IBM Press

DevOps实施手册

在多级IT企业中使用DevOps

The DevOps Adoption Playbook

A Guide to Adopting DevOps in a Multi-Speed IT Enterprise

[美] Sanjeev Sharma 著

万 金 译



2017年度最佳DevOps图书

本书刚刚荣获“2017年度最佳DevOps图书”的殊荣。我强烈推荐本书，书中的内容通俗易懂，适合于所有人，无论是完全没有DevOps经验的新手还是DevOps的专家。

书中列举了许多真实的案例，来启发人们思考和了解DevOps的价值，以及如何在各自的DevOps实施过程中采取正确的方法。

——Ahmed Azraq

清华大学出版社



DevOps 实施手册

在多级 IT 企业中使用 DevOps

[美] Sanjeev Sharma 著

万 金 译

清华大学出版社

北 京

Sanjeev Sharma
The DevOps Adoption Playbook：A Guide to Adopting DevOps in a Multi-Speed IT Enterprise
EISBN：978-1-119-30874-4
Copyright © 2017 by John Wiley & Sons, Inc., Indianapolis, Indiana
All Rights Reserved. This translation published under license.
Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affi liates, in the United States and other countries, and may not be used without written permission. IBM, the IBM Press logo, UrbanCode, uDeploy, System z, Rational, IBM Watson, WebSphere, Bluemix, InfoSphere, Optim, PureApplication, DB2, SoftLayer, and Blue Box are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries. A current list of IBM trademarks is available on the web at "copyright and trademark information" as www.ibm.com/legal/copytrade.shtml. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.
本书中文简体字版由 Wiley Publishing, Inc. 授权清华大学出版社出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号 图字：01-2017-4033

Copies of this book sold without a Wiley sticker on the cover are unauthorized and illegal.

本书封面贴有 Wiley 公司防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

DevOps 实施手册　在多级 IT 企业中使用 DevOps / (美) 桑吉夫·夏尔马(Sanjeev Sharma) 著；万金　译. —北京：清华大学出版社，2018
书名原文：The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise
ISBN 978-7-302-49826-1

Ⅰ. ①D…　Ⅱ. ①桑…　②万…　Ⅲ. ①软件工程　Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字(2018)第 037603 号

责任编辑：王　军　于　平
封面设计：牛艳敏
版式设计：思创景点
责任校对：孔祥峰
责任印制：沈　露

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市金元印装有限公司
经　　销：全国新华书店
开　　本：148mm×210mm　　印　　张：11.625　字　　数：311 千字
版　　次：2018 年 4 月第 1 版　　印　　次：2018 年 4 月第 1 次印刷
印　　数：1～4000
定　　价：79.80 元

产品编号：075958-01

致我的夫人 Ritika，她总是激励我做得多一点、再多一点，从不满足于现状。还有我的孩子们，Saransh 和 Shreya，他们是我不懈努力、持续前行的动力。

译者序

一次成功不是终点，一次失败也并非末日，变幻的是人生的风景，不变的是勇往直前的心。

DevOps 的理念诞生于 2009 年，在同年的敏捷大会上(Velocity 2009-O'Reilly Conferences)，John Allspaw 和 Paul Hammond 发表了震惊业界的演讲，内容是关于如何在 Flickr 实现每天发布 10 次更新，而当时的平均发布周期都是以月或年为计算单位的(发布效率提升了百倍以上)。可想而知，DevOps 的横空出世对于生产环境中的发布效率产生了怎样的革命性影响，用“行业地震”来描述也一点都不为过。

DevOps 的主旨是在开发与运维之间建立信任与责任共担机制，既要满足开发的快速创新需求，又要保证运维的稳定性——这就需要建立反脆弱系统。与传统 IT 建立高可靠性系统(Mean Time To Failure，故障前平均时间)的目标不同，DevOps 强调的是反脆弱性(Mean Time To Recovery，平均修复时间)。顾名思义，反脆弱系统不但不会因为压力而崩溃，反而会随着压力增加而变得更加强大。通过寻找可替代系统的方式实现反脆弱性，可以同时满足业务快速变化与运维稳定性的需求。

蓦然回首，DevOps 已经走过 8 年的发展历程。随着 DevOps 理念的推广，各种工具不断涌现，但是对于传统大型企业而言，每天发布新特性，还是难以企及的目标。只有在初创企业或互联网企业中，才更有可能实现从开发到运维的快速交付。DevOps 无法在多数的大型传统行业实施吗？答案当然是否定的。

我曾经在 IBM 和华为从事 DevOps 实施方面的工作，常常需要解决的问题就是使用什么样的工具才能既满足流程规范的要求，又能提升团队的效率并且保证交付的质量。而几年之后，当我作为 ThoughtWorks 的顾问思考如何缩短周期时间、缩小批次规模以及建立文化理念时，我

才深刻地认识到工具与工具(根据康威定律，软件架构体现了组织的架构)之间竟然存在着如此巨大的效率鸿沟。就如本书的第 1 章开篇案例所讲述的故事，一个只需要运行 20 分钟的回归测试任务，在实际的执行过程中，由于部门间协作低效、环境差异、安全审计、测试数据准备等原因，实际过程长达一周之久。一个惊人但是常常为人所忽略的事实是，在现有框架基础上对当前工作效率提升 20%，这一过程中的投入竟然远远高于建立全新体系，寻求效率提升 10 倍的方法。你没有看错，效率提升 10 倍的方法反而成本更低。一个著名的例子就是，Space X 致力于打造出比目前火箭发射成本至少低 10 倍的产品，而其所需的资源一定会少于美国国家航空航天局(NASA)在原有技术上的“小修小补”。是时候开始 DevOps 变革了。

本书作者桑吉夫·夏尔马(Sanjeev Sharma)是国际知名的 DevOps 与云计算领域的思想领袖，基于丰富的行业经验，他在本书中通过海量案例和大量访谈，为正在实施 DevOps 的团队提供建议与指导，其中还包括相应的反模式案例。作为 DevOps 领域中有着多年经验的从业者，我在翻译这本书的过程中一直遏制不住相见恨晚的感叹以及一拍即合的兴奋。

我与本书的缘分始于第一届中国 DevOpsDay 峰会，会议期间，我偶然地获得了翻译本书的机会。让我兴奋的是，本书介绍的 IBM 云计算与 ThoughtWorks 敏捷方法论同我自己的工作与研究经历高度吻合，在翻阅的过程中，我竟然产生了一种既真实又虚幻的使命感：这就是为我而写的书。在接受翻译任务后，我需要在工作之余每天投入几小时的整块时间，每当翻译到令人叹为观止的理念或者困扰我良久的问题时，我就会完全沉浸在狂喜之中，以至忘记了时间；而每当结束一个主题章节的翻译时，我都深深地被两家伟大的公司在方法论以及软件工程领域进行的探索所震撼，并为能够成为其中一员而感到无比自豪。无数的清晨和安静的长夜，我沉浸在这些伟大的思想世界中，忘记了自我，也少了许多陪伴家人的时间，在此非常感谢家人对我的支持。

在知识快速更新，社会变得十分浮躁的时代，写一本书甚至只是翻译一本书或许都是非常奢侈的事情。2017 年，不断涌现出的容器、微服

务渐成热点，云原生和 Kubernetes 也出现在我们的视野中，难能可贵的是，这些新理念和工具在本书的末尾都有所涉猎，作者桑吉夫也在其个人网站上不断更新本书的英文内容。为了方便中国读者进行反馈和交流，我会建立一个微信群(通过添加我的微信号 aaron-i，我会把读者添加到微信群中)，以方便热爱求知的朋友们在快节奏的生活中与同样爱好本书的人进行互动，微信群中已经添加了我认识的很多这个领域的大咖级人物。

最后，致敬本书的作者桑吉夫，再次感谢我的家人，感谢清华大学出版社以及所有为本书出版辛苦付出的人们。希望这本书对你们有所裨益，衷心感谢大家！

万　金

作者简介

桑吉夫·夏尔马是国际知名的 DevOps 与云计算领域的变革思想领袖、技术高管以及作家。桑吉夫具有丰富的行业经验，曾担任首席技术官(CTO)、全球技术销售负责人、采购集成技术负责人以及 IT 架构师。作为 IBM 的杰出工程师，桑吉夫被公认为 IBM 最高级别的核心技术领袖。

桑吉夫主导并推动 DevOps 与云计算前沿解决方案、架构以及策略的实施。IBM DevOps 技术销售部全球首席技术官的经验，加上对业务及 IT 需求的深刻洞察与理解力，使其对任何业务都能产生独特的见解，从而能够从独特的视角为高层管理者及高级技术管理人员提供建议与指导，以实现跨行业、跨地域的 DevOps 及云计算变革。

作为云计算及 DevOps 专家，桑吉夫经常在国际科技论坛上发表演讲，还经常在领先的科技刊物以及自己的博客(http://bit.ly/sdarchitect)与推特(@sd_architect)上发表文章、博文以及视频。

技术审稿人简介

李·里德在制造与信息技术领域拥有 30 多年的软件工程、架构、产品研发、技术创新以及团队管理经验。李·里德是通用汽车学院(General Motors Institute，BME)及密歇根大学(University of Michigan，MSE)毕业的工程硕士研究生，持有四项美国专利。最近他转行到高等教育领域，正带领 IT 部门将精益与 DevOps 实践引入圣诺伯特学院(St. Norbert College)。

致　谢

本书旨在记录下我与客户、同事以及 DevOps 同行关于 DevOps 以及 IT 优化与创新问题所进行的无数对话、讨论(有时是激烈的)以及争论的内容。通过这些对话与讨论，很多人为这本书做出了贡献，我学习用到的博客、文章、书籍、网络研讨会、视频、会议以及演讲就更不用说了。

我的同事，IBM 的 DevOps 专家及技术思想领袖们是主要的贡献者，他们是(按姓氏的字母顺序)：

- Al Wagner
- Albert Ho
- Alex Abi Khaled
- Ana Lopez-Mancisidor
- Andy Moynahan
- Ann Marie Somerville
- Anshu Kak
- Anujay Bidla
- Ava Hakim
- Bala Rajaraman
- Bernie Coyne
- Bill Higgins
- Bob Bogan
- Brian Naylor
- Chris Lazzaro
- Chris Lucca
- C. J. Paul
- Claudette Hickey
- David Ziskind
- Dibbe Edwards
- Eric Minick
- Erik Anderson
- Greg Wunderle
- Hayden Lindsey
- Helen Dai
- Jagan Karuturi
- James Pierce
- Jeff Crume
- Jim Fieseler
- Jim Moffi tt
- John Lanuti
- John Wiegand
- Kay Johnson
- Kedar Walimbe
- Kristof Kloeckner
- Kyle Brown

- Cliff Utstein
- Dan Berg
- David Curbishley
- David Leigh
- Mark Meinschein
- Mark Roberts
- Mark Tomlinson
- Meenagi Venkat
- Michael Elder
- Michael Samano
- Mike McNamee
- Mustafa Kapadia
- Paul Bahrs
- Paul Meharg
- Peter Eeles
- Peter Spung
- Randy Newell
- René Bostic
- Rick Weaver
- Rob Cuddy
- Leigh Williamson
- Mahendra Pingale
- Maneesh Goyal
- Mark Borowski
- Robbie Minshall
- Roger Snook
- Rosalind Radcliff
- Sal Vella
- Saleem Padani
- Steve Abrams
- Steve Kagan
- Steven Boone
- Sudhakar Frederi
- Swati Moran
- Tony Doyle
- Tim Hahn
- Tim Pouyer
- Varban Vassilev
- Wendy Toh

有些贡献者是以前在 IBM 工作过的，他们是：

- Alan Sanie
- Ashok Reddy
- Bowman Hall
- David Grimm
- David Myers
- Jan Svoboda
- Mike Lundblad
- Murray Cantor
- Steven Pogue
- Walker Royce

作为在自己公司及组织中引领 DevOps 变革的真正领导者，一些关键客户、业务合作伙伴与专家也为本书做出了贡献。他们的实战案例是汲取经验的最好来源。很多情况下，本书中所记录的经验教训与

实践就来自他们所参与的对话。作为 IBM 的员工，我会见了这些人中的大多数，这里无法一一列举。仅在此列出与我共同出席会议、聚会以及网络研讨会，并联合撰写文章或博客的几位，他们以及他们目前的雇主是：

- Alan Shimel, DevOps.com
- Antony Morris, Monitise
- Ben Chodroff, CloudOne
- Brad Schick, Skytap
- Carmen DeArdo, Nationwide Insurance
- Chris Lepre, Wells Fargo
- Gareth Evans, Monitise
- James Governor, RedMonk
- Jayne Groll, DevOps Institute
- John Comas, NBCUniversal Media
- John Kosco, Blue Agility
- J. P. Morgenthal, CSC
- Mark Howell, Lloyds Banking Group
- Tapabrata “Topo” Pal, Capital One

我必须要向 DevOps 大师基恩·金(Gene Kim)表达我的特别谢意，通过其著作以及 DevOps 企业高峰会议，他为本书做出了巨大贡献。我曾有机会亲自与他进行一对一的交谈，其中包括 2014 年录制的视频访谈。

我还要向李·里德表示特别感谢。我和他共事十多年的时间，他也是我的“共谋犯”，曾经在 IBM 领导全球 DevOps 架构团队两年。我们共同开发了 DevOps 价值流图工作坊技术，我的很多想法也都是来源于他。尽管他已经离开 IBM 到圣诺伯特学院工作，我还是请他担任本书的技术编辑，因为只有这样我才有机会分享他的思想才智。没有 Lee 的洞察、批评与反馈，这本书就无法呈现最终精炼、清晰的结构形式。

最后，我要感谢 Wiley 出版社的杰出编辑人员，Adaobi Obi Tulton，

他的技能绝对不会辜负“绝地武士”的这个称号，还有 Marylouise Wiack，感谢她对语言及表达方式的全面把握(是的，标点符号是我的克星)。因为他们的辛勤工作以及细致的校正，将我的只言片语组合成通顺的句子，这本书得以比计划提前许久面世。

前　言

DevOps 实施手册中涵盖的内容

2016 年 4 月，在 2016 届 NCAA 全国大学生篮球锦标赛上，维拉诺瓦大学野猫队对战北卡罗来纳大学焦油踵队。这是有史以来最伟大的一场赛事，一切都要归功于距离比赛结束只剩 4.7 秒时的最后一次进球。Joel Berry II 以一记三分球将比分追成 74 平，维拉诺瓦大学队的教练 Jay Wright 叫了最后一次暂停。在 NCAA，暂停后必须要走完全场。随即，维拉诺瓦大学队的 Kris Jenkins 将球传给控球后卫 Ryan Arcidiacono。Arcidiacono 越过 Berry 沿着球场运球，双方为了最终的胜利都设计了相应的战术。北卡罗来纳对 Arcidiacono 采取 1-3-1 人盯人的压制战术，希望迫使其失误。但是，即便 Arcidiacono 成功越过 Berry，他们还有 Justin Jackson、Isaiah Hicks 以及 Brice Johnson，所有这些人都能阻止三分球命中。维拉诺瓦也已经设计了相应的战术，确保 Arcidiacono 能够把球带到场内，可以越过半场并找到三分线上球员的位置。Arcidiacono 瓦解了北卡罗纳战术，越过 Berry 并迅速将球回传给三分线上的 Jenkins，最终以一记毫无争议的三分球赢得冠军，成功绝杀。

—— Saransh Sharma(Sharma, 2016)

一本规模化实施 DevOps 的指导手册

优秀的团队之所以优秀，不仅仅是因为他们拥有最好的成员、最好的工具、最好的培训、最好的流程或者最好的领导及教练。还因为作为一个团队，他们拥有上述一切的同时，也知道面对各种情况与挑战时应该做些什么。他们都有一本指导手册，里面包含各种情形下可能会用到的解决方案。

当面对独特的形势或挑战时，作为一个团队，球员和教练共同从指导手册中挑选适当的方案并执行，这点尤为重要。我的母校，维拉诺瓦大学在比赛结束前仅仅几秒钟利用最终的战术赢得了全美冠军，因为这些战术他们是练习过的。他们分析形势、选择适当的战术、精准执行并最终获胜。如果他们没有采用让北卡罗来纳队措手不及的战术，结局可能会全然不同。

同样，IT 组织也需要有可执行的方案。对于日复一日的应用交付与运维，可以在开发、交付以及运维流程中获得这些所谓的方案。成功的 IT 组织都拥有良好的流程并以卓越的方式执行这些流程。然而，IT 组织的变革是另外一回事。没有能够克服文化及组织惰性的明确定义的成功方案，大多数组织都要与变革作斗争。针对大型企业内的 DevOps 实施以及大型、复杂、分布式 IT 组织的 DevOps 变革，本书提供了一套行之有效的、可重复使用的方案。

我曾帮助几十家规模及成熟度各不相同、来自不同行业和地域的组织实施 DevOps，本书中的这些方案均来自于我多年的实战经验。当我在 IBM 担任 DevOps 技术销售与实施全球首席技术官的时候，DevOps 还处在初期发展阶段，从那时起，我就前瞻性地看到了 DevOps 必将快速发展并日益成熟，会从初创企业的先驱实践发展为大型企业的文化及技术变革。我是 IBM 的 DevOps 先驱及思想领袖，而且面对 IBM 的客户时，我已成为 DevOps 的代言人。我研究了数百家客户在整个组织或企业范围内成功实施 DevOps 所做的工作与斗争，并提炼出成功的模式，纳入本书内容中。

在没有很多文化记忆的小型集中式组织中，实施 DevOps 并不困难。即使在大型组织中，小型团队，即众所周知的两个比萨团队[①]，也通常能成功取得 DevOps 所承诺的业务成果。在大多数组织中，都能见到这样的付出与努力，而且多数都取得了成功。提取个人与独立团队层面的成功经验，并将其推广至整个企业，这是一项挑战。就像组织中有一系列的小舞团。然而，这些舞团都是独一无二的。有的跳萨尔萨，有的跳

[①] 亚马逊首席执行官杰夫·贝佐斯声称，一个不能用两份比萨喂饱的团队就太大了，以致无法成为富有成效的团队。

爵士舞，有的跳交际舞，还有其他一些人在跳我女儿称为“嘻哈”的舞蹈。他们无法组合起来，并成长为一个可以在下半场演出，占满整个体育场表演场地的庞大舞团，因为要做到这点，他们不仅需要相同的舞曲，还要统一舞蹈的表演形式。同样，小型独立团队无法影响整个组织。这些团队需要努力实现相关实践、流程、平台与工具的标准化以便复制给组织的其他团队。

