docker 与虚拟机的对比 ····· 163
14.3.12 Docker 与配置管理的对比 ····· 163
14.4 小结 ····· 163
参考文献 ····· 163

第 15 章 大规模运行 Docker 容器 ····· 164
15.1 容器编排 ····· 164
15.2 Kubernetes ····· 165
15.3 Kubernetes 部署 ····· 166
15.3.1 CoreOS 和 Kubernetes 群集管理工具 ····· 166
15.3.2 CoreOS 群集部署 ····· 167
15.3.3 etcd 服务器配置 ····· 171
15.3.4 Flannel 网络覆盖 ····· 172
15.3.5 Kubernetes 群集节点 ····· 172
15.3.6 Kubernetes 服务部署 ····· 174
15.3.7 Kubernetes 工作负载部署 ····· 175
15.4 用 Docker 实现平台即服务 ····· 178
15.5 小结 ····· 179
参考文献 ····· 179

第七部分 DevOps 工具链

第 16 章 使用 Razor 配给服务器 ····· 182
16.1 Razor 的工作原理 ····· 182
16.2 使用 Razor ····· 184
16.2.1 Razor 集合和操作 ····· 186
16.2.2 构建 Razor 集合 ····· 192
16.3 使用 Razor API ····· 201
16.4 Razor 组件 ····· 203
16.4.1 Razor 服务器 ····· 203
16.4.2 Razor 微内核 ····· 203
16.4.3 Razor 客户端 ····· 203
16.5 安装 Razor ····· 203
16.5.1 PE Razor ····· 204
16.5.2 Puppet 安装 ····· 204
16.5.3 从来源安装 ····· 204
16.5.4 人工安装发行版本 ····· 204
16.5.5 其他服务 ····· 204
16.6 小结 ····· 206
参考文献 ····· 206

第 17 章 ELK——Elasticsearch、Logstash 和 Kibana 简介 ····· 207
17.1 Elasticsearch 概述 ····· 207
17.1.1 入门 ····· 208
17.1.2 理解索引 ····· 208
17.1.3 使用数据 ····· 209
17.1.4 安装插件 ····· 212
17.1.5 使用客户端 ····· 214
17.2 Logstash 概述 ····· 215
17.2.1 入门 ····· 216
17.2.2 配置 Logstash 输入 ····· 216
17.2.3 应用过滤器 ····· 218
17.2.4 理解输出 ····· 219
17.3 Kibana 概述 ····· 219
17.3.1 共享和保存 ····· 223
17.3.2 自定义数据视图 ····· 223
17.4 小结 ····· 223
参考文献 ····· 224

HZ BOOKS 华章图书

第 18 章　用 Jenkins 实现持续集成 ···· 225

18.1　持续集成概念 ····· 225

18.1.1　持续集成还是持续部署 ····· 226

18.1.2　测试自动化 ····· 226

18.2　Jenkins 架构 ····· 227

18.3　Jenkins 部署 ····· 228

18.4　Jenkins 工作流 ····· 230

18.4.1　Jenkins 服务器配置 ····· 230

18.4.2　Jenkins 构建任务 ····· 232

18.4.3　Git 钩子 ····· 235

18.4.4　你的第一次构建 ····· 237

18.5　质量保证团队 ····· 239

18.5.1　验收测试 ····· 239

18.5.2　开发团队 ····· 239

18.5.3　构建 / 测试基础设施 ····· 239

18.6　小结 ····· 239

参考文献 ····· 239

第八部分　VMware DevOps 实践

第 19 章　DevOps 环境中的 VMware vRealize Automation ····· 242

19.1　DevOps 的出现 ····· 242

19.2　稳定的敏捷性 ····· 243

19.3　人、过程和 Conway 法则 ····· 243

19.4　vRealize Automation ····· 244

19.5　vRealize Application Services ····· 245

19.6　Puppet 集成 ····· 247

19.7　Code Stream ····· 252

19.8　小结 ····· 256

参考文献 ····· 256

HZ BOOKS
华章图书

第一部分 *Part 1*

DevOps 概述

HZ BOOKS
华章图书

- 第 1 章　DevOps 简介
- 第 2 章　DevOps 工具
- 第 3 章　建立 DevOps 配置管理测试环境

Chapter 1 第 1 章

DevOps 简介

DevOps 是一种系统部署方法学，组织可以用它来改善项目部署的速度和质量。它不仅仅只是另一个流行词，传统的 IT 组织正在认真研究 DevOps 实践和工具对它们实现目标带来的帮助，这就是一个明证。DevOps 只对云规模组织（如 Netflix 和 PayPal）有好处吗？当然不是，DevOps 实践对任何规模的组织都有好处。

本章包含如下主题：

- DevOps 原则概述
- DevOps 原则在组织中的实际运用
- 磨练 DevOps 知识和技能的重要性

1.1 DevOps 原则概述

DevOps 包含组织互动和部署工具及实践的变化，主要强调识别和缓解生产率瓶颈。你们当中的一些人可能阅读过 Gene Kim 所著的《The Phoenix Project》，书中他将 DevOps 的重要原则归结为 DevOps 的三条道路：

- 第一条道路：优化从开发到 IT 运营的工作流。
- 第二条道路：缩短和放大反馈循环。
- 第三条道路：鼓励试验，快速从故障中学习。

这些原则和 John Willis 及其他 DevOps 思想领袖所讨论的流行概念 CAMS（文化、自动化、计量和共享）相符：

- 文化：故障不应该立即引起过失认定，改变团队成员对部署方法和故障响应的思维方式。

- 自动化：人工方法容易招致故障。使用能够以可靠的方式重复、快速部署环境的工具。
- 计量：监控和分析对于成功必不可少，否则，故障的根源分析就不可能实现。
- 共享：个人 / 团队独占信息，以维持地位提升或者团队依赖性的“摇滚明星”心态在 IT 文化中站不住脚。这种心态不会加快生产速度，而会降低生产速度。

这些想法中，有些可能需要管理层的权力才能推动变化。那么，它们对于作为 IT 部门中虚拟化专家的读者来说意味着什么？你如何帮助引导这种变化？

虚拟化专家处于一个独特的地位，能够帮助开发团队和运营团队保持一致。在融合式基础设施的世界中，我们在规划和部署虚拟基础设施时已经熟悉了多学科（计算、网络、存储、安全等）的协调。这种专业知识对于协调组织中的开发和 IT 运营团队至关重要。

不管我们部署虚拟基础设施有多快，企业项目的成功将取决于从开发迁移到生产的效率。

想象一场接力赛：如果团队没有预先准备（回顾旧的影片、训练交接棒、力量训练等），在赛场上就肯定会犯错（掉棒、选手相互绊倒等）。IT 运营将会为误期而受到责备，“影子 IT”的使用将更加盛行。

在前一段中我强调“预先”，是因为当开发团队完成敏捷发行过程时，再启动部署工作往往已经太迟了。如果你的一位或者多位团队成员成为开发组织的顾问，在整个“冲刺”过程中都和他们在一起工作，而不仅仅在发行的末期，会怎么样呢？我们将以下面讨论的一个虚构项目作为背景，说明这些方法。