反过来，组织需要为 DevOps 实施建立适当的环境，这可以通过支持变革努力、改变僵化的遗留程序以及自上而下共同克服文化惰性来实现。

注意：自下而上员工导向的努力使得非常高效的独立团队能够实施 DevOps 并茁壮成长；自上而下高层管理者导向的努力使这些个体的成功能够推广开来。

要推广成功，业务的参与是必不可少的。IT 组织的存在就是为了交付业务向其客户交付商业价值所需要的能力。业务要求 IT 组织进行优化，要更加敏捷，适应变化，更具有响应力，利用更少的资源做更多的事情，更加高效，提高产能，更快速、更高质的交付，针对市场灵活应变，加速竞争，遵守不断变化的监管与合规制度以及，当然也要缩减开支。

此外，还可能要求创新，以允许公司进入新的市场，实现指数增长，吸引并发展客户群体，响应客户需求以及缩减开支。这些要求(希望这些要求不是同时存在)是变革需求产生的驱动力，是实现 DevOps 实施效益的工作动机。

注意：实施 DevOps 不能仅仅因为它很炫酷，而是要基于业务上的原因。敏捷或速度的诉求是 DevOps 存在的第一性原理。在过去几年里，DevOps 的实施日趋成熟与广泛，这反映出当今的市场动态和客户期望。

因此，为了使 IT 经历变革，这个变化必须要能够改进并增强其能力，以交付业务能力并进一步改进及提升所交付的业务价值。业务与 IT 之间必须保持适度合作，以便 IT 实施 DevOps 所经历的变革能够通过适当平衡优化与创新来满足业务最为迫切的需求。业务目标必须是驱动 IT 变革的原因，变革原因又会反过来驱动 IT 变革的方式。

本书将 DevOps 实施方案分为以下几类：

- 优化 DevOps
- 创新 DevOps
- DevOps 实施的企业级推广
- 驱动企业的 DevOps 实施

其中包括了每一项实施方案的经验教训、案例、成功模式以及反模式。就像体育运动队的战术手册一样，本书旨在提供适用于不同情境及形势的特定方案(具体取决于组织当前的成熟度及状态)，供组织通过实施 DevOps 向更高绩效的交付组织转型时使用。组织需要采用这些方案，并基于 DevOps 实施项目与团队的实际情况策略性地执行。正如与敌人交战必须要有作战计划一样，实施这些方案必须要有相应的行动计划或者使用更为广泛的为每个组织设计的实施路线图。

此外，任何组织在本质上都不是统一或同质的。组织中的某个部分或许在某些领域更加成熟，而在其他领域则不够成熟。同在一个组织内，有时甚至是同在一幢建筑物内，一些团队或群体或许已经实现了敏捷与速度，而其他团队则可能正经受严重的文化惰性；他们都需要相互合作以实现规模效应。

组织可能已经拥有应用现代化敏捷与 DevOps 实践的创新实验室，但核心系统团队可能仍在进行僵化的瀑布式交付。因此，针对同一组织中的不同部分，要应用不同的实施模式，而且要根据不同团队的需要进行定制化调整。为了帮助组织实施这种定制化调整，本书还应用了价值流图技术。作为精益实践的组成部分，价值流图已应用了几十年，现在也可以用来从这些方案中开发实施路线图，这些实施路线图是根据组织的业务目标、当前成熟度以及能力状况而定制的。

颠覆还是被颠覆，这是值得思考的问题

我们生活在一个巨变的时代。1960 年，财富 500 强企业的平均预期寿命是 75 年。而如今，企业平均寿命只有 15 年，而且在未来还会进一

步下降。为什么会这样呢？只需要看看所谓的“优步(Uber)效应”就能理解为什么这么多的公司都倒闭了。优步的创始人并非出租车行业出身，他利用移动应用上的触摸按键，带领公司颠覆了一个历史悠久的产业。他们使出租车服务成为按需服务经济的一部分，消费者的需求可以随时得到满足，没有任何延迟。诸如敏捷、DevOps 这样的新方法以及云计算、微服务这样的新技术，加速催生了新的 IT 能力，这些能力使初创企业也可以拥有云端及移动应用的服务，与进行大量 IT 投资、拥有价值不菲的基础设施、员工经验丰富的 Uber 大型组织没有差别。

注意：世界上发展最快的运输公司自己却没有一辆车(Uber/优步)；发展最快的酒店管理公司提供房屋出租，却没有任何产权属于自己(Airbnb/爱彼迎)；世界上发展最快的传媒企业不制作任何的媒体内容(Facebook)；世界上最大的百科全书没有全职作者(Wikipedia/维基百科)。颠覆已成事实。

所以，扪心自问，你的组织是颠覆者还是被颠覆者？事实是，多数组织都属于后者，这给今天的 IT 组织带来了前所未有的压力。无论是担心被竞争对手淘汰，还是基于加快步伐的业务需求，IT 组织都必须同时兼顾优化与创新，既要确保核心应用的优化运行，又不可以放弃创新。然而，事实上，创新与保持遗留系统的效率并不冲突。尽管与优步(Uber)、爱彼迎(Airbnb)这样诞生于互联网的公司竞争，似乎前景可畏，但在整个组织范围内实施 DevOps，能够让你的 IT 团队变得更加敏捷、高效及创新。实施 DevOps，能够使 IT 成为组织变革的推动者，从而抵御颠覆及干扰；反过来，这也会促进组织成为颠覆者。在当今技术驱动的世界里，IT 能力已经成为颠覆者与被颠覆者之间的关键差异。

定义 DevOps

在深入研究组织内实施 DevOps 时需要采用的核心能力与实践以及需要执行的各种方案前，必须首先理解 DevOps 的定义。

和行业内应用的任何一项新技术或技术相关运动一样，DevOps 已经成为一个过度流行的术语。每个人都在谈论它，却不是每个人都清楚它究竟是什么，最糟糕的是，许多声称正在实施 DevOps 的人，实际上工作产出一塌糊涂。也有一些优秀的案例，这些公司表现出色并处在 DevOps 运动的前沿，经常被当作示例的有 Etsy、Flickr、Facebook 以及 Netflix。但即便如此，关于什么才是真正的 DevOps 最佳实践，还是存在着争论和争议。Netflix 声称他们的做法是无运维(NoOps)，让开发人员来承担运维的责任。但是也有人反驳说这样的情况会导致混乱与无序。

随着 DevOps 的发展，行业在完善 DevOps 定义的过程中，存在这样的争论是意料之中的。正如我在本书中将利用大量篇幅来讨论的，实施 DevOps 有各种不同的方法，每个组织应该基于各自的风险值与业务驱动的平衡去选择适当的实施能力与 DevOps 实践。事实上，DevOps 实施需要先从试点项目开始启动，然后利用本书中介绍的技术在整个企业范围内推广。

如前所述，DevOps 的定义有很多，或者至少是关于“DevOps 究竟是什么”的观点有很多，因为有很多的博客和技术“专家”在谈论这件事情。关于 DevOps 的观点中，有的认为开发为王；有的认为持续交付是驱动力；有的认为一切都取决于云，没有云计算，就没有真正的 DevOps；有的认为 DevOps 就等同于微服务。所以，我们先来看一下(相对)中立的来源——维基百科上关于 DevOps 的定义(维基百科，2016)：

DevOps(开发 Development 与运维 Operations 的组合词)是一种文化、一场运动或实践，强调在自动化软件交付流程及基础设施变更过程中，软件开发人员与其他信息技术(IT)专业人员彼此之间的协作与沟通。它旨在建立一种文化与环境，使构建、测试、软件发布得以快速、频繁以及更加稳定地进行。

有一点要特别注意的是，随着 DevOps 日益成熟，维基百科上的 DevOps 定义也在不断修正。可以比较一下，这是 2013 年维基百科上的定义：

DevOps (开发 Development 与运维 Operations 的复合词)是一种软件开发方法，强调的是软件开发人员与信息技术(IT)专业人员之间的协作、

沟通与整合。DevOps 是软件开发与 IT 运维之间相互依赖的一种反应。它旨在帮助组织快速产出软件产品及服务。

维基百科定义的这种演变也代表了 DevOps 本身以及行业看待 DevOps 观点的变化。除了替换掉深奥的词汇 *portmanteau*，这个词每个人都要到 Dictionary.com 上去查询才能理解，还需要注意的要点有：

- 将“软件开发方法”替换为“文化、运动或实践”
- 增加了“自动化”一词
- 将最终目标由“快速产出软件产品及服务”替换为“建立一种文化与环境，使构建、测试、软件发布得以快速、频繁以及更加稳定地进行”。由此，DevOps 变革的目标从仅仅是速度变为速度、稳定性以及质量。

当然，史上最为简明的 DevOps 定义绝对不能不提，这是在 2013 年 O'Reilly Velocity 会议的 T 恤衫上看到的：

> DevOps——将命令行(SH)从 IT 工作中赶走

本书读者对象

体育运动队中不是只有比赛当天上场的运动员们，还包括教练、助理教练、团队管理者、团队高层、培训师、医生、营养师、物理治疗师、器械经理再到运球人员及供水服务员的所有这些人。团队要想发挥出最佳水平，所有这些人都是必不可少的，而且每个人都需要在各自的角色中出色发挥，并作为一个团队有效合作。同样，DevOps 也不仅仅关乎开发与运维人员，它要求应用交付流水线中的所有利益相关者都要改变其工作方式、协作与沟通的方式，以及像高绩效团队一样有效合作的方式。

《DevOps 实施手册　在多级 IT 企业中使用 DevOps》是写给组织中的所有团队成员的，他们是应用交付流水线中的利益相关者，从业务线

负责人，到分析师、架构师、设计师、开发人员、测试人员、自动化工程师、基础设施工程师、运维人员、数据库管理员、系统管理员、文档编写人员、项目经理、产品负责人，一直到高层管理者，都包括在内。这些角色可能因组织而异，组织实施 DevOps 时，还需要对很多角色的工作内容以及工作方式进行完善与调整。本书旨在让所有这些人受益。

实施所讨论的每个方案都会对不同角色的利益相关者产生不同的影响，有些利益相关者可能会在角色以及与他人交互的方式上发生重大改变，有些人则看不到任何变化。所有的运球人员以及供水服务员通常都不会受到球队战术的影响，但是他们仍然是利益相关者，如果表现不符合预期，是会影响到球队的整体绩效的。IT 的支持性角色亦是如此。其他角色，比如直接利用构建物及流程开展工作的关键利益相关者，他们是应用交付流水线的组成部分，将从本书所列方案中明显受益。他们是方案执行人员，是为方案执行人员或支持人员提供直接支持服务的员工，以促进执行人员实现最高绩效能力。

第 1 章是 DevOps 的概述，介绍了 DevOps 从起源到今天的发展演变。其中定义并描述了构成 DevOps 的所有常见的实践与能力，为更广泛地定义 DevOps 以及 DevOps 变革奠定了基础，DevOps 变革是本书的主题。

第 2 章聚焦于球队的领导者：教练、队长以及组成团队核心的高级球员。重点关注的是如何评估比赛条件以及竞争对手，以开发并选择合适的战术集，即团队的战术手册。应用到 IT 管理中，指的是项目经理、产品负责人、团队领导、资深从业人员、DevOps 教练以及渴望成为其中一员的任何人。

第 3 章就如何为 DevOps 变革创建商业案例，以取得适当的支持与投资来确保成功，提供了相应指导。

第 4 章到第 6 章是真正的方案。具体内容分为：

- 第 4 章——用于优化的 DevOps 方案：通过消除浪费，优化应用交付流水线以实现效率最大化的方案。
- 第 5 章——用于创新的 DevOps 方案：使应用交付流水线更加快速、敏捷以支持实验能力，驱动创新的方案。

- 第 6 章——用于 DevOps 企业级推广的 DevOps 方案：在大型、复杂、分布式且成熟度不统一的组织中，将 DevOps 实施在整个组织范围内推广的方案。

第 7 章是献给驱动 DevOps 实施的高层管理者的。就如同体育运动队的总经理或高级管理者，高管们做出行政决策，确定组织的方向与文化。他们是需要做出 DevOps 变革决策的人。他们需要为变革提供必要的投资与支持。他们还需要了解如何创建商业案例以确定变革的投资回报率。他们需要推动变革，从前沿试点团队直到整个组织。

本书还有一个附录。是一个 DevOps 变革实施路线图的案例，是为假想的银行开发交付价值流图实践。

本书试图有意保留工具与平台的兼容性。尽管书中列举了一些工具、平台以及技术作为例子，无论是商业化的还是开源的，但这都是一些目前市场上可用于实现自动化的工具及平台。要实现流程自动化，使其高效、可重复、可扩展并且准确无误，工具必不可少。然而，工具和平台都是不断发展变化的，取而代之的是更新更好的版本。因此，推荐甚至记录现有的工具和平台是徒劳之举。我们的目标是强调能力，同时尽可能保持工具与平台的兼容性，即便现有的工具与平台仍在不断改进。

体育运动的比喻

> 正是由于个人对于集体做出的贡献，团队、公司才能够运行，社会、文化才得以发展。
>
> ——文斯·隆巴迪，美式橄榄球传奇教练

没有什么比体育运动更能超越文化、语言及地理的界限。如果有所怀疑，只要再看一遍里约奥运会的重播即可。来自体育项目的比喻也完全适用于应用的开发与交付，因为二者都是团队活动。开发或交付新的应用或服务，可能不要求奥运会金牌得主的体能训练，但一定需要领导力、沟通、协作以及信任，这是任何一项团体运动都需要的。

我个人对运动也极富热情。从小就是在一个热爱运动的家庭中长大的。我的外祖父是印度的奥运级及国家级体育选手，他年轻时曾效力于印度国家曲棍球队，后来在 1952 年赫尔辛基奥运会上担任印度国家足球队的体育主管。1964 年的东京奥运会，火炬传递经过印度时，他还担任了奥运火炬手。此后的几十年间，包括我的童年时期，他一直是国内足球联赛的体育主管。在家中伴随着奥运火炬长大，使我们都很尊重运动员及妇女。

书中我列举、引用了多个运动项目的运动员与教练的经验，并将其映射到 DevOps 实施中。这种类比旨在使方案更加贴切易懂，并希望能够提升阅读的趣味性。



图 1　Major Lachhman Singh 曾是 64 年东京奥运会火炬手(来源：Singh 家族的私人收藏)

目　录

第 1 章　DevOps 概述 …… 1
1.1　DevOps：起源 …… 2
1.2　DevOps：本源 …… 4
1.3　DevOps：实践 …… 10
1.3.1　持续集成 …… 11
1.3.2　持续交付 …… 15
1.3.3　支持实践 …… 19
1.3.4　前移 …… 27
1.3.5　架构与降低风险 … 30
1.3.6　持续改进 …… 31
1.3.7　衡量标准 …… 31
1.3.8　业务驱动 …… 32
1.4　DevOps：文化 …… 33
1.5　总结 …… 35

第 2 章　DevOps 实施 …… 37
2.1　撰写指导手册 …… 39
2.1.1　识别目标状态(业务目标及驱动) … 40
2.1.2　评估现状 …… 43
2.1.3　选择变革方案 …… 56
2.1.4　实施变革方案 …… 57
2.2　总结 …… 61

第 3 章　开发 DevOps 变革的商业案例 …… 63
3.1　开发商业案例 …… 64
3.2　完成商业模式画布 … 67
3.3　客户细分 …… 68
3.3.1　业务线 …… 68
3.3.2　IT 组织 …… 69
3.4　价值主张 …… 70
3.4.1　业务线 …… 70
3.4.2　IT 组织 …… 72
3.5　渠道通路 …… 74
3.5.1　业务线 …… 74
3.5.2　IT 组织 …… 75
3.6　客户关系 …… 75
3.6.1　业务线 …… 75
3.6.2　IT 组织 …… 75
3.7　收入来源 …… 75
3.7.1　业务线 …… 76
3.7.2　IT 组织 …… 76
3.8　核心资源 …… 76
3.8.1　业务线 …… 76
3.8.2　IT 组织 …… 77

3.9 关键业务 …… 77
3.9.1 业务线 …… 77
3.9.2 IT 组织 …… 77
3.10 战略伙伴 …… 78
3.10.1 业务线 …… 78
3.10.2 IT 组织 …… 79
3.11 成本结构 …… 79
3.11.1 业务线 …… 79
3.11.2 IT 组织 …… 79
3.12 总结 …… 80

第 4 章 DevOps 方案之优化持续交付流水线 …… 81
4.1 DevOps 作为优化运动 …… 82
4.2 核心主题 …… 88
4.2.1 缩短周期时间 …… 89
4.2.2 缩小批次规模 …… 91
4.2.3 建设正确文化理念 …… 95
4.3 DevOps 实施方案 …… 99
4.3.1 方案：建设衡量标准与关键绩效指标 …… 99
4.3.2 方案：敏捷实施 …… 107
4.3.3 方案：集成的交付流水线 …… 110
4.3.4 方案：持续集成 …… 116
4.3.5 方案：持续交付 …… 120
4.3.6 方案：测试前移 …… 133
4.3.7 方案：运维参与前移 …… 139
4.3.8 方案：持续监控与反馈 …… 145
4.3.9 方案：发布管理 …… 151
4.4 专注核心方案 …… 154
4.4.1 方案：移动设备 DevOps …… 154
4.4.2 方案：大型机的 DevOps …… 161
4.4.3 方案：物联网 DevOps …… 165
4.4.4 方案：DevOps 用于大数据及分析 …… 168
4.5 总结 …… 173

第 5 章 DevOps 驱动创新方案 …… 175
5.1 优化创新 …… 176
5.2 Uber 综合症 …… 178
5.3 创新与技术的角色 …… 178
5.3.1 商业模式创新 … 179
5.3.2 商业模式实验 … 180

5.3.3 用户参与模式创新……181
5.4 核心主题……183
5.4.1 实现多级 IT……184
5.4.2 构建正确的事物……187
5.4.3 进行实验……190
5.4.4 提供反脆弱的系统……192
5.4.5 IT 系统与反脆弱性……195
5.5 方案：构建 DevOps 平台……199
5.5.1 应用交付与反脆弱性……202
5.5.2 环境抽象层……203
5.5.3 云托管的 DevOps 平台……204
5.5.4 基础设施即服务……209
5.5.5 OpenStack Heat 作为抽象层……214
5.5.6 平台即服务……215
5.5.7 容器……219
5.6 方案：交付微服务架构……223
5.6.1 微服务架构……224
5.6.2 应用的 12 要素…226
5.6.3 云原生应用……228
5.6.4 微服务和容器…230
5.6.5 微服务化改造…230
5.7 方案：API 经济……233
5.7.1 部署自动化和 API……236
5.7.2 DevOps 平台和 API……236
5.8 方案：组织创新……238
5.9 总结……240
第 6 章 DevOps 的企业级推广……243
6.1 核心主题……244
6.1.1 组织文化……245
6.1.2 工具与实践标准化……246
6.1.3 有组织的实施…247
6.1.4 打破组织仓筒…248
6.2 方案：DevOps 能力中心……248
6.2.1 DevOps 能力中心的功能与目标…250
6.2.2 能力中心的核心角色……251
6.2.3 DevOps 教练……251
6.2.4 建立能力中心…253
6.3 方案：发展规模创新文化……254
6.4 方案：发展持续改进文化……259
6.4.1 开发实施路线图……261

6.4.2 持续开发与价值流图……262
6.5 方案：DevOps 团队模型……264
6.6 方案：工具与流程标准化……267
6.7 方案：DevOps 的安全性考虑……271
6.7.1 管理安全相关风险……273
6.7.2 解决 DevOps 流程与平台的安全问题……275
6.7.3 API 经济与安全……279
6.8 方案：DevOps 与外包……280
6.8.1 战略外包……281
6.8.2 IT 供应链……282
6.8.3 利用外包实现 DevOps……283
6.9 总结……283
第 7 章 引领企业的 DevOps 实施……285
7.1 方案：DevOps 作为变革运动……287
7.1.1 令人信服的行动理由……289
7.1.2 DevOps 变革的反模式……290
7.2 方案：发展协作信任的文化……293
7.2.1 可见性促进信任……294
7.2.2 一切都关乎人…295
7.3 方案：业务线的 DevOps 思维……296
7.3.1 业务线与 IT 的接触……297
7.3.2 参与 DevOps 变革……298
7.3.3 让影子 IT 走出阴影……298
7.4 方案：利用试点项目启动……299
7.4.1 试点项目选择…301
7.4.2 高层管理者支持……302
7.5 方案：在航空母舰上培养独角兽……302
7.6 总结……306
附录 A 案例研究……307
A.1 组织背景……307
A.2 路线图组成……308
A.2.1 DevOps 的优化与创新工作坊……309
A.2.2 背景和上下文…310

A.3　实施路线图 …… 312
A.3.1　业务驱动因素 …… 312
A.3.2　现有的 IT 举措 …… 313
A.3.3　瓶颈 …… 314
A.3.4　根因分析 …… 316
A.3.5　DevOps 实践 …… 316
A.3.6　实施路线图 …… 321
参考文献 …… 323

第 1 章

DevOps 概述

开发经理的咆哮

事情是这样的，开发人员在周一下午完成了新服务的编码工作。她构建代码、运行单元测试，并将代码提交到集成流中以便进行持续集成(CI)构建。为了测试其服务，开发人员在下班前给测试(QA)团队开了一个任务单。

星期二早上，QA 团队进来看到了这个分配给他们的任务单。一个测试人员拿到了任务单并给开发人员发邮件请求部署说明。由于没有自动化部署，开发人员回应说她将亲自部署服务到 QA 环境中。星期二下午，他们参加电话会议进行代码部署。开发人员发现测试环境与其代码不兼容。他们需要一个新环境。星期二晚上，测试人员开了一个任务单给运维(Ops)团队申请一个不同规格的新环境。

星期三早上，运维团队把工作单分配给处理规格和处理防火墙端口变更的工程师。当他去吃午餐的时候，他开出一个任务单给安全团队以批准端口变更。星期三下午，安全团队给安全工程师开出任务单，安全工程师批准了变更。星期三晚上，运维工程师收到批准，并开始构建新环境 。他需要手动构建虚拟机(VM)，其中包括操作系统(OS)、应用服务器、数据库以及 Web 服务器。

星期四早晨，服务器构建完成，任务单关闭。测试人员再次发邮件

给开发人员部署新服务。开发人员部署了新服务，并且测试人员开始准备可用的测试脚本。他需要运行回归测试，但是他需要额外的数据进行重新测试。星期四下午，他给生产支持团队开出一个申请新测试数据任务单。

星期五早晨，生产支持团队指派了一名数据库分析师(DBA)从生产环境收集数据。但此时已经是星期五下午了。每个人都知道 DBA 们在星期五的下午是不工作的。星期一的早晨，测试人员从 DBA 手中得到了测试数据。他用了 20 分钟时间运行回归测试，并且发现一个缺陷。他把任务单退回给开发人员，在代码编写构建完成整整一周之后。在这份代码之上整整一周的编码工作等待处理，不知道是否存在缺陷。我们现在又落后一周了。

这个故事的可怕之处在于，当我把它讲给其他公司的同行听时，他们并不感同身受，只是惊讶：相比他们，我们竟然如此高效！

——又一个沮丧的开发经理

1.1 DevOps：起源

DevOps 运动始于约翰·阿尔斯帕瓦(John Allspaw)和保罗·哈蒙德(Paul Hammond)(当时他们都服务于 Flickr/Yahoo)在 2009 O'Reilly 敏捷大会上的一次具有开创性意义的演讲。演讲的题目是《每天 10+部署：开发与运维在 Flickr 的协作》[①]。每天 10 次部署被认为是史无前例的。同年，帕特里克·德布瓦(Patrick Debois)在比利时根特组织第一次 DevOpsDays 活动时，最终将他们的方法称为 DevOps。

这个名称由此开始流行起来并引发极大关注，但最初仅限于初创公司，更具体地说，是那些提供 Web 应用的组织。这些应用由开发人员(Dev)创建，他们通常以非常迅速的方式向其 Web 应用提交变更。他们所面临的主要障碍来自于运维(Ops)团队，由于变更管理流程僵化而且苛刻，

① http://conferences.oreilly.com/velocity/velocity2009/public/schedule/detail/7641。

以致运维团队在部署变更时速度非常缓慢。

DevOps 运动的目标就是要解决开发与运维团队之间的阻抗失衡；弥合彼此之间的鸿沟；促进更多沟通、合作与信任。从本质上讲，这是一次文化运动，致力于改变开发与运维团队之间的文化差异，通过自动化使应用交付更快速、更高效并最终做到持续交付。2010 年，当时在 ThoughtWorks 公司工作的杰兹·汉姆(Jez Humble)，在他的著作《持续交付》中编写了一些 DevOps 的核心实践，使得 DevOps 的实施变得切实可行，由此，将 DevOps 介绍给全行业的从业人员。

不过，DevOps 被看作是那些独角兽公司才做的事情，也就是那些处在创新前沿的，没有大型的、复杂的遗留系统需要维护的初创型组织才做的事情，还没有为大型企业所关注。然而，这些大型企业看到了初创公司运用 DevOps 所取得的成功，并且试图实施 DevOps 以满足其自身需求。像 IBM 这样的组织开始涉足自动化部署以及虚拟化环境架构，甚至将两种能力相融合。与此同时，像 UrbanCode 这样在自动化构建领域信誉卓著的公司，随着 uDeploy 的发布，也开始转向 DevOps，由此创建了一类新的持续交付工具。自动化领域的其他公司，比如 Nolio 也加入行列之中并提供自己的竞争产品。同时，运维和基础设施即代码(infrastructure as code)方面的公司，如 Opscode(现在称为 Chef)和 Puppet Labs 也受到关注(Opscode 的 Chef 和 Puppet Labs 的 Puppet)。

2012 年，IBM 利用 SmartCloud Continuous Delivery 开始了其第一项短暂的持续交付实验，自此，DevOps 开始真正走进了大型企业。一些咨询公司，比如 Thoughtworks 和 IBM，也开始为组织，尤其是想要实施 DevOps 的大型组织提供咨询服务，协助其将独角兽公司的成功经验转换成企业适用模式。通过分别收购(巧合的是在 2013 年 4 月的同一天)UrbanCode 与 Nolio，IBM 和 CA Technologies 宣布正式进军 DevOps 领域。然而，DevOps 运动有史以来最重要的转折点是发生在 2013 年基恩·金(Gene Kim)的著作《凤凰项目》(Phoenix Project)出版的时候。这本书的灵感来源于艾利·高德拉特(Eliyahu M. Goldratt)的著作《目标》，正如高德拉特的著作曾领先制造业几十年的时间，《凤凰项目》现已成为现代 IT 界贯彻执行精益实践以及高德拉特约束理论的必读书籍。利

用这本书，还有后来每年与杰兹·汉姆和帕比特·莱博斯联合发布的《DevOps 现状调查报告》，基恩·金使 DevOps 真正成为主流。

1.2 DevOps：本源

DevOps 从哪里来？虽然我已经大概讲述了其起源的故事，但 DevOps 真正的起源要早于 Allspaw、Debois、Humble 和 Kim 将近一个世纪的时间。这需要追溯到发源于 20 世纪 10 年代的精益运动。

随着亨利·福特在 T 型车生产线中应用了精益流程管理，精益运动开始在制造业兴起。自 20 世纪 30 年代起，丰田的丰田喜一郎(Kiichiro Toyoda)与大野耐一(Taiichi Ohno)将这项工作进一步扩展、细化及整理，二战后得以真正迅速发展起来。20 世纪 50 年代，这项工作受到戴明博士(Dr. William E. Deming)提出的计划-执行-检查-处理(Plan–Do–Check–Act，PDCA)循环理论的影响不断改进，以持续提高生产质量。基于其最为核心的方法，精益生产运动旨在持续改进产品质量的同时减少生产过程当中的浪费。在詹姆斯·P. 沃麦克(James P. Womack)和丹尼尔·T. 琼斯(Daniel T. Jones)1990 年出版的《改变世界的机器》(必读)以及 1996 年出版的《精益思想》(Lean.org，2016)这两本著作中，精益得到进一步的诠释。

戴明的精益思想与持续改进

戴明博士教导说，通过采取适当的管理原则，组织可以在提高质量的同时降低成本(通过减少浪费、返工、员工流失及客户投诉，同时提高客户的忠诚度)。关键是要实施持续改进，并且视生产为一个完整系统，而非零零碎碎的 。

——戴明博士的管理培训(戴明，1998)

2001 年，敏捷出现了。包括阿利斯泰尔·库克伯恩(Alistair Cockburn)

和马丁 • 福勒(Martin Fowler)在内的 17 个思想领袖发表了敏捷宣言[②]。宣言的核心是摆脱僵化、瀑布式的、文档化的软件开发世界，这是大多少数软件开发项目延期、超预算或是惨痛失败的原因。

他们的目标是找到一种迭代的方法，实现与客户、终端用户或代理人的持续互动。他们不想再通过诸如需求文档之类的主要硬性指标来衡量进度，因为这类指标不会提升代码的交付速度。另一目标是利用实际运行的代码(运行的软件)作为衡量进度的真正标准；审视计划与实际进展的匹配程度；创建不需要预先编写，但会随着应用的开发而改进与完善的需求。

随着极限编程(Extreme Programming，XP)、敏捷软件开发(Scrum)以及近期出现的规模化敏捷框架或是 SAFe 这些方法论的发展，敏捷得到进一步提升。今天，不论大型还是小型的组织都在利用敏捷方法交付各种规模与技术的项目。

敏捷是 DevOps 需求的前驱，并已成为核心的驱动力。随着开发人员加快代码交付速度，代码需要更快速地测试；还需要更加频繁地部署到开发及测试服务器，并最终部署到生产环境。但是运维团队并没为此做好准备，这导致了开发到测试的交接过程中出现了瓶颈，因为测试无法根据需要按时创建适当的测试环境，更为重要的是发布时的生产环境。产品发布仍然是一个重大的任务，“发布周末”通常都会持续到周末以后。

发布周末

记得 90 年代早期我在一家金融机构从事开发工作(当时称为银行后台)，在周末发布时，令我非常懊恼的是，我们被要求在周五的早上带着睡袋去工作。我们预计整个周末都要待在那了。开放了多间设有电话会议桥的会议室，让每一个团队彼此沟通。一个会议室就如同一个作战室，项目负责人通过一个大型的电子表格协调所有利益相关者。管理层尽最大努力想要营造一种聚会的气氛，但最初的几小时过后就烟消云散

② http://www.agilemanifesto.org

了。我们第一次和运维人员进行交流。我们把代码交给了此前从未见过这些代码的人。他们使用我们所不熟悉的脚本与工具，将代码部署到我们无法访问的环境中。整个周末都一团糟。大量的食品和变味的咖啡，似乎一切都没有按计划进行。我们所支持的业务交易人员都是很聪明的。他们总是在接下来的周一安排一次郊游或野餐。他们知道一切都会停摆，而且他们是对的。幸运的是，我们一年只有两次这样的情况。对我个人而言更为幸运的是，我在那里工作期间只赶上两次发布。

代码的短迭代快速开发，增强了开发与运维团队之间更好沟通与协作的需求。生产环境发布频频失败，反映出开发人员对类生产环境的访问需求。仅仅提高流程中的一个部分，即开发代码的效率，使整个流程的低效暴露无遗，这就导致测试与运维成为主要瓶颈。如果把应用开发及交付流程比作工厂里的生产线，在下游的工序仍然低效工作的情况下，仅仅加快其中某一道工序的工作速度来增加零部件的数量，对提高整条生产线的交付速度没有任何帮助，只会制造更多的积压而已 (如图 1-1 所示) 。这不仅是运维的挑战，也是交付生命周期中所有利益相关者的挑战。

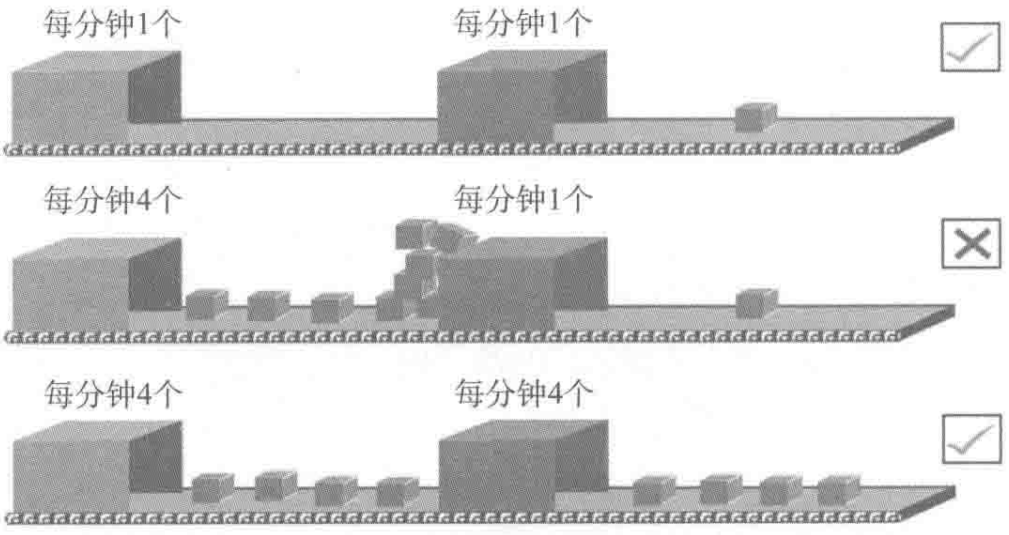


图 1-1　交付流水线瓶颈

如今的焦点转向了周期时间最小化。周期时间指的是从需求或者用户故事开始到把功能交付到客户手中，或者至少是完成集成、测试并已

为部署到客户机做好准备的时间。这导致了 DevOps 两项核心能力的发展：持续集成(已经是敏捷的核心竞争力)与持续交付。稍后将详细讨论这两项能力。将运维团队纳入交付生命周期中，作为流程的一个部分，而非在代码发布前都不参与进来的独立仓筒，这种超越开发-测试周期的敏捷扩展已成为 DevOps 的核心原则。

解决开发与运维的对立

开发与运维在传统开发过程中位于不同的仓筒中，有着不一致甚至是对立的优先级。开发(Dev)的任务是提供创新并尽快交付给用户。运维(Ops)的任务是确保用户可以访问稳定、快速的响应系统。尽管开发和运维的最终目标都是让用户满意其提供的系统，但对于如何达成这个目标，他们的观点在本质上却是对立的。任何开发人员都不是故意制造一个玩具车一样的系统，当用户使用时，应用就崩溃。所有运维人员都希望开发人员能不断更新产品，使其具有更新、更令人兴奋的性能及功能。这就是他们的差异所在。这是一种典型的情形，被称为敏捷瀑布(water-Scrum-fall，Forrester, 2011)。开发人员想要并期望能快速为产品提供新功能。运维想要并期望能创造一个在任何时候都稳定运行的系统。

1. 开发对运维

在敏捷出现之前，在纯粹的瀑布型案例中，开发人员和运维人员生活在真正独立的世界中时，这些对立的优先级并不是那么严重的问题。开发和运维人员的工作日程只是在发布时有限交互。开发人员清楚发布日期，并且只在发布日才发布新的功能。如果截至发布日期，他们没能开发出新的功能，那就只能等待下一趟发布列车了。运维人员清楚发布列车何时到来，所以，在部署之前，他们有足够的时间对新功能进行测试，并且可以用几天(周末)的时间将它们部署到客户机。对于大型系统，他们甚至可以长期进行分阶段部署，保持稳定性。

敏捷改变了所有这一切。利用持续集成(CI)，开发人员现在可以每天部署新的功能。不需要再等待发布列车；这是一条时刻运转的传送带(流水线)。现在开发人员希望其开发的新功能能够在开发环境、测试环

境以及最终的生产环境中按照生产与集成同步的频率良好运行。他们希望运维人员能支持所有这些新版本。

运维人员现在要处理的不再只是偶尔的一次发布，而是持续不断的密集 CI 构建。这些构建可能做好了也可能还没有做好部署准备，但必须由运维人员进行管理并部署到测试以及最终的生产环境中。现在运维人员更关心质量。开发人员与测试人员关心的是如何快速获得开发及测试环境，以及这些环境是否与生产环境类似。他们不想在一个功能和表现与生产环境迥异的环境中测试其代码。因此，运维人员不可以再花上几天时间去配置及构建开发、测试以及最终的生产新环境。他们做这一切的同时，仍然保持生产环境稳定和可靠。他们必须在保持生产系统稳定性与可靠性的同时做到这一切。