例如，假定你的公司 DevWidgets 决定发行一个软件即服务（SaaS）解决方案，包含其集成开发环境（IDE）和源代码管理（SCM）软件解决方案（Taao 项目）。目标是利用互联网，通过在线结对编程、代码评审网络会议等手段，培养地理上分散的团队成员之间的互动。

Taao 项目是高级领导层确认为公司旗舰产品的重大举措。有些高管人员曾经阅读过《The Phoenix Project》，他们希望对该项目实施“DevOps 工作”，因为他们对过去的发行安排感到不满。CEO 甚至考虑建立专门的 DevOps 团队，全部配属新雇用的人员（因为人数越多，项目总是推进得越快，对吗？）。

开发和 IT 组织的经理们努力使高级领导层保持镇静，请求领导给出时间，并承诺在当月月末提交一项计划。在这方面你能提供什么帮助？下面几节我们将探索你可能采取的措施。

1.2　采用系统思维

系统思维意味着将涉及软件发行版本部署的所有团队当成一个紧密相连的单位，而不是日程安排相互冲突的多个分散团队。这些团队包括信息安全、运营、开发、质量保证（QA）、产品管理等。

注意 在我们的讨论中所提到的系统指的是经历整个软件开发生命期（规划、开发、QA、部署和维护）的任何项目，不管这些项目是供内部还是外部消费的。

1.2.1 改变团队的互动方式

我们将焦点放在第一件应该做的事：成为开发团队的顾问。和负责定期规划会议的开发团队领导（在敏捷的术语中称为产品负责人和敏捷教练）对话，要求加入他们的一些回忆。主动倾听可能需要基础设施采购 / 部署的任何团队目标或者可交付成果。下面的几段强调来自开发团队的需求 / 规格示例，说明了在这种会议期间你所能提供的反馈 / 专业意见。

- “我们需要用于客户源代码的长期数据存储。”

开发团队不一定直接说出他们需要一个新的磁盘阵列，甚至没有意识到需要采购存储设备。但是，对存储系统的认识会提示你，有必要和存储主题专家（SME）一起进行容量规划。

你还需要在开发人员讨论用户数据生命期管理时考虑不同的存储层次。

- “我们需要在生产环境中运行独立于客户使用实例的 Taao 项目实例。”

这立即让你想起附加虚拟网络或者虚拟防火墙的需求，这能够使生产和客户数据流量与内部流量分隔开。你还需要咨询信息安全团队，了解提供客户数据分离保证的审计过程。

- “在我们的最后一个发行周期中，代码在我的便携电脑上成功编译和运行，但是在部署到生产环境时崩溃。”

开发人员可能提出一个在之前的代码部署中遇到的问题，提交的代码在她的便携电脑上工作正常，但是在生产部署时发生故障。她的本地开发环境运行的是 Ubuntu 虚拟机（VM），和非军事区（DMZ）中的 CentOS 服务器上实施的安全补丁或者防火墙规则不匹配。所以，需要立即打补丁以满足最后限期（对于开发团队和运营团队都是一个漫长的夜晚）。你对 Vagrant 和 Packer 等环境构建工具和 Puppet、Chef 及 Ansible 等配置管理工具的认识，使你能够构建一个用于开发人员便携电脑且匹配生产服务器规格的标准测试环境。

由于开发团队领导了解了运营团队在发行周期早期为了缓解开发环境问题所做出的努力，你在协调与此努力相关的一些活动时就会更容易一些。首先，这意味着开发团队需要确保其成员不会将任务推给其他团队。（也就是说，“现在是周五下午 5 点了，我不知道代码能不能在生产环境中正常工作，但是无论如何，我都会把它提交到存储库！”）

某组织在最近的 PuppetConf 上发表讲话，分享其代码部署策略：当代码部署到生产环境时，编写代码的开发人员参与更改窗口期的轮班。这样，如果代码中出现任何复杂现象，他们可以立即协助运营团队查错和解决。坦率地说，这也使开发人员有动力编写更好的代码，避免在凌晨 3 点接到电话去修复缺陷。

在继续之前，对于可能阅读本书的经理，Jez Humble 强烈建议：不要构建新的 DevOps 团队！雇用在 DevOps 方法学上有专业能力的开发人员、QA 工程师、运营人员是个好主意，

但是创建不同于组织其余团队的全新 DevOps 团队只会带来新的“竖井”，可能对你的目标造成反作用。相反，应该像前面提到的那样，继续更好地协调各个团队，使他们像一个团队那样思考和行动，而不是各自寻求自己的最大利益。

1.2.2　改变基础设施部署方法

数据中心的一些扰人而又常见的现象可能妨碍系统的进展：手工制作的“金映像”、雪花服务器和易碎箱。

服务器（不管是物理还是虚拟服务器）部署的常用方法是维护一组包含必要更新、补丁、设置等内容的操作系统配置，人工运用这些配置使系统立刻为部署做好准备。这些配置被称作“金映像”，传统上采用 ISO 形式，已经根据某个运行手册人工应用补丁。最近，金映像已经采用模板 VM 的形式保存。但是，老实说：金映像也可能生锈！

金映像本身没有什么问题。但是，构建它们的方式可能带来问题。如果人工构建，该过程可能是一个全职工作，必须跟踪模块更新、安全补丁等，然后重新配置 ISO/ 模板 VM 供以后使用。必须有一种更有效的方法，也的确有！

使用第 4 ~ 13 章介绍的配置管理技术，可以自动构建映像。随着服务器配置更改并登记到 Git（第 3 章中介绍）等存储库，Jenkins（参见第 18 章）等持续集成（CI）系统可以输出一个更新的金映像。CI 系统可以使用 Packer 等工具生成用于 Vagrant、虚拟化管理器和云提供商的模板 VM。

Martin Fowler 提出了“雪花服务器”的思路，雪花服务器是一台物理机器或者 VM，最初是为了保持标准化的良好愿望。但是，不管是为了完成故障报告而进行快速修复、高管要求的特殊项目还是其他原因，这种服务器都变成高度定制的。这些特殊更改解决了短期问题，但是会造成长期的麻烦。当对这些服务器上的软件包升级时会发生什么？安全补丁甚至操作系统升级又会发生什么？

如何解决雪花服务器的问题？首先，我们通过登录到服务器命令行界面强制避免任何更改；不允许一次性执行前端和软件包更新工具或者更改配置文件。所有服务器更改都只能通过配置管理技术进行。

对于现有的雪花服务器，必须进行一些乏味而不必要的工作。首先，检查服务器配置，以便准确地知道需要修改的配置文件、需要安装的软件包和其他需要包含在配置管理脚本中的关键设置。多次迭代构建和测试服务器，每次都对配置管理脚本进行调整，直到测试环境与雪花服务器完全相同。将更改提交到源代码库，就可以拥有一个可在生产服务器无法恢复时使用的可重复构建过程。

易碎箱这一术语来源于 Nassim Nicholas Taleb 的《 Antifragile 》一书，描述了一台运行关键软件栈的物理或者虚拟服务器，每个人都害怕接触它，因为业务可能因此遭遇重大故障。通过许多支持电话 / 专业服务，这台服务器已经达到了一个稳定状态，糟糕的是，管理该服务器的 SME 离开了公司。

如果稳定性确实是个考虑因素，进行物理 – 虚拟转换或者虚拟 – 虚拟迁移到 VMware vSphere 平台是个好主意。这样，你可以利用分布式资源调度器（DRS）、存储 DRS、高可用性（HA）等 VMware 基础设施可靠性机制。在服务器转换且拥有成功的业务持续性工作流（备份、快照）后，就可以考虑是否可以利用前面雪花服务器的弥补措施复制这一服务器。开发和 QA 工程师就可以很容易地构建一个测试环境而不接触易碎箱。

1.2.3 改变软件开发和部署方法

我们已经解决了基础设施部署的速度和质量问题，那么团队如何减少从开发、展示到生产阶段代码移动引起的缺陷？众所周知，在开发周期中越早发现缺陷，修复的代价就越低。这就是我们首先要有 QA 团队的原因。

但是，如果我们可以在代码移交给 QA 工程师之前就捕捉到缺陷，又会如何？这就是前面提到的持续集成系统的用途所在。我们后面会更详细地讨论 CI，基本要点是，当你将源代码提交到存储库时，CI 系统可以修改并设置为自动执行对提交代码的一系列单元测试。在过程结束时，可以构建一个软件包并自动分发给 QA 团队，实施他们的全面测试。

1.2.4 经常收集和响应有用的系统反馈并相应调整

系统思维的转变只有在有能力监控和分析系统性能时才能成功。业务的变化节奏似乎是指数级的，消费者对系统响应能力和正常运行时间有更高的预期，被动的问题解决方案不再成为选择；相反，你的团队必须在问题发生之前预测到它们，以维护系统稳定性。

日志分析可能是最重要的工具之一，尤其是在启用了二进制代码的调试选项时。如果你的环境由较多的服务器组成，就应该采用某种形式的自动化日志分析。这方面的选择很多，包括 VMware vRealize Log Insight、Splunk 甚至 Logstash、Elasticsearch 和 Kibana 等开源工具（第 17 章中介绍）。这些解决方案使系统管理员能够检查重复的事件与低下的性能关联。当团队更加熟悉这些工具，且更擅长识别问题时，系统的实用性就会提升。

1.3 增进 DevOps 知识和技能

一旦在系统思维和改进的系统验证上构建了很好的基础，团队对定期试验新功能就会更加自信。DevOps 实践使开发人员能够分阶段逐步地投产各种功能。这样做的好处之一是，开发人员可以利用针对选择的客户的限定发行版本对新功能进行 Beta 测试，收集基础设施影响和用户接受度方面的指标。有效的日志分析方法能够在重大问题蔓延之前发现它们。

持续的学习和改进不只是用于新产品功能的部署：持续发展自己的技能、扩展知识集合也很重要。除了跟上 VMware 技术的最新、最杰出的发展之外，还要积极学习有利于和开发人员进行更有意义互动的新技能。学习公司使用的编程语言的基础知识，培养对敏捷过程的

兴趣。和公司开发人员的沟通越有效，项目从开发到生产的迁移就越顺畅。

积极增进对“快速流程”的理解，Gene Kim 用这个术语描述合理的产品开发和部署。这方面的最佳方法往往是参加有关 DevOps 实践和工具的聚会、会议、用户组（例如，DevOpsDays、DevOps 企业峰会、PuppetConf 和 Chef Conf）。阅读来自思想领袖、关于这一主题的数据 / 博客帖子 / 推文，并在行业活动和社会化媒体上与其交流。

1.4　小结

我们已经确定了 DevOps 的概念和对组织的好处，下面我们将更仔细地研究有助于团队成功的一些工具。

参考文献

[1] *The Phoenix Project*, by Gene Kim, Kevin Behr, and George Spafford

[2] “What Devops Means to Me,” by John Willis: https://www.getchef.com/blog/2010/07/16/what-devops-means-to-me/

HZ BOOKS
华章图书

Chapter 2

第 2 章

DevOps 工具

有一些工具能够帮助团队采用 DevOps 技术。本章介绍这些工具，在本书后面，将用实操示例更详细地介绍。

本章包含如下主题：

- 为成功而组织
- 服务器部署
- 配置管理
- 持续集成
- 日志分析

2.1 为成功而组织：看板

传统的运营团队任务管理通常涉及某种单据系统。故障单据很适合于跟踪问题，并在问题解决以后进行历史分析。故障单据系统的问题在于，它们给应该相关的任务之间的关联带来了困难。另一个挑战是生产交付瓶颈根源的识别。而且，工作者们可能无法看到其他人处理的故障单据，从而无法借助他人的专业知识。最后，管理工作过程、避免过度分配运营团队成员的最佳途径是什么？也就是说，如果运营团队总是专注于堆积如山的指派任务，他们何时才有时间改善系统，偿还技术债务呢？我们如何正确排定工作的优先级，考虑任务之间的依赖性？

看板（Kanban，字面翻译为“标记卡片”）系统有助于解决这些问题，以及其他的一些问题。这种方法是 Taiichi Ohno 在开发丰田制造系统时为了实现即时生产（JIT）目标而开发

的，它通过检查制造过程不同步骤的流程，识别需要补救的瓶颈，使系统更加高效。具体的思路是，缓解瓶颈，就会将工作任务从在途状态带到完成状态。限制在途工作可以为工作者带来空闲时间，对制造过程进行改进（例如，在缓解旧瓶颈的同时识别和消除新瓶颈）。图 2-1 展示了制造过程的估计完成时间没有实现的例子。

随着工作流从原材料输入到制造过程，最后进入装配过程，我们可以看到估算的完成时间和实际不符。装配过程似乎有某些问题。在进一步调查中，工厂工人们可能发现在金属薄板上喷涂特种涂料的过程效率不高。例如，服务器面板制造过程的输入可能表现出没有正确应用涂料的信号，所以进行返工，导致整个过程的效率不高。如果喷涂过程中固定薄钢板的托架没有超载，输出产品的制造可能从一开始就是正常的，不需要太多的返工。

在前一个例子中，如果工人埋头修复没有正确喷涂的金属薄板，而没有去识别问题的真正根源，就会继续浪费精力。我们从来没有在 IT 运营工作流中看到这种问题，对吗？

你可能会觉得奇怪，我们为什么在关于 DevOps 工具的章节中讨论制造工程组织方法。但是，在成功地改变工作方式之前，我们必须用一种条理性的方法来安排工作、识别系统中的问题。