周期时间？

如果迭代开发(Scrums)周期是两周，却需要三个星期时间才能取得新的测试服务器，那么这个迭代周期是多长？

2. 开发与运维

开发人员与运维人员之间这场战争的解决方案正是 DevOps 要给出的，即：实现创新与稳定、交付速度与质量之间的平衡。要实现这一点，开发人员和运维人员都需要改进其工作方式以相互融合。

开发视角　之前的内容可能会让人产生一种运维人员需要比开发人员改变更多的印象，但是开发人员也同样需要做出一些改变：

- 开发人员需要与运维人员合作，了解其应用所运行的生产系统的性质。生产系统(环境模式)的标准是什么？他们的应用在系统中如何执行？应用的运维有哪些约束条件？现在，开发人员需要了解系统以及企业架构。
- 开发人员应该更多地参与测试工作。这意味着不仅是要确保代码零缺陷，而且还要测试应用，看看其在生产环境中将如何运行。这就要求开发人员与测试人员(QA)紧密合作，并且在类生产环境中对其应用进行测试。

- 开发人员还需要学习如何监控部署后的应用，并理解运维人员所关心的衡量指标。他们要能够破译程序是如何交互的，以及一个进程如何导致另外一个进程慢下来甚至崩溃。他们需要理解代码的变化如何影响整个生产系统，而非仅仅影响自己的应用。
- 开发人员需要和运维人员更好地沟通、协作。

运维视角　运维人员需要具备快速配置新环境的能力，他们需要构建系统以响应快速变化。

- 运维人员需要清楚代码的来源及其对系统可能产生的影响。这就要求他们从了解应用的需求以及系统规格开始，就要参与开发协同。这个过程在精益与 DevOps 中称为前移(shift left)。他们需要确保其系统能够适应这些增强版的应用。
- 运维人员需要自动化系统管理方式。没有自动化，就无法实现快速、稳定的变更。自动化不仅使快速变更得以实现，也能在系统崩溃时实现快速回滚。
- 理想情况下，运维人员需要版本化其系统。只有当基础设施以及所有的变更都可以作为版本控制的代码获取并管理的时候，才能完成这样的版本化。因此，他们需要引入基础设施即代码实践，软件定义的环境就更好。
- 无论在哪种运维团队所管理的环境下，运维人员都需要监控整个交付流水线中的一切。他们需要在最短的时间内发现潜在的不稳定因素。
- 运维人员需要与开发人员更好地沟通、协作。

简而言之，开发人员和运维人员都需要纳入 DevOps 规范中。他们都需要知道，这不是一件容易的事，也不是一蹴而就的事。他们需要拟定计划并逐步实施变更以实现 DevOps 的承诺。开发人员与运维人员或许永远无法合成一个团队，至少在多数情况下无法实现，但是他们都需要理解自己的角色将随着 DevOps 的实施而发生改变。他们需要尽可能改变自己以进行协同工作，找到组织所需的开发与运维之间恰当的合作模式并进一步改善。

也就是说，开发与运维之间的分歧并不是快速交付生命周期的唯一障碍。交付生命周期中的所有利益相关者都需要进行更好的沟通与协作。

业务视角 来看一下业务的视角。最终，IT 通过交付的应用与服务提交的是业务需求。业务(业务线的需要更加明确)需要什么？

- 业务需要的是 IT 所交付应用与服务的状态的可见性。他们是否按时并依照预算交付了应用与服务？
- 业务需要应用交付团队就客户及终端用户如何使用应用与服务为其提供反馈。他们能否获得预期的商业价值？

关于业务视角以及 IT 期望方面更为详细的分析，以及 DevOps 如何为业务提供帮助，将在下一章详细讨论。

1.3 DevOps：实践

关于构建 DevOps 所需的功能，书中已经写了很多，博客文章中甚至更多。一些思想领袖把这些实践分为不同的类别，某些情况下甚至使用的是不同的名称。IBM 列举了几项这样的实践，均包括在以下几大类中：

- 思考
- 代码
- 交付
- 运行
- 管理
- 学习
- 文化

这种分类来自 IBM Garage Method[③]，这是一种专注于交付 Cloud Native 及 Hybrid Cloud 托管应用的采用 DevOps 实施的新方法论。

③ https://www.ibm.com/devops/method

DevOps 有两个关键核心能力：持续集成及持续交付。没有这两种能力，就没有 DevOps。这两种能力连同其他扩展能力或者支撑能力是实施 DevOps 所必须考虑到的。这两个概念聚焦于周期时间最小化。再来回顾一下周期时间的定义。

注意：

周期时间：从需求或者用户故事开始到把能力交付到客户手中，或者至少是完成集成、测试并已为部署到客户机做好准备的时间。

1.3.1　持续集成

而今，交付软件应用或系统涉及负责不同组件开发的多个团队。通常，已经完成的应用还需要与其他应用或服务交互才能实现其功能。其中一些外部应用或服务可能是企业中现存的遗留应用，或者是外部第三方的服务。因此，开发人员必然需要将自己的工作与其他开发团队构建的组件以及其他的应用与服务进行集成。

这种需求使得集成工作成为软件开发生命周期中的一项必要且复杂的任务。按固定节奏进行集成的过程通常称为持续集成，持续集成是敏捷的一项关键实践。在传统的开发流程中，集成是在组件(有时是完整的应用)构建之后才进行的次要任务。这样的顺序本质上是高成本且不可预测的，因为本来应该只在集成过程中发现的不兼容问题及缺陷，却要在开发流程的后期才能发现，这通常会导致返工及风险的剧增。

通过组件的持续集成，敏捷运动引入了一个逻辑步骤来降低这种风险。在这一步骤里，开发人员定期(至少每天)将其工作与团队中其他开发人员的工作集成并进行测试。在跨越多个平台、应用或服务的企业级系统中，开发人员还会尽可能频繁地与其他系统及服务进行集成。跨越多个团队与组件的持续集成案例如图 1-2 所示。

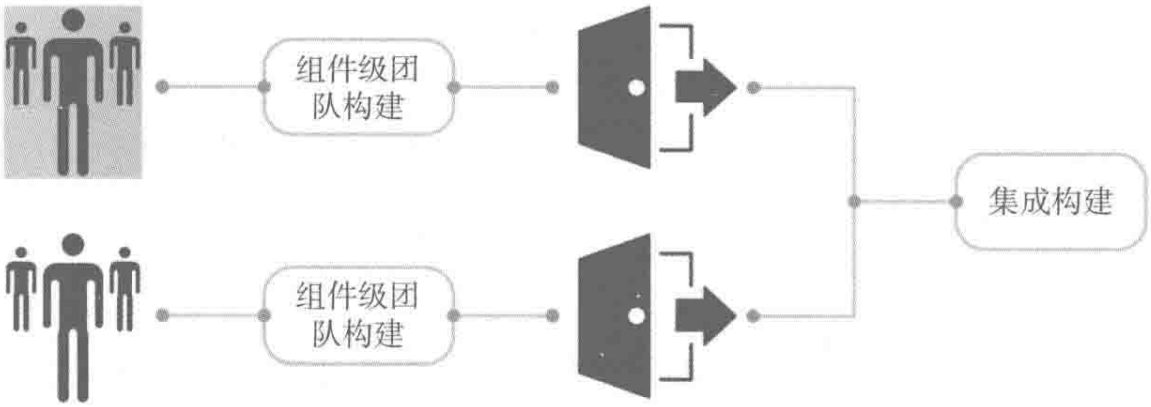


图 1-2　持续集成

这些集成结果的步骤使集成风险得以尽早发现并揭示出来。在企业级系统中，这些步骤还能够揭示出关于可能存在风险的技术或者日程的已知及未知的依赖关系。随着这些实践日益成熟，一些组织已采用持续集成实践，开发人员每次提交代码时都会遵循这些实践。在大多数成熟的组织中，持续集成会带来持续交付的能力，在持续交付流程中，代码和组件不仅集成，还会交付到类生产环境中进行测试和验证。这部分内容将在下一节讨论。

业务及客户对开发组织的需求推动了敏捷开发实践在开发团队的广泛应用。这些实践旨在缩小业务(或客户)与开发团队的分歧。开发团队主要通过三种途径来开展工作：

- 通过将开发工作分解成可以在迭代周期内完成的小块工作。这使得开发人员能够尽早地识别及解决风险，而不是等到完成整个项目或大部分工作的时候才能发现。
- 通过让最终用户或代表用户的代理在迭代周期中与开发团队互动。这有助于开发人员更好地理解用户需求，从而使变更需求更快上线。
- 通过在每一次迭代结束时发布软件。这使开发人员能够定期地演示所构建的最新应用，以便获得用户反馈。

如前所述，持续集成是这种敏捷开发的原则之一。使开发人员得以将其软件组件与内、外部的其他开发人员所开发的组件进行定期集成，以便尽早识别风险。

持续集成实践

马丁·福勒曾签署了著名的敏捷宣言，是持续集成流程开发的思想领袖。他把持续集成的概念分解成如下所述的十个实践：

(1) 维护同源的代码库。无论管理代码还是文件，使用版本管理工具来管理源代码是至关重要的，以实现多用户访问及代码流化[④]，或者分支开发与代码合并，还能够实现分布在不同地点的多位开发人员在同一组文件上工作。对于任何多平台的开发工作，使用通用的、跨平台的同源代码库就更加重要。如果这类代码库不能跨平台运行，那么任何独立的平台(例如，System Z 或移动平台)都将无法参与持续集成实践。在独立平台上进行的任何工作集成都将成为一种后处理、瀑布式的集成。

对于多年来可能一直使用相同功能的遗留系统开发团队而言，向现代源代码库的转换代表着巨大的变化。然而，同源代码管理(Source Code Management，SCM)工具对于管理所有的构建物，帮助打破仓筒以及消除关键瓶颈是至关重要的。

(2) 自动化构建。自动化构建使持续集成得以持续。此外，如果需要，应该还可以跨多个平台协调构建。

(3) 测试自动化。正如构建需要自动化，测试也同样需要自动化。持续集成的目标不仅仅是集成团队的工作，还要审视所构建的应用或系统在功能与性能方面的表现是否符合预期。这就需要为单元级测试、组件及应用级测试建立一套自动化测试脚本。在真正的持续集成中，开发人员是可以通过提交代码时启动适当的测试来发起集成构建的。这个流程要求构建脚本具备按需构建软件、提供测试服务器、配置测试环境、将软件部署到测试服务器、初始化测试数据以及运行正确的测试脚本这一系列能力。

要求配置可以进行构建、部署及随时进行自动化测试的环境有助于提高最终代码的质量。这需要系统资源可用，愿意定期运行大量自动化测试以及开发自动化测试。

(4) 确保每个人每天都能坚持主线交付。让跨越所有组件及所有开

---

④ Streaming：以流的方式访问代码和文档最新版本，而不是静态访问其分支上的某一个版本。

发环境的每一位开发人员每天提交代码到开发流的主线，这个目标有助于确保集成简单化。即使到了今天，许多开发人员仍然独立工作直到最终的构建完成，也只有这时，他们才意识到自己的工作是受到其他开发人员的工作影响的。这可能导致功能发布延期，或者上线前变更未经充分测试就部署到生产环境中。代码定期集成有助于确保迅速识别出这些依赖关系，以便开发团队能够及时处理，并且不需要受时间限制。

(5) 确保每次提交都在集成服务器构建主线。这是实践(4)的第二部分。确保每一次提交代码都执行了构建，并且运行了自动化回归测试，这样有助于确保在开发生命周期中更早地发现及解决问题。

(6) 保持快速构建。事实上，运行时间过长的构建才是持续集成的最大阻碍。由于构建的标准化实践就只是转换文件，使用现代的工具进行构建通常速度很快。

(7) 克隆生产环境中的测试。在无法准确代表生产系统的环境中进行测试会给系统带来很多风险。这一实践的目的是在克隆生产环境中进行测试。然而，也不太可能每次都仅仅为了测试而创建完整的多服务器克隆环境，创建一个有其他工作负载运行的克隆环境就更加困难。

这个实践要求创建类似的生产环境，称为“类生产环境”。类生产环境的规格应该尽可能接近生产环境。还应具备适当的测试数据管理能力。测试环境中不应该包含生产数据，因为在许多情况下，数据需要脱敏。适当的测试数据管理还可以缩小测试环境的规模，降低复杂程度。

由预先存在的(比如其他服务与应用)组件及开发的新组件所构成的多组件复杂系统也带来了挑战。应用需要访问及交互的所有组件、服务还有系统可能无法用以运行测试。导致这种情形发生的原因有很多：组件、服务或系统可能还没有构建完成；可能已经构建，但只能作为生产系统使用，无法进行非生产数据测试；或者其使用会带来额外的成本。比如，对于第三方服务，成本就是一个主要的问题。

(8) 使任何人都能轻松获得最新的可执行文件。任何项目相关人员都应该拥有构建的访问权限并获得与之交互的途径。这样可以验证构建是否符合预期。

(9) 确保每个人都能看到发生的一切。这是一个沟通与协作相关的

最佳实践，而不是一个持续集成相关的实践。但其对于团队实践持续集成的重要性不容忽视。通过中央门户或仪表盘展示持续集成进度，为所有团队成员提供信息。

这样可以鼓舞士气，并有助于建立目标一致的团队合作意识。如果发生问题，可见性可以促进人们参与并帮助他人或其他团队克服挑战。通过公共团队门户实现可视性对异地工作的团队尤为重要，不过对于同地工作的团队还有负责不同组件的跨平台团队而言也是非常关键的。这种可视性应尽量将所有信息反馈给业务。正如前面所述，所交付应用与服务当前状态的可视性是业务的重要需求。

(10) 自动化部署。持续集成自然而然地导致了持续交付的概念产生与持续交付的实践应用——将软件自动化部署到测试、系统测试、类生产以及生产环境的过程。

1.3.2　持续交付

持续交付仅仅是把持续集成的概念引入下一个阶段。在每一次持续集成构建的最后，应用一旦完成构建，就会被交付至生命周期的下一个阶段。交付到测试(QA)团队进行测试，然后交付给运维团队进行生产环境发布。持续交付的目标是尽快获得开发人员为客户及用户创建的新性能。而今，并非从持续集成工作中产生的所有构建都需要交付给测试团队；只有那些具有功能特性的可以在开发阶段进行测试的“良好”构建才需要交付。

同样，并非所有经过测试的构建都需要交付到生产环境。只有那些在功能性、稳定性以及其他非功能需求方面都准备就绪的构建才会进行生产环境上线。为了测试是否已做好生产发布的准备，应当先将构建交付到类生产的部署或测试环境。定期将开发中的应用交付给测试及运维进行验证并潜在发布给客户的这种实践称为持续交付。

持续交付需要创建一条交付流水线(如图 1-3 所示)，其核心功能是实现持续交付自动化。随着持续集成产品以稳定的节奏构建，这些构建需要快速地交付至流水线中的其他环境。构建需要部署到测试环境中进行测试，部署到集成环境进行集成与测试等，最终部署到生产环

境。持续交付有助于将应用根据需要从一个环境部署到另一个环境。

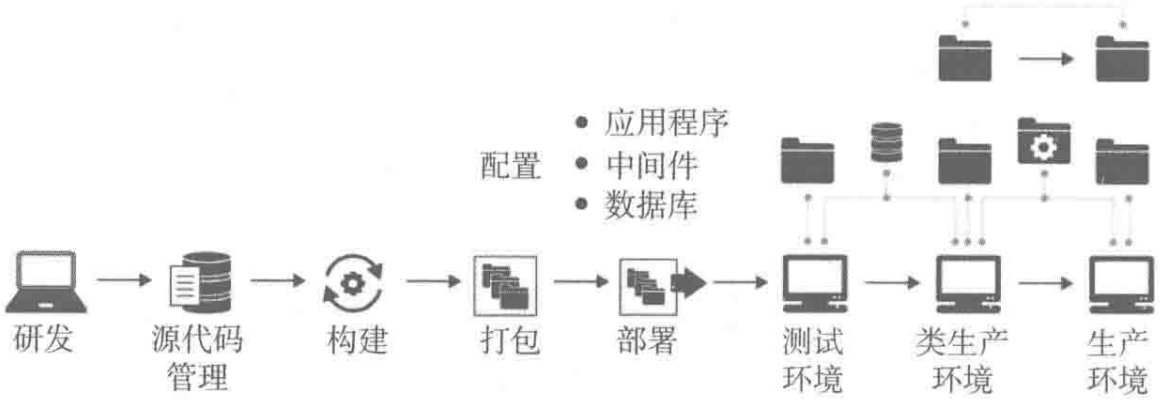


图 1-3　交付流水线

然而，持续交付不只是移动文件那么简单。需要编排代码部署、内容、应用、中间件以及环境配置，还需要变更流程，如图 1-4 所示。

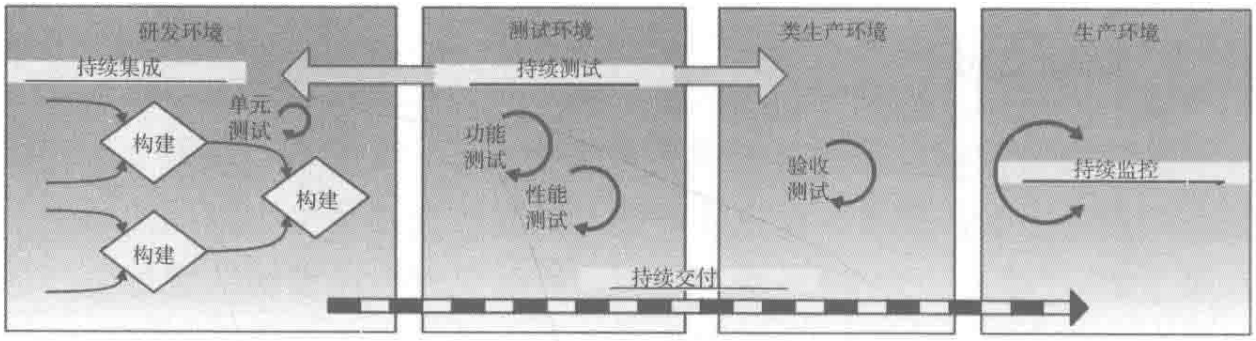


图 1-4　持续交付

关于持续交付，还要记住两个关键点：

- 并不是说每一次将变更部署到生产，都是通常所说的持续部署流程。相反，持续交付不是一个流程，而是随时根据需要部署到任何环境的能力。
- 持续交付并不总是意味着部署完整的应用。部署的可能是完整的应用、一个或多个应用组件、应用内容、应用或中间件配置变更，或是应用部署的目标环境。还可能是这些内容的任意组合。

十项持续集成实践中有两项构成了持续交付的链接，并且是持续交付所必需的：

- 在克隆生产环境中进行测试

- 自动化部署

在克隆生产环境(第七项实践)中进行测试时，或许也是一项测试实践，也需要持续交付能力，以将新的构建交付到克隆测试环境中。交付过程中可能需要配置测试环境、虚拟化的服务实例与应用。除了将应用真正部署到适当的测试环境之外，可能还需要配置相关的测试数据。

第十项持续集成实践，即自动化部署是持续交付的核心实践；没有自动化的部署流程，持续交付不可能实现。无论目标是部署完整的应用还是仅仅部署一个组件或配置变更，持续交付都需要准备好适当的工具与流程，以根据需要将应用或组件随时部署到交付流水线中的任意环境。

实践持续交付也是对实际的部署流程进行测试。将应用部署到生产环境时，组织遭遇严重问题的情况并不少见。然而，通过自动化部署流程以及生产上线前多次部署到类生产环境进行验证，是有可能在交付生命周期的早期就发现这些问题的。

持续交付与持续部署

过去，像 Flickr 这样的公司会在博客[⑤]上发布，截至目前，他们在一天或一周内一共部署了多少次。看到一个每周上线 89 次生产环境的组织可能会觉得很恐怖。更重要的是，回避了一个问题：“一周之内向生产环境部署 89 次，究竟都部署了什么内容？”

这是一个会让人想要远离 DevOps 实施的情景，因为他们认为必须将每一项变更都部署到生产环境。情况肯定不是这样的。首先，需要了解的是这里部署的内容是什么，其次(也更加重要)，必须要明白，这对于每一个组织而言都是不适用、不必要的，甚至是不可行的。

一周 89 次部署都部署了什么？当组织说他们每天进行两位数生产部署时，这并不意味着他们每天都交付几十个新功能或是修复几十个漏洞。这些公司实施的是真正的、全方位的持续部署。这意味着每个开发人员的每一个变化都会影响到生产环境。这些可能并不是完整的功能；

⑤ http://code.flickr.net

几天之内，由多个开发人员完成的同类变更有可能构成一个完整的可用功能。这部分功能可能是客户根本看不见的；只有在全部功能可用并经过测试之后，客户才能看见。然后，这也可能是 A/B 测试的一部分，所以只有少数客户能见到。部署也可能是从未有人见过的简单配置或数据库架构的变更，但会改变某些性能或功能表现。另一种情况是，部署涉及新的环境变化，而不涉及应用的变更，即操作系统(OS)或中间件补丁、操作系统或中间件配置的变更、新数据库架构的版本、全新的架构拓扑节点等。

这样的流程对许多组织来说是不可行的。有些组织可能有一些类似于敏捷瀑布的要求和政策，要求在部署到生产环境之前要通过手动审批流程。其他组织可能要求职责分离，要求部署到生产环境的人与开发可部署资产的人必须是不同的人或团队。

是否要进行持续部署？人们还是会混淆持续部署与持续交付的概念。

持续交付并不意味着每一次变化都要尽快部署到生产环境。而是意味着每一次变化都是随时可以部署的。

—— Carl Caum(Caum，2013)

这条 Carl Caum 的微博，简简单单的一句话却道出了问题的本质，组织应该做什么与能够做什么。依据这种标准区分，持续交付是必须的，而持续部署是一个可选项。具有持续部署的能力比实际持续地生产上线更为重要(这里的关键词是生产上线)。遗憾的是，这些术语仍然为大多数人混淆使用。

在交付生命周期中，需要的是可以部署到任何环境的测试与验证能力，最终到生产环境上线。或许只能持续部署到一个预生产环境(简配的环境)，例如，用户验收测试(User Acceptance Testing，UAT)，预生产……，但是部署的环境应该与生产环境类似，以确信真正生产上线时，最终的部署不会出现任何问题。

持续交付是对开发、测试环境以及其他(简配环境)非生产环境的每

一次变更进行交付。最终选择部署到生产环境的将会是一项完整的功能或一组功能，又或者是一项完整的应用或服务。

1.3.3　支持实践

除了 DevOps 的两项核心实践持续集成与持续交付(不应用这两项实践，就不是在实施 DevOps)以外，还有一些支持实践，用以支持及促进这两项核心实践。以下是其中的一些实践，这些实践是支持性的，却是必不可少的。

1. 基础设施即代码

运维世界的主人

想象一个经验丰富的运维工程师。在职业生涯中，他肯定已经开发出一个脚本工具包，利用这个工具包，只需要通过很小的改动，就可以完成他所有的日常工作，配置及管理他所见过的与处理过的大批环境。进行配置的时候，他清楚所有的管理控制台，就像自己的手背一般。为了解决所面对的问题，他可以登录并对应用服务器的配置进行精确的调整。对于数据库相关的问题，他确切地知道应该给谁打电话，而且知道 DBA 已经掌握其终端业务，就像他所熟练掌握的那样。他按例行程序处理事情。他确切知道下一次应用发布是什么时候。它知道什么时候操作系统会进行下一次更新。他是自己世界的主人。

随着系统的虚拟化以及开发人员启动持续集成(Continuous Integration，CI)实践，情况开始发生变化。运维工程师必须处理的环境及实例的数量增加了好几个数量级。开发人员不再每隔几个月才发布一次更新或新版本；他们每天都在推出持续集成构建，实际上是每天构建多个版本。所有这些构建都需要测试和验证。这就要求新的环境实例要快速周转。这些构建还经常伴随着配置变更。登录控制台逐一进行变更已不再可行。此外，对速度的需求也至关重要。开发人员的构建正在创建一个庞大的任务列表，因为连测试构建的环境在需要时都无法使用。

“休斯敦，我们有麻烦了。”

先来重新介绍两个概念：

1) 周期时间。周期时间指的是，从审批通过新需求、提出变更请求，或是识别出需要通过补丁进行修复的漏洞，直到交付至生产环境上线的平均时间。敏捷组织希望交付周期时间做到最短。这限制了他们向客户发布新特性与修复缺陷的能力。电商 Etsy 已经将周期时间降到分钟级的水平！虽然这对于企业应用来说是不可能的，但是当前数周有时甚至是数月的周期时间是绝对不能接受的。

2) 版本化环境。要根据要求维护开发当前所需的多个不同配置及补丁级别的环境，需要运维人员改变其处理变更和维护这些环境的方法。无论是打补丁还是变更配置，运维人员所进行的任何环境变更都应该视为创建了新版本的环境，而非仅仅是通过控制台调整了配置项。正确管理的唯一方法是通过脚本执行所有变更。这些脚本在运行时将创建所运行环境的新版本。在不影响运维的最佳实践 IT 基础设施库(Information Technology Infrastructure Library，ITIL)和 IT 服务管理(Information Technology Service Management，ITSM)的同时，这一流程使变更管理更加简单、高效，并可以实现规模化。

通过获取并管理基础设施即代码，可以同时满足最小化周期时间和环境版本化这两种需求。上线一个新虚拟环境或一个环境的新版本，就变成执行脚本的过程，脚本可以创建并提供一个或一组镜像，完成从操作系统到完整的应用堆栈的安装与配置。过去花费几小时的任务现在只需要几分钟即可完成。

正如在源代码管理(Source Code Management，SCM)系统中将代码版本化一样，对这些脚本进行版本化可以实现适当的配置管理。如今，创建新的环境版本要包括签出正确的脚本以及对脚本做出必要的修改，即给操作系统打补丁、更改应用服务器设置或者安装新版本的应用，然后在执行之前，将脚本作为一个新的环境版本重新检查。

注意：如果没有基础设施即代码，运维会很容易成为瀑布式敏捷的最后发布瓶颈。

市场上已经出现了一些用来获取及管理基础设施即代码的自动化架构。其中流行的有：Chef、Puppet、Salt 以及 Ansible。

随着云计算的发展，IT 即将完成软件定义环境(Software Defined Environment，SDE)的转变。这需要使用代码来定义、版本化及维护完整环境。像 OpenStack CloudFormation (亚马逊 Web 服务)这样的技术已成为行业领导者。例如 OpenStack，可以根据需要，利用 Heat 模板以软件方式定义全栈环境，使其可以利用 Chef 与 Salt 进行版本管理、创建及配置，还可以进行环境的扩容管理。运维人员不再专注于管理那些生命周期很长的独立服务器；他们现在管理的是大量的临时服务器，按需进行环境的提供和释放。只有软件定义环境才能实现这种规模化与敏捷性。

注意：在软件定义环境的世界里，服务器是“牲畜”，而非“宠物”(McCance，2012)和(Bias，2012)。

2. 持续反馈

回过头来从整体上认识一下持续反馈，其本质上意味着从交付流水线的每一个功能区域到其左侧区域获取反馈。因此，开发人员随着代码的开发与交付，向架构、业务分析及业务线提供相应的反馈；测试人员通过持续测试提供反馈给开发人员、架构、业务分析以及业务线；最后，运维人员提供反馈给测试人员、开发人员、架构、业务分析以及业务线，如果还有其他的利益相关者，将继续反馈下去。

持续反馈的目的是，对生成的代码进行验证，对与其他开发人员还有其他应用组件的代码所集成的代码进行验证，确保其功能与性能表现符合设计预期。一旦应用部署到生产系统，对应用进行监控以确保其功能及性能在生产环境中按照设计要求运行，如同终端用户使用时一样，这也是持续反馈的目标。持续反馈是实现持续改进与提高质量的必要条件，也是戴明 PDCA 循环的核心，因为其能够提供相应的信息输入，用以决定要改变的内容以及行动方案。

注意：没有持续反馈，持续集成与持续交付(几乎)没有任何意义。

不进行持续的测试与监控，就无法了解应用在生产环境中的运行状态，整个 DevOps 流程也就没有了实际意义。如果不满意用户开出的投诉单是发现应用功能或性能低于标准的唯一方式，那么拥有无缝的持续交付流程该有多好！

要实现持续反馈，需要两项 DevOps 实践：持续测试与持续监控。

持续测试 持续测试是在应用交付流水线的每个阶段实施应用、环境以及交付流程测试的能力。测试项目与测试类型可以根据交付生命周期的阶段不同而进行调整。如果正确地执行，持续测试实际上与持续集成及持续交付流程紧密相关。下面介绍其具体的运作模式。

开发人员独立编写代码。他们正在进行的其中一些工作任务(工作项)可能包括：修复缺陷、添加新特性、增强功能或提高代码运行速度。完成后，他们自己运行单元测试，然后交付代码并与团队中的其他人员开发的代码连同未做更改的代码进行集成(持续集成)。集成完成后，对集成代码进行单元测试。他们也可能运行其他的测试，比如白盒安全测试、代码性能测试等。之后，工作会交付到团队级别的公共集成环境，将项目与构成服务、应用或开发系统的所有代码组件的所有团队的工作进行集成。

这是持续集成流程的本质。为集成签入代码的时候，单个开发人员的代码会与团队的代码进行集成，这使得流程得以持续进行。这里要注意的重点是持续集成流程的目标：验证是否所有级别上的代码集成都准确无误，并且所有测试都是由开发人员准确运行。因此，确切地说，持续测试始于开发人员。

在确认完整的应用(服务或者系统)构建无误之后，应用交付到测试区域。将开发环境的代码交付到测试环境是持续交付的第一个重要步骤。当开发人员将代码交付到团队的集成空间以及项目的集成空间时，持续交付就开始了，但这仅限于开发空间内部，没有新的目标环境。

当交付到测试时，我指的是从一个环境到另一个环境的完整迁移。测试有自己的环境来运行功能与性能测试套件。DevOps 原则要求的是类生产环境。此外，测试每次运行测试套件时也可能需要新的数据集。

由于持续集成会导致持续稳定的交付构建，每天可能一次或多次出现这种需求。这意味着持续交付不仅要求从开发到测试的代码迁移流程，还需要刷新或提供新的测试所需的类生产环境，并且使用正确的配置及相关测试数据运行相应的测试。这使得持续交付流程更加复杂，而非仅仅是代码复制。需要特别注意的一点是，持续交付的目标是为测试及发布准备代码，并持续不断地将应用部署到合适的环境中，以便应用得以进行持续测试。

如果将此处描述的流程扩展到将服务、应用或者系统交付至部署环境及最终生产环境的过程中，那么其过程和目标也是相同的。在将应用交付到必须尽全力保障稳定的生产环境之前，运维团队希望运行自己的一系列冒烟测试、验收测试和系统稳定性测试。这个过程是利用部署环境完成的。如同测试环境一样，部署环境也是需要配置的一个类生产环境。运维人员运行验收测试与性能测试，也需要必要的脚本及测试数据。只有完成最终阶段的持续测试后，应用才会进行生产上线。因此，持续交付流程同样执行生产环境准备和应用交付的任务。

深入研究这一流程，发现持续测试是通过测试应用与环境的所有方面得以实现的，其中包括但不限于以下方面：

- 单元测试
- 功能测试
- 性能测试
- 集成测试
- 系统集成测试
- 安全测试
- 用户验收测试

在持续测试中，最大的挑战是运行某些测试所需的应用、服务以及数据源都是不可用的，或者即使可用，相关的使用成本也会使测试无法持续运行。此外，用以支持并行开发多个应用的所有团队的大型测试环境，其维护成本也非常高。

引入被称为“测试虚拟化”的实践(如图 1-5 所示)，可以解决这个问题。该实践利用虚拟化的桩代码取代了在测试期间必须与应用进行通

信与交互的实际应用、服务以及数据源。这些虚拟化的环境使应用的功能、集成以及性能测试得以实现，而不需要调用整个生态系统。这种虚拟化的方式可以用来执行此前所述的各种类型的测试。

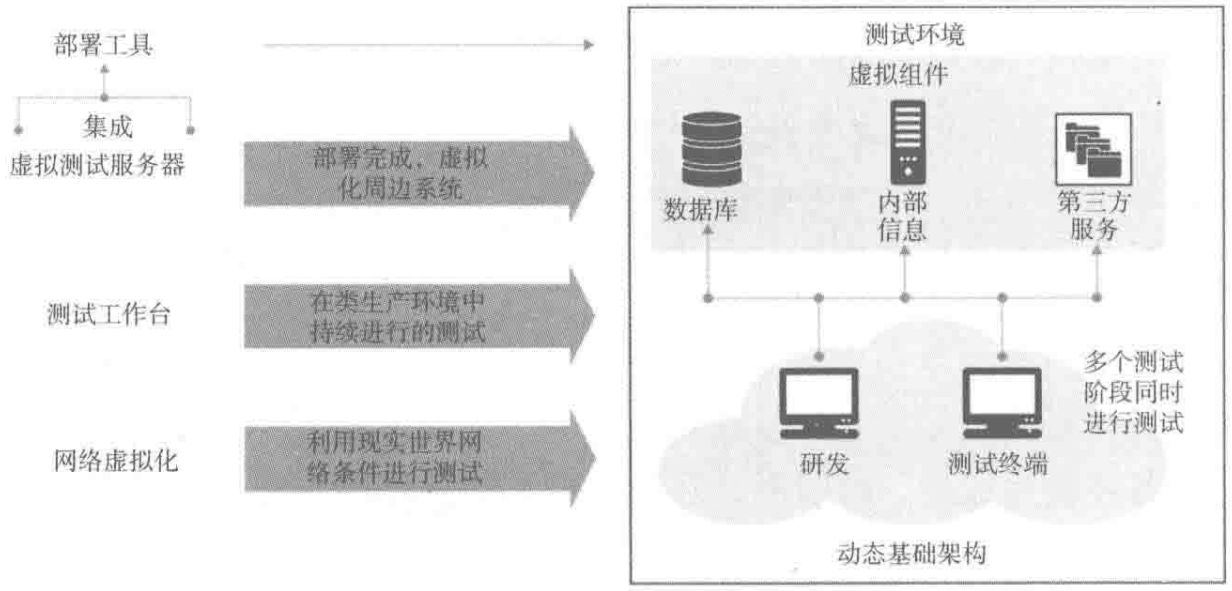


图 1-5　测试虚拟化案例

谈到 DevOp 情境下的测试，除了持续测试，还有测试前移实践(Shift Left，在价值流图中，流的前端在左侧)，本章后面 1.3.4 节“前移”会进行详细说明。

持续监控　在生产环境中，运维团队通过持续监控来管理并确保应用运行符合预期。运维团队有自己的工具来监控环境以及运行系统。最终，从流程级到系统监控工具所不允许的更低级别，运维团队需要保证应用都能正常运行。这就要求运维团队使用能够监控应用性能及异常的工具。可能还需要与开发人员合作，将自监控或分析收集功能植入到所构建的应用中。这就实现了真正的持续的端到端监控。

随着这一领域中技术的发展，出现了相应的工具与服务，可以用来监控应用的表现以及用户情绪，为开发人员及业务线提供有用的、详细的反馈。

简而言之，持续监控需要获取并分析以下四个方面的衡量指标：

- 应用性能
- 系统性能

- 应用用户行为
- 用户情绪

然而，运维团队不仅要收集这些数据，而且还要对其进行分析，这是非常必要的。此外，反馈必须要让其目标受众能够接受并应用，无论目标受众是技术性强的运维人员，比如性能工程师，还是非技术性的业务线利益相关者。数据能用才有价值。好的数据，数据分析就更具意义，可以实现真正的持续改进，因为交付流水线中从业务线到开发人员、测试人员的各级决策都是由数据驱动的。

反馈的前景是可预见的

随着 IBM 的沃森认知功能的出现，反馈数据预测性分析的领域展现出巨大的市场潜能。现在有关用户行为、应用行为以及系统行为的数据都可以进行分析，利用认知技术，从系统故障到用户行为(高兴或不满意)的各种预见性结果都能交付。预测分析可以使业务部门提前采取行动，防止产生宕机以及用户不满意。

3. 持续业务计划

持续业务计划的 DevOps 实践关注的是业务线及其计划流程。企业需要敏捷以及快速响应客户反馈。为了做到这一点，而今的许多企业都采用了精益思想技术。这些技术包括，通过识别测试业务愿景或价值所需的成果与资源来启动小规模业务，然后根据客户反馈不断进行调整。