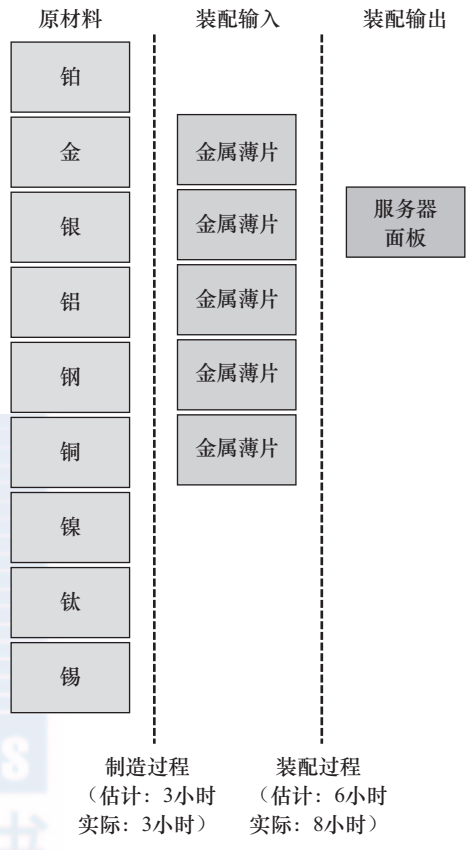

图 2-1　制造过程

尽管看板是制造中的一个学科，但是通过 David J. Anderson 和其他人的努力，将精益的概念与丰田的看板方法相结合，在 IT 业流行起来。我们不对看板做全面介绍，但是将讨论有助于 IT 运营团队的关键点。

看板系统最重要的特征是工作过程管理。在新工作请求时分配所有 IT 运营人员会造成效率低下，这一点似乎有些违反直觉。但是，运营团队面对和制造团队类似的问题。例如，如果团队使用预先制作的 ISO 金映像构建服务器，如果金映像有一段时间没有更新，那么在需要人工更新应用程序和打补丁时就有可能出现错误。如果团队的管理者将重点主要放在满足服务器请求，以至于 100% 的工作人员都参与这类工作流，而没有寻找改善过程效率的方法，这样的情况还会持续。不过，人为的错误可能引起失败，重要的安全更新可能遗漏，容易遭到攻击的端口保持打开，必须加以补救才能确保服务器不会遭到攻击。团队可能习惯于这种低效的状态，管理层也对产量感到满意。但是，很多时间浪费在这种重复劳动上，这种状态被称作“技术负债”。

技术负债是在计划好的工作期间，由于错误或者效率低下造成的所有计划外工作。运营团队需要充分的时间进行系统改进，即使这一工作没有相关的具体可交付成果。我并不是为运营团队的“游手好闲”辩护（经理们可能这么想）。运营团队看上去似乎很高效地服务于当前的客户需求，但是，当公司希望很快增加运营规模时会发生什么情况？现有的工作流不能停止，在你遇到没有准备好解决方案的伸缩性问题时，就要对系统进行改进。如果现在已经遇到某些这类问题，积极调查并缓解瓶颈，可以帮助你增进效率。

看板的另一个重要特征是工作流自始至终的可视化。最流行的展示方式是看板图，它可以采用物理或者数字形式。图 2-2 展示了可供第 1 章介绍的 DevWidgets 公司使用的看板图示例。

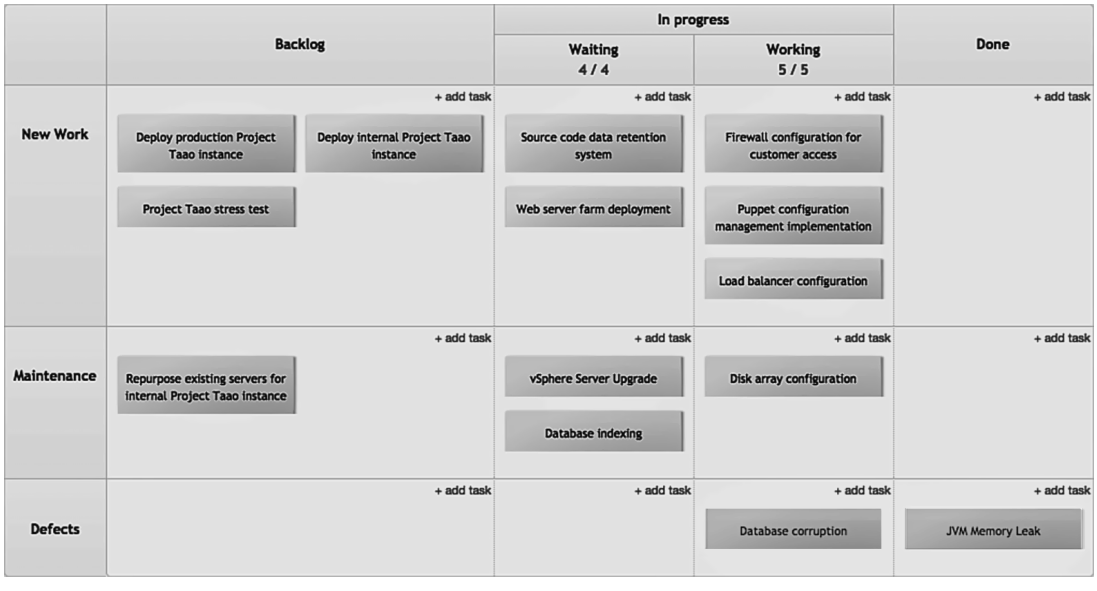

图 2-2 在线看板图

看板图的思路是每个任务由一张索引卡或者即时贴（也被称作看板）表示，在看板图左侧的“积压工作”（Backlog）分类下排队。“积压工作”和“完成”（Done）之间的栏目代表工作过程。在工作任务移出积压工作栏时，在它们上面放置工作过程（WIP）限制（例如，4/4 和 5/5）。在一张看板移到工作流的下一栏之前，不应该在 WIP 栏上放置其他工作。

图 2-2 展示了不同的任务行——也称为泳道。这些泳道对应于不同类型的工作（例如，新产品、维护和缺陷）。正确地为工作分类有助于优先级的排定。

团队第一次开始使用看板时，WIP 限制可以根据过去的经验设置。以后，随着瓶颈的缓解和团队效率的改进，这些限制可以更改，这样团队就可以随着时间的推移不断寻求工作流的改善。记住，目标是持续改善，同时避免团队 100% 投入新的任务。如果你对在团队中引入看板方法感兴趣，咨询该学科从业者中的佼佼者（如 Dominica DeGrandis 或精益看板大学

认可的从业者）是值得的。你的看板图不一定要和图 2-2 中的相似，目标是开发对团队有意义的清晰、实用系统。

2.2　服务器部署

系统管理员当然无法负担人工部署服务器场中的操作系统。20 世纪 90 年代出现的 Ghost 和其他映像备份工具使我们离快速部署更近了一步，但是使用这些解决方案需要某种初始化过程（例如，Windows 的 Sysprep），新系统才能使用。后来出现的 Red Hat Satellite 等 PXE 启动实用程序加速了这种部署过程。

当今的工作流程需要在 OS 部署完成之后进行更多的个性化，才能准备好实用的系统。当然，VMware 发布了 Auto Deploy，对 vSphere 服务器的部署有很大的帮助。Red Hat 用另一种眼光看待系统部署方法，开发了 CloudForms，该系统包括了多平台的服务器部署、配置管理（可与 Chef 和 Puppet 集成）、服务器生命期管理等。在开源战线上，Michael DeHaan 开发了 Cobbler，Nick Weaver 推出了 Razor（第 16 章中介绍）。其他流行工具包括 Crowbar 和 Foreman。