为了实现这些目标，组织要衡量当前的基线状态，找到客户真正想要的，然后通过更新业务计划来相应地调整方向，使其能够在资源受限的环境中不断做出权衡决策。

利用精益创业运动所推广的，以及埃里克·莱斯(Eric Reis)在其著作《精益创业》中所描述的那些技术，领域内已经开展了很多工作。在想要开拓新市场以及尝试新业务模式，却不想为了向这些新领域提供复杂的 IT 系统而制定完整计划的企业中，像莱斯在其著作中所介绍的

提供最小可行产品这类的技术已经广泛流行。这个问题将在第 4 章进行详细讨论。

最近增加的功能库不仅可以确保交付成果的正确构建，而且还能确保构建正确的交付成果，这就是设计思维。与精益和敏捷一样，设计思维已经在工业产品设计的各个发展阶段应用了几十年。彼得·罗(Peter Rowe)1987 年出版的著作《设计思维》使其成为主流。设计思维最近才应用于 IT 领域，特别是专注于用户体验的应用设计方面。第 4 章将对设计思维进行详细讨论。

4. 协同开发

协同开发由 IBM 提出后逐渐成为主流，主要作为协同生命周期管理(Collaborative Lifecycle Management，CLM)工具套件所支持的一种实践。这种实践对于确保具有大型分布式团队的组织实现跨职能部门操作人员以及跨仓筒团队之间的协作与可见性是必不可少的。这可以通过确保跨交付流水线的两项功能来实现：

- 提供给操作人员的访问权限与可见性，不只是针对相关功能领域的构建物、工作项以及衡量指标，而是应该针对所有的功能领域(当然，访问权限是根据角色及安全需要进行管理的)。
- 构建物在操作人员或团队间进行无缝迁移。这有可能跨越功能边界，不需要对构建物进行任何编译或转换，以避免损耗。

这些功能只能通过交付流水线上的操作人员及团队使用集成工具套件来实现。

如果将 DevOps 视作一项文化运动，促进沟通、协作与信任是为之努力的核心原则，那么也可以将协同开发视为 DevOps 的核心能力。让操作人员之间使用统一的工具(不是电子邮件)进行沟通，是促进沟通、协作与信任的最佳途径。

这可以利用诸如 Slack 或 Rational Team Concert 这样的流行工具来实现。利用工作项间的工具协作，促进操作人员之间彼此转移工作项、添加注释、附加代码变更集，并实现对其他团队成员曾经从事或是正在进行的对本身工作产生影响的项目可视化，能够进一步促进协作。

谈到可见性，没有什么比可见性更能促进信任的。如果测试人员对开发人员正在进行的单元测试具有可见性，开发人员就知道如果没进行适当的单元测试，就不可以提交代码。

注意：完全的可见性驱动全面的信任。

在我们公司，将不再要求填写费用报销单。你可以任意消费，而我们都会报销。不问任何问题。唯一的要求就是把收据放到开放的维基页面，而公司的每一位员工都可以看到这个页面。相信我，你会明智地消费。

——硅谷初创公司 CEO

1.3.4　前移

前移也是一个源于精益的概念。这里的基本思想是将生命周期中影响质量的任务尽可能提前，以此来提高质量。这是贯穿整个生命周期的活动。潜在的前提是，越早发现质量问题，就能越早识别问题产生的根本原因并解决。

注意：测试领域有一个众所周知的原理：在需求阶段发现并修复一个缺陷或问题如果需要花费一美分，那么在开发阶段修复同样的问题则需要花费十美分；在测试阶段要一美元，在生产环境则需要十美元(Rice, 2009)。

当然，这些都是说明性的数字，并不是基于对实际成本的统计分析。然而，逻辑是正确的。前移任务能够尽早识别出缺陷与问题，可以节省成本并提高质量。

从 DevOps 文化角度来看，也可以将前移视作一种方法，通过让原本处于流水线后端的相关功能人员在生命周期的早期参与进来，促进协作与沟通。

DevOps 还是婚姻咨询？

我已经被架构师邀请去见他一个客户的开发总监及运维总监。我们共进午餐，架构师和我坐在桌子的一边，两位总监坐在另一边。我立刻就知道他们的情况不太妙。他们彼此分开坐着。开发总监抱怨运维人员有多不敏捷，而运维总监说开发人员交给他们的都是垃圾，把服务器弄崩溃了都无法运行。他们甚至在谈论对方的时候看着自己的手。我感觉我是在做婚姻咨询。

我推荐给他们的解决方案是从小处着手，将运维工作前移。他们的主要挑战是在部署到生产环境之前，开发团队和运维团队之间完全没有可见性。我的建议是选择一个关键项目，每周一次，由运维团队指派一名成员参加开发的每日例会，只需要听不需要参与，看看情况是否有所改善。不到三个月后，在一次会议上遇到了两个总监，我进行了跟踪。他们很高兴地说，运维人员现在出席每日例会，不只是听，而且还积极参与，分享他们的进展、计划和困难。运维管理已然前置。他们实现了协作。

为了最大限度地提升质量，有两个主要方面需要在交付流水线中进行前移。

1. 测试前移

测试人员从需求阶段就开始参与，可以使其更清楚需要测试的内容，相反，他们也可以确保所编写的需求都是可测试的。然而，我们的目标是在生命周期的早期开始测试。在工业领域越来越重视“前移测试”实践，所有的精力都用于确保在生命周期早期进行集成测试。虽然前移生命周期中其他形式的测试(如之前“持续测试”部分所述)也很重要，但是集成测试前移的意义最为重大。

当团队实践持续集成时，测试那些集成点以尽早识别集成及架构的缺陷，这对质量存在着重大影响。如果在集成时无法与其他服务及组件交互，那么拥有完美功能及性能的服务或组件又有什么用处呢？为了能够在生命周期的早期进行集成测试，测试虚拟化成为必要前提，因为完成测试需要的所有服务或组件届时可能都不可用。测试虚拟化利用虚拟

实例为无法获得的服务打桩，使集成测试以及其他测试得以在生命周期的早期进行，从而实现测试前移。要实现众所周知的“尽早测试、频繁测试”的目标就需要前移测试。

2. 运维前移的关注点

正如本章开头故事所描述的，运维团队通常被视作交付生命周期中的一个单独仓筒。他们通常在项目开始时加入，运维需求确定后就会离开，直到交付生产环境之前开始准备运维的时候，运维人员与开发人员之间都没有相互联系。运维人员在生命周期的早期加入并参与开发-测试周期，可以防止由于运维参与较晚而在部署到生产环境的过程中出现困难。运维人员尽早参与，会使他们清楚交付的内容及其可能导致的运维环境变化，因为需求有可能会偏离设计的状态。

运维人员尽早参与，也有助于确保开发-测试期间所部署的开发及测试类生产环境确实与真正的生产环境类似，而没有偏离。最后，运维人员的尽早参与还能确保开发团队所开发的部署流程及程序可以为运维团队所用。在 DevOps 出现之前，在发布周末进行生产部署时最大的挑战之一就是运维人员此前从未使用或测试过这些部署流程。代码部署到非生产环境时，要利用与运维团队相同的流程及程序，尽早并且频繁地对这些流程进行反复测试，确保其在生产环境中也可以正常运行。

前移的一个重要影响是使运维人员的角色发生变化。这些变化随着时间的推移悄然发生，涉及所需技能以及最终的流水线人员分配时却带来了意想不到的结果。

随着职责的前移，运维人员的角色从执行者转变为服务供应商。测试人员可能不再是进行测试的人，转而成为自动化测试供应商，开发人员通过自助服务完成测试。同样，对于运维人员来说，他们不再运行构建、配置以及服务器的释放，取而代之的是构建服务器镜像、管理服务器实例并对问题做出响应。开发人员、测试人员及其他运维人员，利用运维团队提供并管理的自助服务，根据需要提供、配置并释放服务器实例。这就提高了测试人员和运维人员工作的抽象性，进而影响所需的技能以及资源数量。

1.3.5 架构与降低风险

架构思维

在 20 世纪 90 年代中期加入 Rational 软件公司的时候，我开始关注架构思维。随着“UML 三友”，Grady Booch、James Rumbaugh 和 Ivar Jacobson 开发出 UML 理论以及 Philippe Kruchten 开发出 4+1 软件架构视图模型(Kruchten，2002)，架构思维渗透到我的血液里。

为了实现 DevOps 的全部承诺，架构终于开始在应用交付领域获得所需的关注。你无法通过大型单块架构系统实现持续交付。架构重组在 DevOps 早期曾被严重忽视，而今却成为主流，这主要归功于微服务的发展(或称之为 12 要素应用[⑥])。当关于微服务能否真正为每种应用交付价值的争论还在进行的时候，微服务所受到的关注已经再次引发对于架构的关注需求。如果真正了解 12 要素应用，会发现其对于 Web 应用以及软件即服务的关注是显而易见的。没有昂贵的代码及数据重构，它们不可能为大型的、复杂的、数据庞大的遗留应用及系统带来附加价值。只有当那些系统现代化到云原生应用时，这种投资才是可行的和必要的。微服务以及 12 要素应用将在第 5 章进行深入讨论。

无论是否使用微服务，实现持续交付所需的架构转换使变更得以按照小批次进行交付，从而缩小了批次规模。“批次”指的是在每个周期或迭代中所交付变更的数量。这些变更包括构成开发到运维完整周期的所有代码、配置、基础设施、数据、数据模式、脚本、部署流程等的变更(记住，并非每次都将所有的变更部署到生产环境)。缩小批次规模必须做到以下几点：

- 降低风险
- 提高质量
- 实现更快交付

这些好处是不言而喻的。在提高速度的同时，管理风险与质量最有

⑥ http://12factor.net。

效的方法是在每次迭代中缩小批次规模。这是一种思想转变，要转向更小规模、更频繁的新版本交付。批次规模缩小时，每一个周期中要进行的测试及验证就会更少；部署也会更少；而且，因为变化较少，风险也会随之降低。即使识别出挑战或问题，其影响也仅限于小批次内部，通过修复或回滚可以更容易地降低风险。

1.3.6　持续改进

最终，DevOps 的核心在于实现持续改进。不管你起点在哪里，处于什么成熟度水平，DevOps 实施都不是一次性的工作，而是需要持续不断的努力。正如彼得·圣吉(Peter Senge)在 20 世纪 90 年代所展望的那样，最终的目标是成为学习型组织(David A. GarvinAmy C. Edmondson, 2008)。在 DevOps 背景下，学习型组织是要不断地从刚刚交付的内容中学习并持续改进。改进的是什么？有以下三个方面可以改进：

- **应用。**刚刚交付的应用变更，其功能与性能是否符合预期？从持续反馈中学到了什么可以用于下一次迭代中的应用改进？
- **环境。**应用运行的环境，其性能及表现是否符合预期？是否满足服务水平协议(SLAs)的要求？从持续反馈中学到了什么可以用于下一次迭代中的环境改进？
- **进程。**从运维人员或利益相关者的经验中可以学到什么，能用于下一次迭代中的交付流程改进？

虽然大多数组织都致力于不断改进所交付的应用，但就真正的衡量指标而言，很少有组织会像持续改进运行环境那样严格。拥有持续改进交付流程相应方案的组织就更少了。尽管精益运动以及相应变异形式，如敏捷 Scrum 方法还有广泛采用的精益创业，已经具备成为学习型组织或团队所需要的基础条件，并且在流程层面上不断改进，但情况就是如此。

1.3.7　衡量标准

无法衡量，便无法管理。

——据说源自彼德·德鲁克(Peter Drucker)

无论彼得·德鲁克是否说过这句话，或是这种说法是否准确(Kaz，2013)，事实上，为了管理并持续改进某件事情，就必须能够测量一些关键的指标：关键绩效指标(Key Performance Indicator，KPI)。既要有这些关键绩效指标的衡量基线来标识初始状态，也要有持续的衡量标准以确认是否有所改进。不仅要从积极的方面评价进展，还需要了解背后原因及影响要素，即什么行为导致了关键绩效指标的改进。如果对人员与流程进行了一些改变，那么弄清楚究竟是哪种变化导致了关键绩效指标的改进，这点至关重要。

1.3.8 业务驱动

要知道哪些指标需要衡量与改进，就必须了解业务驱动。所要追求的是什么样的业务影响？组织若要投入 DevOps 实施变革，了解需要处理的业务驱动是前提。这有助于确定衡量指标，进而确定所要关注及投入的能力。仅仅关注速度意味着对世界的看法是非常短视的。

作为医疗器械制造商，对于我们而言，质量比速度更重要。我们宁愿推迟设备发布，也不要发生必须召回产品的情形。可以想象，召回已经安装的起搏器对于任何人来讲都不是一件好事。

——医疗器械制造商测试总监

什么样的关键绩效指标或衡量指标需要评价并努力改进呢？如前所述，这完全取决于业务驱动。业务线要求 IT 组织进行哪些改进？(即使在同一组织内部，不同的业务线也会有不同的要求)。是速度、质量、敏捷性、创新能力还是成本降低？是更高层次的要求，例如部署新的业务模式或是占领新市场的能力？还是更低层次的要求，例如减少平均故障间隔时间(Mean Time Between Failures，MTBF)、改善平均修复时间(Mean Time To Resolve，MTTR)，或是降低代码缺陷密度？能否利用 API 开发合作生态系统？是否缩短了 IT 交付新应用所需审批的处理时间？是否能通过参与开源项目吸引更多技术人才？(众所周知，向开源项目贡献代码的都是很厉害的公司)。

这是一套 DevOps 核心衡量指标子集，是 IBM 的一个部门用来衡量 DevOps 实施影响的。下述所列的这些衡量指标，均是根据这个团体需要施以影响(投放市场的速度、市场份额以及提高所交付产品的利润率)的业务驱动而确定的。

- 项目启动
- 清理积压
- 总体开发时间
- 综合构建时间
- 冒烟测试(Build Verification Test，BVT)可用性
- 迭代测试时间
- 总部署时间
- 整体上线时间
- 发布间隔时间
- 创新/维护占用时间比率(百分比)

1.4　DevOps：文化

“每个人都对生产交付负有责任。”这是 T 恤衫上印着的口号。我把它发给了所有人，甚至包括与项目关系不太密切的人。当然，分配到此项目的分析师、设计师、开发人员、测试人员、运维人员全部都有 T 恤衫。而企业架构团队、应用架构团队还有安全人员也都发了 T 恤衫。项目管理办公室(PMO)的人肯定要给。我给楼层管理员发了一件，如果厕所坏了，工程师就要浪费 20 分钟去另外一个楼层，楼层管理员现在要为线上发布的延迟负责。我给咖啡机维修工发了一件，如果咖啡机出了问题，我们就得派实习生去星巴克买咖啡，咖啡机维修人员现在就得为延期负责。我快递给我们的 CFO 一件。如果她不能在 12 月份管理好哪怕是其中一个外包商的预算与休假，就像去年那样，那么她就造成了生产上线延迟。CIO 得到一件，以便让我们的团队远离电子邮件泛滥。送给 CIO T 恤是想让其不要

> 拖延技术审批。糟糕，如果我妻子没能让我相信那是个坏主意，我应该会给出席公司野餐活动的每一位“重要人物”都发一件这样的 T 恤衫。DevOps 文化对于我就意味着朋友。
>
> —— 一家大型保险公司副总裁对 DevOps 文化的定义

如前所述，DevOps 核心是文化运动。那么，你如何改变文化呢？最终，即使组织完成所有的流程改进及自动化工作，也只有克服了组织内根深蒂固的文化惰性，才能够成功实施 DevOps 文化。组织具有惰性，本能地抗拒变化。改变并不容易，尤其是在大型组织中，这些组织的文化经过多年的沉淀积累并潜移默化地影响着成百上千的操作人员。这些人员，作为独立个体，他们或许认可 DevOps 实施的价值，但作为一个群体，他们就会抗拒变化，进而形成惰性。克服这种惰性是关键所在。文化惰性可以通过以下表达方式表现出来：

“我们这里就是这样做的。”

“是的，但是要改变 X 不是我所能控制的。”

“我们的流程没什么不好，为什么要改变？”

“你需要跟 Y 去谈，我们无法改变他人的工作方式。”

“管理层绝对不会允许的。”

“难道你不知道我们处在一个受管制的行业吗？”

“DevOps 是这个月的新花样。我们看看这项工作能持续多久。”

久而久之，组织发展出了行为规范；团队和群体按照组织划分职责分工；以管理之名设立检查平衡机制，纵然这并非真正意义上的管理；流程存在，却没人去思考其中的原因，只是视作“本来就有”；报告写出来已无人阅读，却没人愿意提出取消；以往发生的事故通过加强审批防止其再次发生等。所有这些行为形成了组织文化的惰性。

实施 DevOps 需要什么样的文化呢？信任、沟通和协作缺一不可。除非这种文化开始发展，否则单独实施 DevOps 实践无法培养出这样的文化，DevOps 实践也不会根深蒂固并固化为组织的 DNA。这是先有鸡还是先有蛋的问题，需要共同努力去克服文化惰性。文化惰性可以通过解决以下三个方面的问题来克服：

1. 可见性。本章前面用了很长的篇幅来讨论这个问题，其价值不容忽视。对必须共事的团队以及操作人员缺乏可见性，不能确定他们在将构建物移交过来之前都做过些什么，这是导致彼此缺乏信任的最重要原因。

2. 有效沟通。在 DevOps 环境下，电子邮件及语音邮件不可以作为沟通方式；项目计划、状态文档、幻灯片、电子表格也一样。沟通需要现场面对面地进行，而不是通过电子邮件、工作单或是通过管理手段进行。操作人员不需要通过一连串的命令，就能够根据需要与其他任何相关的操作人员进行沟通。这些实时沟通应该取代电子邮件、状态更新与协作，并且应该流化。Slack、HipChat、Yammer 和 Wrike 此类工具因此流行起来。

3. 通用衡量标准。除了此前所提及的，操作人员与团队缺少适当的衡量标准，是最容易导致惰性的原因。除非将新的、所期望的行为作为评价标准，否则人们的行为是不会发生改变的。此外，为了实现真正的协作，并形成每个独立团队都朝着跨越仓筒的整体目标而努力的意识，就要对所有操作人员采用相同的成功衡量标准。开发、测试及运维需要有相同的，或至少是相似的成功衡量标准。每一个人都必须对生产上线部署负责。