Razor 已经变得越来越流行，最新版本的 Puppet Enterprise 甚至在安装介质中捆绑了它。Razor 不只限于与 Puppet 协同工作，它的代理组件还支持 Chef，也可以为其编写用于其他配置管理技术的插件。

2.3　配置管理

正如第 1 章中讨论的，只依靠手工制作的金映像进行服务器部署可能导致效率低下。配置管理技术（CM）可以显著地改善金映像构建和生产系统部署的速度及可靠性。当你把服务器配置当成软件看待，就可以利用 Git（第 3 章中介绍）等源代码管理系统跟踪环境变化。

CM 技术还可以用于配给与生产服务器配置完全相符的一致开发环境。这可以消除开发和运营团队之间“但是它在我的开发工作站上工作得很好”的争论。CM 技术的设计很灵活，可以通过动态调整服务器特性（例如，OS 风格和位置），在不同平台上使用相同的指令集。本书介绍的 CM 技术——Puppet（第 4 ~ 6 章）、Chef（第 7 ~ 9 章）和 PowerShell DSC（第 12 ~ 13 章）——是描述性语言，你可以描述配给资源的预期状态，而不用担心工作是如何完成的。

配置管理的好处可以通过使用 Ansible（第 10 ~ 11 章）、Fabric 或者 MCollective 等编排系统大规模实现，这些系统用命令式的风格描述环境状态。编排框架允许配置管理在多个系统上以受控方式并行执行。

Ansible 也可以视为一种 CM 技术，因为它能够描述预期状态。有些公司可能选择仅使用 Ansible，而其他一些公司则使用 Puppet/Chef 的组合，由 Ansible 编排 CM 的执行。

2.4 持续集成

Jenkins（第 18 章中介绍）和类似的解决方案可能显著地节省开发和运营团队的时间。对于开发团队，如果编写了好的单元测试，它可以在代码移交给 QA 之前及早识别缺陷。对于运营团队，能够更加确保他们不会在预演阶段之前遇到未经验证的代码。

Jenkins 可以和 Git 及 Gerrit 集成，它们可以在开发团队将代码提交到软件存储库之后，立刻自动将代码提交给 Jenkins 供编辑和测试执行。而且，Jenkins 的作业历史和控制面板输出允许所有团队成员检查自动化测试和构建循环的各个阶段。

2.5 日志分析

系统的成功必须是可计量的。验证系统稳定性的最佳手段是什么？观察日志！你可能需要一支军队才能人工查看日志。而且，并不是整个团队都拥有日志通常需要的访问特权。幸好，整个运营团队可以得益于市场上的一些开源和商业化解决方案，这些方案可以用于日志收集和数据可视化。

VMware vRealize Log Insight 和 Splunk 是第一个想到的熟悉名称。还有 Logstash 等开源解决方案，它能够与 Elasticsearch 和 Kibana（第 17 章中介绍）、Graphite 以及 DevOps 社区中流行的其他工具相结合。因为这些开源工具都是 Linux 操作系统中包含的软件包，它们可以用你所喜欢的软件包管理器（如 yum 或者 apt）部署。

2.6 小结

现在，我们已经简单地概述了 DevOps 从业者可用的工具，第 3 章将聚焦于如何构建一个测试环境，以便跟踪本书其余部分介绍的工作流。

参考文献

[1] *Kanban*, by David Anderson
[2] *Toyota Kata*, by Mike Rother
[3] *The Phoenix Project*, by Gene Kim, Kevin Behr, and George Spafford

第 3 章 Chapter 3

建立 DevOps 配置管理测试环境

在不顾一切地深入 DevOps 工具的使用之前，我们先来研究如何建立测试环境，以便做好充分的准备。我们还要了解如何开始像对待开发人员的源代码那样处理基础设施指令。

本章包含如下主题：

- 用 AutoLab 进行环境配给
- 用 Vagrant 进行环境配给
- 用 Packer 创建映像
- 管理源代码
- Git 源代码控制

3.1 用 AutoLab 进行环境配给

AutoLab 是 Alastair Cooke 和 Nick Marshall 在 VMware 社区中其他人的帮助下开发的 vSphere 测试实验室配给系统。AutoLab 已经十分流行，至少有一个云提供商（Bare Metal Cloud）允许你在它的服务器上配给 AutoLab 安装。在本书写作时，AutoLab 支持最新版本的 vSphere，可以运行包含 VSAN 支持的虚拟实验室。如果不熟悉这个工具，可以在 http://www.labguides.com/ 首页上找到 AutoLab 链接。在填写注册表单后，就可以访问虚拟机（VM）模板，开始设置自己的 AutoLab。对于需要额外协助的人，该团队在 AutoLab YouTube 频道上有一组精选的帮助视频。

3.2 用 Vagrant 进行环境配给

Vagrant 是 Mitchell Hashimoto 创建，由他的公司 HashiCorp 支持的环境配给系统。它能够帮助你根据模板文件（称作 Vagrantfile）中定义的模式快速配置 VM。Vagrant 可以在 Windows、Linux 和 Mac OS X 操作系统上运行，支持流行的桌面虚拟化管理器（如 VMware Workstation Professional、VMware Fusion Professional 和 VirtualBox）。Rackspace 和 Amazon Web Services 等云提供商也可以用作你的测试环境。本书中的 Vagrant 示例基于 VMware Fusion 插件，但是我们提供的例子作少数修改就可以用于其他虚拟化管理器和云平台。如果按照本章中的例子学习，一定要为创建的每个 Vagrantfile 建立一个新目录。

注意 VMware Fusion 和 Workstation 提供商需要购买插件的许可证，更多细节参见 http://www.vagrantup.com/vmware。

安装 Vagrant 和 VMware Fusion 或 Workstation 插件后，你需要找到一个用于该系统的 Vagrant 盒子（box）。Vagrant 盒子可以看作一个 VM 模板：可以根据 Vagrantfile 中指定设置修改的预安装操作系统实例。Vagrant 盒子是所需操作系统的最小安装，有些盒子不超过 300MB。盒子的思路是提供一个完全由所选择的自动化工具配置的操作系统骨架。

有些盒子创建者选择包含 Vagrant 支持的流行配给系统（如 Puppet、Chef 等）的二进制代码，但是其他盒子创建者不这么做，用户必须使用命令行配给程序部署自己喜欢的配置管理解决方案才能使用。盒子的创建过程超出了本书的范围，但是，如果想要开发自己的盒子，可以尝试 veewee 和 Packer（本章后面将讨论）等工具。

在过去的 Vagrant 版本中，必须在初始化 vagrant 环境时指定所要使用的盒子文件的 URL：

```
vagrant init http://files.vagrantup.com/precise64_vmware.box
```

如果盒子文件在你的计算机上，可以用文件完整路径代替 URL。

从 Vagrant 版本 1.5 开始，HashiCorp 推出了 Atlas 系统（原称 Vagrant Cloud），这是一个在线存储库，如果使用保存在其站点上的账户和盒子名称，Vagrant 将在存储库中搜索盒子：

```
vagrant init hashicorp/precise64
```