1.5　总结

DevOps 现在是主流。假设每个人都对 DevOps 的概念有着不同的理解，在更为重要的如何实施问题上也没有达成共识。非常遗憾，正确的答案是“视情况而定”。确实如此。这取决于为之奋斗的业务目标；取决于实践当前的成熟度；取决于组织能够承受的变化速度。为了实现业务增值，必须实施变革，但不能以牺牲成本为代价。任何中断都会导致生产力下降，实施 DevOps 也定会如此。

实施 DevOps 也是一段旅程，首先要识别出 A 点(当前的状态)与 B 点(业务目标)。一旦这些点确认完成，就可以绘制实施路线图来实施本章所介绍的正确实践与能力(合适的方案)。如何绘制这样的实施路线图？这是下一章的内容。

第2章

DevOps 实施

场地状况研究

板球运动是世界上最盛行的运动之一，拥有大约 25 亿球迷，仅次于足球运动(在美国，足球被称为“Soccer”，而在其他地区，则通常称足球为“football”)。板球运动从新西兰不断发展到圭亚那，球迷也随之遍布全世界。盛行这项运动的国家大都为前英国殖民地，而其中以印度为最。板球运动仍然在不断发展和普及，加拿大、苏格兰、阿富汗、阿拉伯联合酋长国已纷纷组建国家级板球队。

板球球场的中央区域称作“球道”。球道是精心修整的粘土带，甚至比高尔夫球场更为平滑。比赛主要在球道上进行。投球手使用各种技术试图改变板球在球道上的弹跳轨迹，从而使击球手出局。他们可以旋转手指、调整手腕动作，也可以利用板球上凸起的缝合线。基于球道的不同特点，投球手的努力会使板球的弹跳、速度、方向以及摆动产生不同的效果。最终目的是干扰击球手的判断，使其无法准确预测板球的运动轨迹实现得分，从而使击球手出局。

对于球员、管理团队以及专家评论员而言，研究板球球场的球道已然成为一种艺术形式。甚至在比赛开始之前，管理团队和球员都会花时间去研究球道的状况，去分析或预测土壤湿度、草坪状况、土壤脆度以及是否会有露水，因为所有这些因素都有可能改变夜间比赛时球道的状态。他们会分析哪位投球手的技术能力最适合当前的球道状况，并据此

决定最终的出场球员名单。

球队的表现在很大程度上取决于其对于球道状况的研究与评估，这是板球运动所独有的特征。除了篮球，几乎其他所有的团体运动项目都有标准的场地和地面，地理位置和比赛流程的变化不会导致明显的差异。但板球比赛的结果取决于队员对球道状况的了解与准备程度。

如何理解 DevOps 实施或应用呢？“DevOps 解决方案”又如何定义？DevOps 不是简单地采用某一个产品或者某一项流程。它是一个哲学体系，其中蕴含着影响到人、流程以及工具的一系列原则和实践。实施 DevOps 不仅仅是应用产品或流程，而是一场根本性变革。

此前，想要实施 DevOps 的组织提出的问题大多是：“什么是 DevOps？”“DevOps 是最新潮流吗？”“DevOps 会持续火下去吗？”而现在，他们会以这样的方式提出问题：

- DevOps 实施该如何开始？
- 现在我们已经在组织内部小范围成功实施 DevOps，该如何在整个企业范围内进行推广？

在任何情况下，实施的都不仅仅是 DevOps，而是一系列 DevOps 能力的构建(详见第 1 章)。DevOps 实施并非一劳永逸的项目。它实施的是一种思想、一种文化，是一个通过实施基于精益原则的一系列能力和实践实现持续改进的过程。这是一个长期的变革过程，需要一份定义明确且精心策划的实施手册进行指导，手册中应附有变革路线图。当然，随着实施项目的进展和其他变化的发生，指导手册的内容也需要不断更新。因此，要把指导手册视作一份动态文档，作为指导实施的关键资料妥善管理。这也就意味着，一份定义明确的指导手册对于变革成功至关重要。

那么问题来了，“我们该从哪里着手呢？”这就需要了解变革进程中的 B 点(实施目的以及希望通过实施 DevOps 实现的业务目标)和 A 点(当前的状态以及实践这些能力的成熟度)。清楚了 A、 B 两点，便可以撰写指导手册，其中包括一系列的实施原则与措施，这些内容是需要 IT 团队去遵循和实施的，以便实现预期的业务目标(或达到 B 点)。

指导手册的内容中需要涉及实施 DevOps 所需的四个核心领域的提升：

(1) 流程改进——如何使流程更加精益以及通过杜绝浪费提高效率。

(2) 工具自动化——如何利用工具实现流程的自动化改进，使其具有可重复性以及可扩展性，同时减少错误发生。

(3) 平台及环境——如何为从需求到生产上线全过程中各应用的交付流水线搭建更具弹性、灵活、可伸缩、可配置的平台及环境。

(4) 文化——尤其是如何塑造信任、沟通、协作的文化氛围。

进行这些方面的改变是组织变革的必经之路。不同组织的变革历程各不相同，事实上是因组织内部项目和团队的不同而异。组织实施变革的过程以及所需的时间受各种因素的影响，而其中最重要的因素是组织当前的成熟度等级。如果目前不具备良好的源代码管理实践能力，要想实现“零停机”部署，需要一个漫长的过程。

我不相信团队激励。我推崇的是让团队做好充分准备，这样上场的时候，他们便能拥有足够的信心打好比赛。

——汤姆·兰德利，美式橄榄球运动员及教练

2.1　撰写指导手册

和所有比赛项目(不论是不是运动项目)一样，能否成功实施 DevOps 变革最终还是取决于向目标冲刺过程中所采取的措施。同样，与所有运动项目一样，需要参考指导手册中的一系列行动指南。还需要根据不同情况选择及运用合适的实施办法，在这里，不同的情况指的是项目、部门或者组织的不同。

在为 DevOps 变革应用项目撰写指导手册时，对实施环境的理解至关重要。在篮球这类的运动项目指导手册中包括以下诸多方面的内容：

- 预备赛

- 边线发球
- 紧逼防守
- 区域联防
- 二次快攻
- 快攻后战术配合
- 比赛结束前的最后一投
- 最后一秒的比赛

依据位置、得分、比赛时间、场上球员(包括本队及对方球员)等各种情况的不同，教练与球员会采用不同的战略战术。同样，DevOps 实施指导手册中所涵盖的内容适用于团队、项目、程序、部门或者企业层面的各种情形。

要撰写这样一本 DevOps 实施指导手册，需要以下三方面的关键要素：

(1) 对“目标状态”(业务目标及驱动)的明确定义。

(2) 对“现状”(现时的能力及成熟度)的充分理解。

(3) 对最佳“路径”或最优“方案”(风险-价值-投资平衡)的选择决策。

对这些关键要素的分析与评估直接关系着能否正确把握项目实施的环境与条件。

2.1.1 识别目标状态(业务目标及驱动)

所有获胜的团队都是目标驱动的。这些团队持续获胜是因为每个相关的人都将注意力集中在特定的目标上。他们心无旁骛地朝目标前进，不受任何因素的干扰。

——卢·霍兹，美式橄榄球运动员、教练以及分析师

让我们稍微停留片刻，透过业务视角来探究一下 IT 世界。业务线(Lines Of Business，LOB)需要 IT 部门向他们的客户或用户提供相应的能力，以使客户能够获得商业价值。简而言之，IT 系统只是业务线向客户提供商业价值的工具而已。无论这种商业价值应用在哪个方面，呼叫

出租车、操作武器系统、支付税费，还是用来管理组织内员工的薪资计算，IT 只负责开发提供这些商业价值的系统。

业务线也可以通过这些系统获取相应的反馈，例如：客户和用户是如何使用业务能力并交付价值的；他们如何与业务系统交互；为了获取更大价值，他们需要什么增强性功能和特性。因此，IT 部门不但需要调整自己以配合业务线的目标，还需要不断进行调整以应对业务线的变革需求。作为一个行业，IT 在持续满足业务需求方面做得并不出色。这直接导致了“影子 IT”现象的出现，业务线跳过自己组织内的 IT 部门，转而寻求外部的 IT 供应商或服务商来满足其 IT 方面的业务需求。

所以，业务线到底需要 IT 部门提供哪些服务，又如何提供呢？事实上，确切的需求因组织、行业甚至地域不同而相去甚远。总体而言，业务线需要 IT 部门提供满足下列要素的系统及应用：

- 快速交付
- 敏捷
- 创新
- 优质
- 低成本

目前，这些要求或体现在业务线向 IT 部门提出的各种业务需求上，或体现为 IT 部门为了满足业务线需要对自身提出的技术要求上，例如：

- 价值交付时间
- 部署速度
- 成本节约/交付时间
- 成本节约/测试时间
- 提高测试覆盖率
- 提高环境利用率
- 最大限度地缩短部署所需停机时间
- 最大限度地减少部署所花费的时间(例如，连周末都占用的马拉松式部署)
- 尽可能地减少部署应用的回滚
- 提高再现与缺陷修复能力

- 最大限度地缩短产品问题的平均修复时间(MTTR)
- 缩短缺陷周期

不同的组织，甚至组织内不同的项目和流程都会提出各不相同的要求。

是的，桑吉夫，业务现在要求我们做的五件事情就是降低成本、降低成本、降低成本、降低成本，没错，就是降低成本。

——欧洲知名金融服务公司首席信息官

IT 部门，特别是大型组织内部的 IT 部门，并非为了交付速度、敏捷与创新而设计。相反，他们历时数年，主要通过改变管理来实现交付服务的稳定性、高品质以及确定性。这就导致了变更管理流程、实践以及工具的发展，形成了专制的管理与严苛的体制，产生了过时的仓筒组织结构以及惰性的文化。另一个结果则是以月度或季度为周期，进行大规模交付。然而，正如我将要在第 3 章进行的讨论所述，无论管理有多么完善，要在短时间内开展大量的工作，通常会更加复杂，更加容易产生相互依赖，生产过程中发现问题也更加难以回滚。在现代化的架构与流程开发出来之前所设计的遗留系统与交付流程敏捷性不足，这加剧了问题的严重性，但是仍然需要 IT 部门去支持。

在某些情况下，IT 部门在交付敏捷性、速度以及创新性方面能力的缺失会直接导致业务线创建自己的“影子 IT”部门来提交他们的 IT 需求。“影子 IT”是一个专业术语，指的是业务线在企业的 IT 部门之外创建属于自己的 IT 团队与架构，或者利用诸如第三方开发团队以及公有云服务这类的非企业资源来开发及交付他们的 IT 需求。“影子 IT”带来的挑战是他们在暗处，不受组织 IT 部门的管理和监督。而且，“影子 IT”团队通常不了解组织内部的业务知识。也就是说，通常情况下，他们不像组织内部人员那般了解业务，他们经常不在状态，他们也不会参与或影响企业的战略方向。因此，在进行高附加值决策时，他们不如组织内部的 IT 团队值得信赖。

当然，这个问题的解决方案就是实施 DevOps 变革，以使 IT 部门能

够实施必要的实践、运用自动化工具、改变组织与文化，以提供兼具速度与敏捷性的应用及服务，并通过快速实验能力实现创新。

DevOps 的目标之一是通过优化应用交付流水线来实现效率最大化，而这种优化涉及敏捷性、速度、质量以及成本控制各个方面。如前所述，要实现这样的目标，需要精益高效的流程、集成应用开发及交付工具、方便快捷的环境管理，以及全功能团队间互信与沟通的文化氛围。随着效率的最大化进程，大多数组织都需要推动创新。创新的需求又会相应地驱动组织去采用为实现快速创新而设计的新技术平台与流程，例如云化的计算平台 Platform as a Service (平台即服务，PaaS)，其本身就能提供所有必要的 DevOps 服务。创新的需求还会使 IT 部门意识到他们也必须优化相应的遗留系统应用交付能力。他们需要这样做以释放创新所需投入的人力和财力，他们必须这样做才能确保传统的低效率交付不会影响其快速创新的能力。

此时提及行业中应用的模型就显得尤为重要了，其被用来平衡不同速度下的“创新”与“优化”。我们可以将其视作体育项目中平衡进攻与防守的规则。这个通常被称为“多级 IT”的模型将在第 3 章后面的部分进行讨论。

2.1.2　评估现状

注意：我经常被问及 DevOps 变革项目的投资回报周期，这就如同在问我从华盛顿哥伦比亚特区到德克萨斯州的奥斯汀所需要的时间一样，飞机的飞行时间是三小时零三分钟，然而实际所花费的时间远不止于此，因为我不住在机场。旅程究竟需要多少时间取决于起点的位置，DevOps 变革项目亦是如此。

评估组织或团队的现状或者成熟度有很多种方法。运动队的教练和队长需要深入了解团队的各项能力，其中包括：自身的优势与不足；每个队员的技术水平、当前状态和健康状况；对手的技术水平和现状以及比赛的实时状况。这些数据能够帮助教练员和队长选择正确的战术来应对接下来的比赛。同样，在 IT 世界中，对团队、项目或者组织当前成

熟度以及状态的了解能够帮助管理者在指导手册中正确选择行之有效的实施方案。

要启动当前成熟度的评估，可以先从以下几项涉及组织能力的核心问题着手：

(1) 是否具备快速交付全新的创新型应用以及利用现代化架构的能力？

(2) 是否具备更新现有应用以适应更快交付与创新的能力？

(3) 是否具备对有助于成功的那些文化、工具以及流程的适应能力？

大多数组织会在至少其中一个方面遇到挑战，很多组织甚至三个方面都有问题。妥善应对这三方面的挑战需要以下三个步骤：

(1) 识别低效或浪费。

(2) 识别导致低效或浪费的根本原因。

(3) 制定计划或实施方案来解决这些根本问题。

价值流图是一种精益管理办法，针对如何将产品或服务交付给客户全过程中的一系列事件，分析其当前状态并设计其未来状态。

——维基百科

---

1. 利用价值流图识别低效或浪费

通过审视组织所交付的各类不同应用的交付流水线以及查找浪费发生之处，可以识别低效或浪费。我自己和其他人做的一些非正式调查以及坊间证据都显示，很多组织中 40%的资源都被浪费掉了。因为这并非科学调研，所以 40%这个数字的误差可能比较大，然而并没有人提出相反的论据。玛丽·帕彭迪克把浪费定义为："消耗资源却不能提升客户价值的所有事情"(Wagner，2009)。浪费来自于很多方面，主要是源于流程以及对于从业人员的管理费用，还有流程中的多余步骤，尽管这一切的出发点都是好的。举例来说，浪费包括以下方面：

- 不必要的流程步骤
- 不必要的返工

- 不必要的功能
- 创建错误的构建物或流程
- 转换他人创建的构建物结构
- 等待他人采取行动或执行任务
- 等待审批
- 等待平台环境就绪
- 为手动任务创建任务单
- 制造或升级无法提升终端客户价值的构建物
- 手动更新数据或报表向上级汇报
- 状态报告

大多数浪费发生在交接过程中，特别是跨职能部门的利益相关者进行交接的期间，还有就是操作人员等待他人做出行动的时候。当构建物发出去，接收方却无法直接使用时，也会产生相应的浪费。也就是说，构建物需要接收方进行修正或改良才能使用或者需要退回返工。精益理论用完成率以及准确率(%Complete and Accurate，%C&A)作为衡量的标准(Martin，2011)。

识别浪费和低效的最有效工具就是众所周知的价值流图。价值流图并非全新的概念，其在精益管理领域已运用多年。作为识别软件开发过程中所产生的浪费的一种工具，汤姆和玛丽·帕彭迪克将其在软件行业进行了推广(Poppendieck，2008)。

在 DevOps 环境下，绘制价值流图需要同时审视交付流水线一端进来的需求流以及另一端在生产环境下运行的应用程序。这些需求涵盖以下方面的内容：

- 新的需求
- 需求改进
- 需求变更
- 故障修复
- 政策变化
- 配置更改
- 内容更新

这基本上囊括了能够改变生产环境下运行应用程序的所有要素。

当一个需求产生，要求进行改变或提交新事物的时候，随着构建物在利益相关者之间的转移，利益相关者会执行进程，对构建物进行创建、修改或转换，并将其从一个环境迁移至另一个环境。作为构建物的需求在交付流水线的一端进入，生产环境中运行的代码在交付流水线的另一端生成最终构建物。

DevOps 实施的终极目标是减少浪费(或者用精益理论的说法，称之为“去除无附加值工作”)、提升流程效率(也可以称为“创建更高效的流程”)并持续进行，为实现持续改进(或者按照日本的习惯称为“kaizen”，即“持续改进”)而不懈努力。要想实现这个目标，需要审视以下所有事项：

- 构建物
- 利益相关者
- 环境
- 流程

目标是识别出导致浪费的情况(详见此前所述的样例)，然后进行分析并最终确定导致浪费的根本原因。以下是可能产生浪费的一些情况：

- 流程低效
- 流程手动
- 构建物形式错误
- 利益相关者之间的交接低效
- 利益相关者无法在流程中执行任务
- 利益相关者无法根据需要访问构建物
- 利益相关者在不必要的任务上耗费时间
- 利益相关者创建不必要的构建物
- 流程被过度监管

关于根本原因分析，会在本章后面的内容中进行论述。

运用价值流图

识别交付流水线中所产生浪费的最有效工具是价值流图(Value Stream

Mapping，VSM)。简单的价值流图及相关内容请见图 2-1，现在探究一下如何利用价值流图识别浪费。

可以从两个方面将流程可视化：

(1) **以活动为中心：**流程被定义为一系列的操作活动。而这些有序的操作活动能够使流程产出工作成果。路线图和系统图(IDEF 方法)有效地说明了此过程。

(2) **以构建物为中心：**工作成果及其生命周期对流程进行了解释。工作成果被视为显示状态转换的状态机。用状态转换过程中的工作成果定义每一个流程步骤(Cantor，2014)