对这两类语法都加以了解是很好的，因为对于没有寄存在 Atlas 上的任何盒子，都必须使用旧的方法。在线站点是搜索各种操作系统、代替自行构建的好地方。

vagrant init 命令自动创建一个简单的 Vagrantfile，引用你所指定的盒子。程序清单 3-1 展示了上述命令生成的默认 Vagrantfile。

程序清单 3-1 默认 Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
```

```
# Vagrantfile API/syntax version. Don't touch unless you know what you're
doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "hashicorp/precise64"
end
```

注意，为了节约空间，我删除了初始化 Vagrantfile 时自动生成的注释。不过，如果查看注释，会看到使用配置管理技术在 VM 启动时自动化修改的有用提示。目前，配给 VM 的可用选项包括基本 shell 脚本和配置管理工具（如 Puppet、Chef 和 Ansible）。这样很有价值，因为开发和测试环境可以使用和生产部署完全相同的设置，从而消除出现部署问题时没完没了的“它在我的笔记本电脑上运行得很好”的讨论。Vegrant 还添加了 Docker 支持，配给系统可以自动安装 Docker 守护进程，下载指定使用的容器。

有了 Vagrantfile，现在可以使用如下命令启动第一个测试环境：

```
vagrant up --provider=vmware_fusion
```

注意　如果你在 Windows 或者 Linux 平台上使用 VMware Workstation，可以使用不同的提供者：vmware_workstation。

现在，可以用如下命令登录 VM：

```
vagrant ssh
```

查看 VM 中的 /vagrant 文件夹，将会看到它包含了你的 Vagrantfile。这是一个为 VM 自动创建的共享文件夹，可以轻松地在桌面上传入和传出文件，而不需要使用 SCP、FTP 等。

如果检查操作系统资源，会注意到有一个 vCPU 和 512MB 的 RAM。这对你想要运行的应用程序可能不足，所以，我们来看看如何修改分配给 Vagrant VM 的资源。

首先，我们删除这个 VM，以便转向其他的配置选项，退出 VM 并使用如下命令：

```
vagrant destroy
```

Vagrant 将要求确认是否真的要删除该 VM，也可以使用 -f 选项跳过确认。

程序清单 3-2 说明，Vagrant 可以修改 VM 的 VMX 文件，进行我们需要的更改。我们使用 config.vm.provider 代码块实现。顺便提一句，memsize 属性的单位是 MB。注意，我们将创建一个名为 v（包含在两条竖线之间）的对象，专为这个 VM 更改设置。这个对象名只有 config.vm.provider 语句内的局部作用域，可以在定义其他 VM 时再次使用，在后面的例子中可以看到。执行 vagrant up 之后，将创建具备所需属性的 VM。在本书写作时，虚拟磁盘的大小和数量还无法控制，但是你的 Vagrant VM 将启动 40GB 的精简配置存储。

程序清单 3-2 更改默认的 Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# Vagrantfile API/syntax version. Don't touch unless you know what you're
doing!
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "hashicorp/precise64"
  config.vm.provider :vmware_fusion do |v|
    v.vmx["memsize"] = 1024
    v.vmx["numvcpus"] = 2
  end
end
```

可以修改 VM 的资源是很好的。那么，更复杂的设置（如多个 VM）该怎么进行？Vagrant 也支持这种拓扑。当然，一定要有足够的 CPU 核心和 RAM 以支持想要使用的拓扑！例如，多 VM 设置对测试具有独立数据库服务器和前端服务器的实际部署很有用。程序清单 3-3 展示了多 VM Vagrantfile 设置的一个例子。

程序清单 3-3 多机器 Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# Vagrantfile API/syntax version. Don't touch unless you know what you're
doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.define :first do |vm1|
    vm1.vm.box = "hashicorp/precise64"
    vm1.vm.hostname = "devops"
    vm1.vm.provider :vmware_fusion do |v|
      v.vmx["memsize"] = 1024
      v.vmx["numvcpus"] = 2
    end
  end

  config.vm.define :second do |vm2|
    vm2.vm.box = "hashicorp/precise64"
    vm2.vm.hostname = "vmware"
    vm2.vm.provider :vmware_fusion do |v|
      v.vmx["memsize"] = 1024
      v.vmx["numvcpus"] = 2
```

```
    end
  end
end
```

这一部署利用了多个 config.vm.define 代码块：对我们要创建的每个 VM 各使用一个。:first 和 :second 是 Vagrant 在运行 vagrant status 等命令时用于标识 2 个 VM 的标签。这些标签也用于通过安全外壳（SSH）连接到 VM 时——例如，vagrant ssh first。如果你熟悉 Ruby，就会注意到这些标签是 Ruby 符号。包围在管道符号中的名称（例如，|vm1|）表示提供 vagrant 用于构建和定义 VM 信息的对象。对象名可以和符号相同（例如，first.vm.box…），但是不一定是这样。

当你想要部署超过 2 个 VM 时，使用这种语法可能有些乏味。幸好，因为 Vagrant 是用 Ruby 编写的，可以使用该语言的特性（如列表、循环和变量）优化 Vagrantfile 代码。程序清单 3-4 展示了我从 Cody Bunch 和 Kevin Jackson 的《 OpenStack Cloud Computing 》中学到的一些优化技巧。

程序清单 3-4　优化的多机器 Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
servers = ['first','second']

Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise64"
  servers.each do |hostname|
    config.vm.define "#{hostname}" do |box|
      box.vm.hostname = "#{hostname}.devops.vmware.com"
      box.vm.provider :vmware_fusion do |v|
        v.vmx["memsize"] = 1024
        v.vmx["numvcpus"] = 2
      end
    end
  end
end
```

在文件的开头，我创建了一个名为 servers 的 Ruby 列表，它的元素是想要创建的 VM 名称。然后，使用 Ruby 列表迭代子 each 循环执行服务器列表中每个元素的 VM 定义。如果想要增加部署的 VM 数量，只要在列表中添加更多的条目即可。不是每个 VM 都必须使用相同的资源集，可以在 box.vm.provider 代码块中使用 if 语句选择：

```
      if hostname == "first"
        v.vmx["memsize"] = 3128
        v.vmx["numvcpus"] = 4
```

```
elsif hostname == "second"
  v.vmx["memsize"] = 1024
end
```

Vagrant 还有许多特性无法在本书中介绍，但是用这几个简单的命令就能构建我们在本书中使用的测试环境。如果想要更多地了解 Vagrant，一定要访问 Vagrant 的网站（http://www.vagrantup.com），阅读 Mitchell 的《Vagrant: Up and Running》一书。

3.3 用 Packer 创建映像

Packer 是 HashiCorp 的另一种帮助你为多平台开发自定义盒子的产品。假定你想要从同一个基本盒子出发，开发用于 Workstation/Fusion 和 ESXi 的 VM 映像，Packer 就可以实现。

Packer 使用 JavaScript 对象标记法（JSON）文件格式指定 Vagrant 盒子的配置（磁盘大小、内存等），一旦指定了相关的自动化参数（例如，Ubuntu preseed 文件），它将帮助你进行初始 OS 部署。

Packer 不仅对创建 Vagrant 盒子有用；它的主要用途是制作与流行云提供商格式（OpenStack、AWS 等）兼容的映像文件。但是，Packer 包含构造器功能，可以自动输出与 VMware Fusion/Workstation 和 VirtualBox 兼容的 Vagrant 盒子，可以使用 Puppet 和 Chef 等流行配置管理技术，自定义生成的映像。

我们不深入讨论 Packer，但是如果想要自己试验自定义 Vagrant 盒子的构建，我们希望你了解它。可以在 http://www.packer.io 上找到更多关于 Packer 的信息。如果想要查看可用于开发你自己的 VM 的 Packer 定义文件，Chef 团队在自己的 Github 账户上维护着一个名为 bento 的存储库：https://github.com/chef/bento。

3.4 管理源代码

源代码管理（SCM,Source Code Management）是 DevOps 环境中必不可少的元素。想象一下：如果你要把基础设施转换为代码，重要的是有一种回顾任何更改、在新更改引入问题（例如，在最好的情况下是定期出现不稳定的情况，在最糟糕的情况下引起停机）时回到文件不同版本的手段。有些人可能认为“容易”的方法是建立文件的多个拷贝，每个都使用唯一的名称（Vagrantfile1、Vagrantfile2、Vagrantfile01012015 等），但是接着当你想要使用这些文件并且试图记住所有文件的不同之处时，就会陷入文件重命名的麻烦之中。

开发组织中的不同团队很有可能已经使用某种 SCM 系统管理他们的工作（例如，软件开发人员保存其源代码，QA 团队管理测试脚本）。在你开始使用 SCM 技术时，和其他小组讨论最佳实践是值得的。

SCM 解决方案包括 SVN、Mercurial 等。Git 是 DevOps 社区中较为流行的 SCM 系统之一，所以，在本书中我们使用 Git。

使用 Git

Git 是一个分布式版本控制系统，这意味着建立中央存储库的一个本地拷贝，而不只是访问单独的文件。本地提交可以同步到中央服务器，保持环境中的一致性，用户总是可以从中央存储库获取最新的源代码版本。这种架构与传统的源代码管理系统不同，原来的系统中只有中央服务器有完整的存储库拷贝。

在实验本书中的例子时，建议使用免费的在线 Git 存储库，如 BitBucket、GitHub 和 Gitorious，作为存储代码的中心位置。每个站点都有自己的独特功能。例如，BitBucket 允许无限制的免费私有存储库。GitHub 是本书作者用于自有代码的在线存储库系统。但是，完全可以使用任何符合你的需求的系统，因为获取代码（克隆 / 拉取）和存储代码（推送）的方法在任何 Git 系统上都通用。

注意　对于生产环境中的项目，考虑使用公共存储库站点之前请咨询法律部门。虽然许多站点都提供私有存储库功能，但是公司的领导可能更愿意使用内部中央 Git 服务器。

创建第一个 Git 存储库

首先，用你喜欢的包管理器（例如，Mac OS X 上的 homebrew 或 macports，Ubuntu/Debian 上的 apt-get，Red Hat/CentOS/Fedora 上的 yum）安装 Git。对于 Windows 用户，GitHub 和 BitBucket 等流行在线存储库提供软件客户端，简化了与在线存储库系统的交互。另外，http://git-scm.com 网站维护用于 Windows 的一个 Git 二进制独立安装版本。

如果你使用 Linux 或者 Mac OS X，可以打开一个终端窗口试验下面的例子。Windows 用户必须使用随 Git 客户端安装的特殊 shell。示例的语法基于 Liunx/Mac，但是在 Windows 平台上的命令应该等价。

在开始编写代码之前，必须设置两个全局变量，让 Git 知道我们是谁。这不像存储库的本地拷贝那么重要，但是在向远程服务器推送代码时非常重要。这两个全局变量是你的电子邮件地址和用户名：

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

事实上，如果尝试使用 Git，在第一次提交时没有设置这些变量，Git 将提示你设置它们之后才能继续。

如果你试验过前面的 Vagrant 示例，就已经有了一个用于处理内容的目录。否则，创建一个新目录并在其中创建一个文本文件。从现在起，要在命令行提示符上确认处于该目录。

首先，初始化该目录以建立一个 Git 存储库：

```
git init
```

如果用显示隐藏文件的选项列出目录（Linux/Mac 上的 ls-a，或者 Windows 上的 dir/

A:H)，就会看到一个隐藏目录 .git。这个目录包含存储库的文件和特定设置。这些本地设置和我们前面设置的全局设置组合，使用如下命令可以确认：

```
git config -l
```

如果想要查看目录中文件的状态（文件是否已经加入存储库？最后一个命令之后有无更改？等等），可以输入 git status，会看到类似程序清单 3-5 中展示的输出。

程序清单 3-5 Git 存储库状态

```
git-test $ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .vagrant/
        Vagrantfile

nothing added to commit but untracked files present (use "git add" to track)
```

最后一行最重要，它告诉我们需要跟踪存储库的文件以进行管理。注意 SCM 工具不会自动跟踪放入目录中的文件是很重要的。这一特性可以避免我们跟踪存储库中的垃圾文件，浪费空间。

我们告诉 Git 跟踪 Vagrantfile：

```
git add Vagrantfile
```

但是 .vagrant/ 目录没有必要跟踪，因为它只包含 Vagrant 用于设置 VM 的临时文件。可以创建一个 .gitignore 文件，明确地告诉 Git 忽略这个目录。使用你所喜欢的文本编辑器，创建有一个条目的 .gitigonore 文件：

```
.vagrant/
```

也可以使用简单的 echo 命令实现相同的功能。（Windows 用户需要使用包含在 Git 安装中特殊的 Git Shell 二进制文件才能正常工作。）

```
echo '.vagrant/' > .gitignore
```

如果再次运行 git status 命令，将会看到 Git 显示关于 .gitignore 文件的信息。发生了什么情况？记住，必须告诉 Git 如何处理存储库所能看到的任何文件或者目录，包括 .gitignore 文件。有两种方法可以处理 .gitignore 文件：

- 将 .gitignore 文件本身加入需要忽略的文件及目录列表。

■ 告诉 Git 跟踪 .gitignore 文件。

我将使用第二种选项，以便让使用存储库的其他人都能忽略相应的文件：

```
git add .gitignore
```

现在，如果再次检查存储库状态，会看到类似程序清单 3-6 中展示的输出。

程序清单 3-6　更新后的存储库状态

```
git-test $ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   .gitignore
        new file:   Vagrantfile
```

目录中的所有文件都已经准备好提交或者添加到非必要文件及目录的 .gitignore 列表。剩下的工作就是提交存储库更改：

```
git commit
```

这将自动打开一个文件编辑器，可以输入关于提交文件的细节。（默认使用 vi。）参见程序清单 3-7。

程序清单 3-7　你的提交消息

```
git-test $ git commit
  1 This is my first commit.
  2 # Please enter the commit message for your changes. Lines starting
  3 # with '#' will be ignored, and an empty message aborts the commit.
  4 # On branch master
  5 #
  6 # Initial commit
  7 #
  8 # Changes to be committed:
  9 #       new file:   .gitignore
 10 #       new file:   Vagrantfile
 11 #
```

必须输入一条消息；否则，提交将被撤销。如果不熟悉 vi，按下 I，输入一些文本，按下 Esc 键，然后输入 :wq 并按 Enter 键。如果不想使用文本编辑器，可以使用 git commit 命令的简短形式，加上 -m 选项，在同一行输入提交消息：

```
git commit -m "This is my first commit."
```

如果提交成功，应该看到如下输出：

```
[master (root-commit) d962cd6] This is my first commit.
 2 files changed, 119 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Vagrantfile
```

使用中央 Git 服务器（远程服务器）

如果在 GitHub、BitBucker 或者任何公共 Git 存储库站点上开立了账户，就必须向站点提供计算机的 SSH 公钥，以便验证你的身份。每个站点完成这一操作的方法不同，查询文档找到对应的步骤。对于 Windows 用户，通常可以安装站点的客户端软件自动处理。Mac 和 Linux 用户必须使用 ssh -keygen 命令生成 SSH 公钥。

在远程服务器上正确配置 SSH 公钥之后，就可以在远程站点创建存储文件的存储库：这一过程称作推送（pushing）。在网站上创建远程存储库时，应该跳过 README 文件的自动生成。在存储库创建之后，站点将提供一个链接，可以使用它告诉本地存储库远程服务器的位置，该链接的标签往往标记为 origin。

在我的设置中，为存储库取了和本地存储库相同的名称。这是一个可选项，名称可以不同。可以使用 git remote 命令和 GitHub 提供的链接更新本地存储库设置：

```
git remote add origin git@github.com:DevOpsForVMwareAdministrators/git
```

使用 git config -l 命令，可以看到关于远程服务器位置的新数据：

```
remote.origin.url=git@github.com:DevOpsForVMwareAdministrators/git-test.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
```

现在，可以从本地存储库向远程存储库推送文件：

```
git push origin master
```

会看到类似程序清单 3-8 展示的输出。

程序清单 3-8　Git 远程推送结果

```
git-test $ git push origin master
Warning: Permanently added the RSA host key for IP address '196.30.252.129'
to the list of known hosts.
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 2.13 KiB | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To git@github.com:DevOpsForVMwareAdministrators/git-test.git
 * [new branch]      master -> master
```

图 3-1 展示了向远程服务器推送第一次提交的文件后 GitHub 上的存储库。

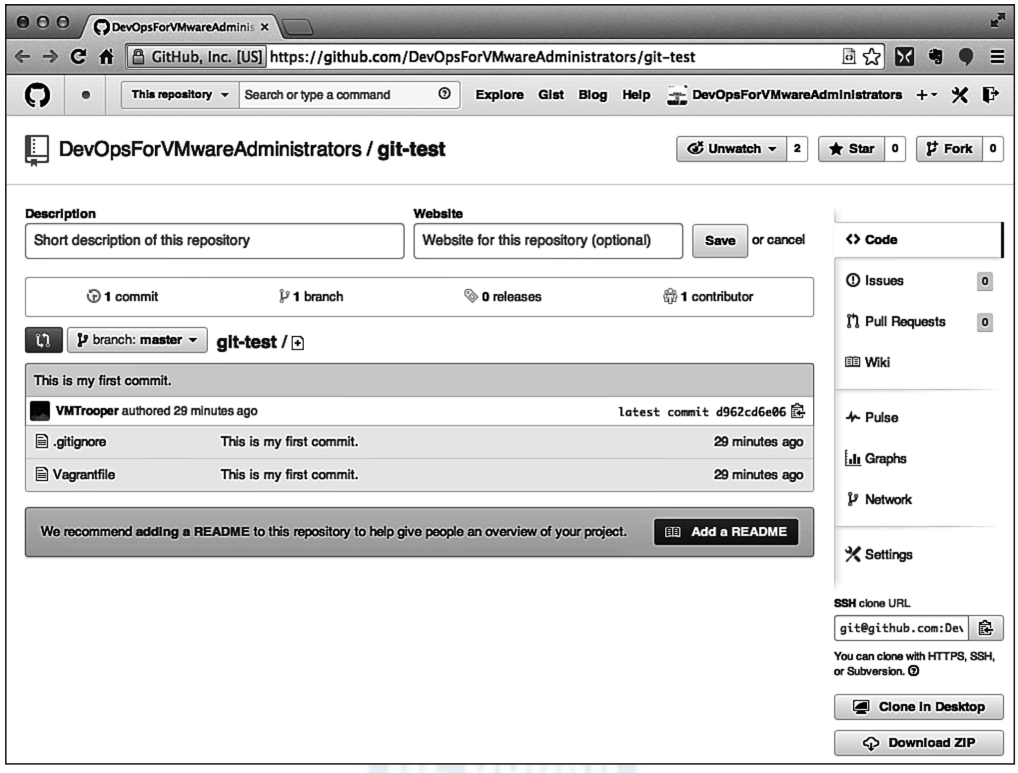

图 3-1　远程 Git 存储库

如果有了想要使用的远程存储库（正如我们为本书提供的样本），可以使用基于 CLI 或者 GUI 的方法将存储库克隆到本地机器。远程存储库系统应该有类似于图 3-1 左下角的选项。

根据使用的站点，基于 GUI 的方法有所不同。然而，在 GitHub 上，如果使用 Mac 或者 Windows 客户端，可以使用 Clone in Desktop 按钮，或者使用 Download Zip 按钮——这是一个包含存储库所有源代码的简单 zip 文件。如果使用 Windows 或 Mac 平台，建议使用 Clone in Desktop 选项，因为它将自动创建指向远程存储库的 Git 链接。

基于 CLI 的方法包括取得 SSH 或者 HTTP 克隆 URL 和使用 git clone 命令，如：

```
git clone https://github.com/DevOpsForVMwareAdministrators/git-test.git
```

执行上述命令之后，Git 将以存储库名称（在本例中是 git-test）创建一个目录，并将存储库的内容拷贝到这个目录中。你可以开始使用和更新代码，Git 远程链接将自动创建，正如前面提到的 Clone in Desktop 按钮那样。如果你有更改远程存储库的权限，可以执行 Git 推送，将任何本地更改转发到远程存储库。请求更改其他人的存储库不在本书讨论的范围内，但是，如果对此感兴趣，可以研究存储库的分支和拉取请求。这些功能的实现在不同的公共

存储库站点上各不相同，所以，必须查看所使用站点的对应文档。

如果在设置远程存储库时有问题，替代的工作流如下。

1. 开始工作之前，在所选择的远程 Git 站点（例如，GitHub）上创建一个新的空存储库。

2. 用前面讨论的 GUI 或者 CLI 方法将空白的存储库克隆到本地机器。

3. 开始在克隆的目录中工作，并执行提交。之后，可以使用前面介绍的 git push 命令将源代码转发到远程存储库。

3.5 小结

我们讨论了本书中用于构建测试环境的基础工具——Vagrant 和 Git。Vagrant 可用于快速构建测试环境，Git 将帮助你跟踪对环境定义的变化，以便在必要时撤销更改。在第 4 章中，我们开始用实操示例讨论 Puppet 配置管理技术。

参考文献

[1] http://docs.vagrantup.com/v2/

[2] https://help.github.com/

HZ BOOKS
华章图书