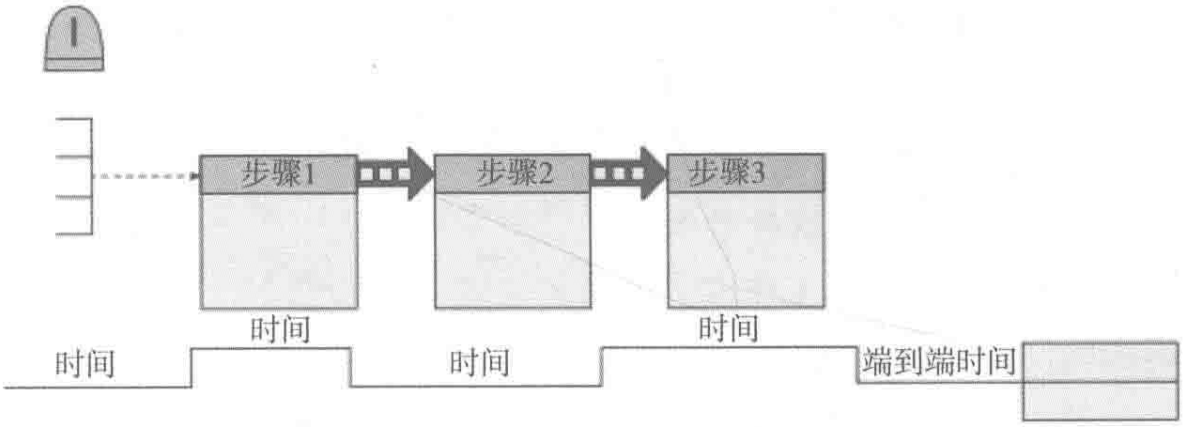


图 2-1　简易价值流图

当运用价值流图的时候，需要同时审视活动与构建物两个方面。选择一种构建物并对其在交付流水线中的状态进行跟踪，利益相关者会在此期间对构建物进行结构转换(状态转换)，创造更多的构建物并且对现有的构建物进行修改。例如，一个单一的改进需求会衍生出多行代码，也会修改其他代码。随着构建物在利益相关者之间流动、修改以及环境的变化，流程内分析是必须要做的，以确定是否存在瓶颈或者浪费源。图 2-2 就是基于这样的过程所创建的一幅价值流图。

基于交付流水线这一理念，跟踪构建物在交付过程中的状态，价值流图因此应运而生。价值流图能够辨识价值流中所有可识别的瓶颈问题。

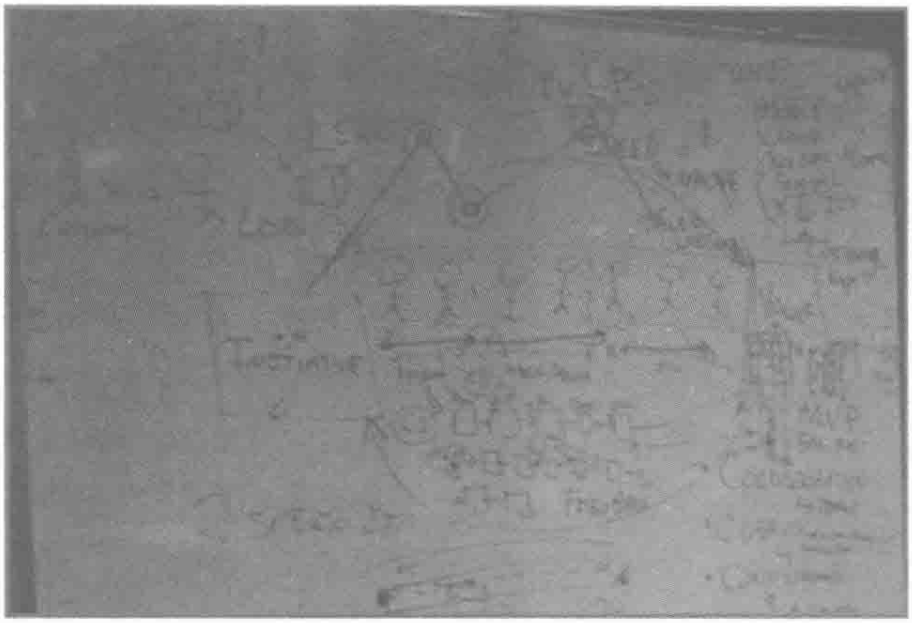


图 2-2　价值流图交付流水线

价值流图中包含两类评价指标：

(1) 流程时间：实际作业时间。

(2) 前置时间：从工作准备就绪到完成并交接给下一个利益相关者所需的时间(Martin，2011)。

另外，针对利益相关者之间交接时工作质量的评价，采用的也是我们本章前面部分介绍过的完成率与准确率(C&A)作为衡量指标。

价值流图的运用，不管做得很深入具体(辨识具体的作业时间和等待时间，形成流程以及流程内衡量标准)还是相对全面概括(更倾向于识别瓶颈问题而非具体指标)，之后都需要进行根本原因分析(Root-Cause Analysis，RCA)以确定每项识别出的瓶颈问题所产生的根本原因。反过来，这些根本原因就是我们要应对和解决的问题，也是 DevOps 实施路线图的核心内容。

如前所述，我们可以按通用的两种方式来使用价值流图(事实上价值流图的使用方式多种多样，鉴于领域和重点不同，每个咨询公司以及每种方法论都有各自不同的方式)：

(1) 深入具体——这种方式需要多个顾问人员耗时多日，甚至多个星期与操作人员在一起以衡量每一项任务的作业时间与等候时间。基于这些衡量数据，绘制出详细的价值流图。这份详细的路线图能够辨识出

每一种操作模式下产生浪费的确切原因。在劳动密集型企业，这种价值流图体现为生产标准化。

(2) 全面概括——这种方式一般需要利用半天至一天的时间与能够代表交付流水线中核心功能领域的高层领导进行专题研讨，而不是与操作人员一起。在研讨的基础上，为典型的或者新产生的变革需求绘制典型的交付流水线，目标是识别出交付流水线中的主要瓶颈问题。之后需要对这些瓶颈问题进行分析以确定浪费产生的根本原因。这种路线图在 IT 领域应用比较广泛，在本章后面的部分会进行详细论述。

交付流水线与工厂装配生产线

DevOps 持续交付常被比喻为工厂的装配生产线。更通俗点讲，就好比软件供应链与生产供应链。尽管由于诸多原因，这种比喻在 IT 行业广泛流传，甚至本书也采用了这个比喻，但是其还是具有一定的局限性。在开发价值流图的过程中这种局限性尤为明显。现在我们就来具体分析一下两者的异同。

供应链是指产品或服务从供应商到客户流通过程中所涉及的组织、人员、技术、行为、信息以及资源共同组成的一个系统。

——维基百科

这个生产供应链中常用的定义实际上也适用于软件供应链，纵然两者有些明显的区别。

生产供应链

生产供应链将自然资源、原材料以及零部件转换为交付给终端客户的成品。在这样的供应链中，生产制造商的角色更像是一个资源整合者。不论产品是简单如少许组件即可生产的洋娃娃(塑料身体、假发、衣服、硬纸板以及塑料盒)还是复杂如包含 20 000 多个软硬件的汽车，针对每一个构成终端产品的组件，生产制造商都需要思考这样一个问题：其他公司能否保质保量地以更快速度、更低价格生产这种组件？如果答案是肯定的，那么生产制造商的角色就会转换为采购商，而这些生产组件的公司就成为供应商。当然，这些供应商可以是内部供应商，可以是其他

的团队，也可以是提供组件公司内部的其中一个部门。

举个典型的例子，例如汽车的制动器。制动器是每辆汽车的必备组件，生产制造商还是要去评估是否其他厂商能够提供更加物美价廉的产品。如果答案是肯定的，他们就应从供应商那里进行采购。还有一种需要外部采购组件的情形是，生产制造商不具备自行生产这种产品的专业技术和能力，例如汽车内的牵引控制系统或防抱死制动系统。

还有一种情形是，如果供应商拥有某种组件的知识产权，那么生产制造商就必须从供应商那里进行采购，例如电动汽车使用的高容量电池。在这种模式下，现代的汽车制造商基本只负责汽车的设计与装配，大部分的零部件都从外部供应商采购。事实上，除了成品所需的物理组件之外，生产制造商还会外购一些设计或是构型作品。例如，飞机制造商会将机翼边缘的设计外包给专业从事机翼边缘设计工作的精品航空工程公司。

采购商与供应商之间渐渐转变成沟通与协作的关系。采购商提供设计规格和质量要求，供应商通常按时交付保质保量的组件产品。对于采购商而言，其成本收益来源于不需要自行运营生产工厂(包括设施、设备以及人员)来制造本可以由他人提供的组件。对于供应商而言，其成本收益则来源于规模化生产所带来的成本降低。

软件供应链

软件供应链将供应链的概念延伸至软件与系统的交付，但基本的商业逻辑以及供应链模式的应用价值与生产供应链并无差别。对于软件制造商而言，创建软件工厂时对其软件供应链中的部分组件进行外购是具有相当意义的。这些软件的供应商，其制造成本和人工费用更低，相关经验丰富，拥有采购商所不具备的专业技术(例如移动开发技术)。这里有必要重申的是，供应商可以是公司内部其他团队或部门中的软件开发团队。

注意：现在要说的是二者的区别。工厂和装配生产线制造的是相同的部件及成品，但是软件交付做不到这点。软件中的组件各不相同，每一部分都是独一无二的。开发人员的任务就是编写全新且唯一的代码。

这也同样适用于价值流图。尽管能够衡量具体等待和作业时间的详细版价值流图在装配生产线和生产供应链系统中运行良好，但对于应用的交付却发挥不了同样作用。举例来说，开发人员花费 X 小时的作业时间去开发一个组件，但是开发下一个组件的作业时间并不一定也是 X 小时，等待时间也是同样的道理。开发人员编写出代码，需要等待测试人员完成测试，而这个时间又会因为需要测试的新代码数量不同而有所差异。因此，交付流水线最好聚焦于发现主要瓶颈问题，而非具体细节。全面概括性的价值流图发挥的正是这样的作用，而且建议从 DevOps 开始着手。

现在我们来分析一些具体的差异：

- 需求——软件规格(即需求)很难像物理组件那样进行明确的定义。试着比较一下，软件工程中对用户界面(User Interface，UI)的一系列需求与汽车仪表盘所要求的具体工程及设计规格之间的不同。
- 需求稳定性——软件需求通常不够稳定，甚至连采购商本身都无法准确把握。随着对应用的进一步了解或者随着系统的建立，需求也会相应发生变化(这是行业内最初推行敏捷软件开发实践的主要原因)。
- 变更——软件产品通常比实物产品变更频繁。同是 2012 款的丰田车，一月份生产的与五月份生产的并无差别。与此同时，大多数的手机应用却几乎每过几周就会更新。
- 成本——软件组件的生产成本不会因规模化生产而有所下降。如果一年能够销售逾百万辆汽车，比如说一级方程式赛车，每辆车需要四套制动装置，那么制动装置的生产成本就会远低于单独一套定制化制动装置的成本。但是软件开发过程中，几乎每一次所编写的代码都是定制化的。
- 集成——在生产制造过程中，组件间的接口是明确的，是标准化的(比如，固定尺寸的螺栓及螺母)，即使是非标准化的，也能在允许的误差范围内向供应商提供相对具体的规格说明。软件接口的情况则完全不同。事实上，在各组件之间的集成点，采购商与供应商之间的职责区分已非常模糊。加之多家组件供应

商介入的复杂性及其各不相同的集成点，整个形势变得非常有趣(思考，healthcare.gov.网站第一版) 。

- 评估——一旦建成工厂进行批量生产，大量生产零部件所需的原材料以及耗时基本可以准确估算。但是要估算软件开发的工作量是非常棘手的事情。除非聘用同一个团队，而这个团队又对这类组件的生产具有相当丰富的经验，否则很难去准确判断开发一个组件所需要的时间。需求以及接口规格的变化进一步增加了评估的复杂程度。敏捷这类实践的确在某种程度上提升了评估的准确性，但这是基于 “计划扑克”(Hartman，2009)的游戏规则。
- 测试——质量控制是软件开发区别于生产制造的另一个方面。汽车生产商收到组件时，很容易测试出这些组件是否符合要求以及误差是否在合理范围内。他们只需要抽检部分样品即可确认整批产品的质量。软件的每个组件都需要通过所有用例的测试。质量要求的程度以及测试的费用成本决定了测试的严谨性。

注意：比起文字处理器，除颤器软件需要更严格的检测(尽管足够多的文字处理器崩溃还是要用到除颤器)。

- 标准实践——生产实践通常都执行标准的作业流程，拥有诸如 ISO 这类特定资质的供应商之间不会有明显差异。而软件开发实践却很难明确定义或者标准化。即使有详实的文档，软件开发实践也很难实现和严格遵循。过去几十年间瀑布式项目的惨淡成功率充分证明了这一点。敏捷实践仍在不断发展其所定义的敏捷，但是同样拥有 Scrum 认证的两家供应商在实践 Scrum 时却差别显著，结果也大相径庭。
- 增量构建——硬件组件的制造不是增量式的，尽管组件的设计甚至生产实践会不断进行改进(尤其是施行“精益管理”或是“持续改进”实践的组织)，但每个组件都是全新制造的。2017 款汽车上的制动器或许性能优于 2016 款，但是其没有用到旧款制动器中的任何一个零部件，而都是全新制造的。与之相反，软件

组件的制造则是增量式的。无论是在当前版本还是在新版本中，组件的每一次迭代都伴随着新代码的增加或者原代码的修订。随着再次利用以及开源库的发展，即便是全新的组件，现在也是利用其他组件或是开源库中的代码进行构建。

- 合约——生产制造组件采购协议均基于数量、时间以及质量服务水平协议(Service Level Agreement，SLA)。基于已知的制造及原料成本，协议通常很容易创建及执行。然而，软件组件采购协议却非常复杂。合约金额固定会带来相应问题，因为对时间和投入的错误判断会导致延期，或者供应商为了满足时间要求而降低产品质量。涉及时间和投入的合同需要复杂的服务水平协议以及严格的监督，这使得合同的标的额很难确定及执行。

综上所述，软件与系统开发，尤其是大规模的、复杂的软件与系统的开发同工厂中的供应链看起来很相似(从较低层次的抽象概念来理解，应用交付流水线类似于工厂的装配生产线)，但是仔细分析却发现二者截然不同。

组织价值流图工作坊

价值流图工作坊最好由具有决策能力的高管来组织。而且要取得成功，至少需要以下利益相关者的参与：

- 副总裁或是业务线的主管领导
- 副总裁或是应用开发的主管领导
- 副总裁或是测试(QA)的主管领导
- 副总裁或是运维的主管领导

此外，其他一些核心人员，比如下面这些部门的主管领导，虽然不一定必须参加，但是他们的加入会进一步提升工作坊的价值：

- 安全
- 企业架构
- 项目管理办公室(Program Management Office，PMO)
- 产品/采购管理(如有)

工作坊最好在企业、部门、流程或者业务线层面而非项目层面组织

召开，因为项目内容太琐碎，很难进行价值评估。工作坊通常历时 2～4 个小时。所有管理人员全程参与至关重要，目标就是让管理人员了解已识别的瓶颈问题及相应的解决计划。

我从未在同一个房间收到过如此多的直接报告。相比于专注各自的领域，作为一个团队来应对共同的挑战反而更好。

——知名金融管理公司的首席信息官参加工作坊后

工作坊需要一个组织者推动和指导参与的管理人员去采取一系列行动。以下行动内容供参考：

(1) 识别客户的业务目标，而这些目标能够驱使其引进或考虑实施 DevOps；

(2) 识别为了实施 DevOps 已经执行或是计划执行的 IT 举措。这些举措又会反过来调整业务目标的定位；

(3) 创建业务流程图，识别管理人员所发现的那些阻碍业务目标实现的瓶颈问题及低效率事件；

(4) 确定有助于解决所识别瓶颈问题的 3-4 种能力的优先级列表；

(5) 绘制最佳实践的实施路线图，要包含关键事件、时间期限(日期)以及识别出的挑战。

当然，工作坊的核心任务就是创建价值流图，这也正是工作坊中耗时最多的部分。

创建价值流图的最佳途径是通过交付流水线获取一个构建物，通常是一种新的需求。如本章前面部分所阐述，正如“构建物”(新需求)利益相关者实施相应流程去改变其状态，然后移交给其他利益相关者。如此，每一个利益相关者，在每一次切换过程中，都需要识别出浪费的原因所在。浪费主要有以下两种方式，这也正是交付流水线的瓶颈问题：

(1) 等待时间——利益相关者等待行动或是等待其他人员进行构建物移交。

(2) 过度生产或是错误生产——利益相关者对构建物进行不必要的更改或者没能提升终端交付的价值。

交付流水线运用价值流图可识别 10～15 项关键瓶颈问题。接下来的关键任务是进行根本原因分析以确定每个瓶颈问题产生的根本原因，这些原因正是展现给利益相关者的典型问题。关于根本原因分析，将在“诊断根本原因”部分进行详细论述。

一旦识别出每个瓶颈问题的根本原因，便可以探寻他们彼此之间的依赖关系并最终确定优先级。在确定优先级之前，需要首先进行依赖性分析，以消除重复项并且降低那些在解决好其他相关问题之后才能解决的瓶颈问题的优先等级。另外，优先级的确定还需要考虑业务影响、投资回报以及所需要付出的努力程度。建立一种在高业务影响力项目与易实现目标(容易实现，并且立即产生影响，显现价值)之间的平衡策略是可行的。成功案例以及快速响应显然要在业务中体现。之后这份优先级清单需要纳入确定的时间计划表以及组织的预算范围内。

瓶颈问题的优先级列表需要与最初确定好的业务目标相对应。如果一个瓶颈问题对业务线提出的关键目标不产生直接影响，解决它就没有意义。此步骤会进一步提炼出需要解决的瓶颈问题的优先级，并最终形成前 5-6 个问题的优先级列表。然后利用这份列表来确定需要采取的 DevOps 实施方案，以创建 DevOps 实施路线图。

2. 诊断根本原因

> 冠军队注重的是基础战术——挡拆、团队合作以及防守。他们激情澎湃并且全力以赴。球在手里的时候，他们运球；球不在手里的时候，他们继续跑动。他们敏锐地利用错位。他们评估对手的弱点并攻其不备。
>
> ——沃尔特·弗雷泽，前 NBA 球员

如前所述，需要对识别出的每一项瓶颈问题的根本原因进行审视，以确定哪些办法可以纳入指导手册。之所以要这样做，是因为通过价值流图辨识出的瓶颈问题实际上所反映出的就是导致浪费或低效的根本原因。要识别出这些根本原因，就需要进行根本原因分析。实施根本原因分析的典型办法就是问五次“为什么”(Ohno，2006)。这种办法可以让我们透过表面的现象看到事情的本质或是根本原因。

让我们探讨一些常见的瓶颈问题并分析一下可能导致问题产生的潜在根本原因。下面是组织或项目中出现的一系列无效及浪费现象：

- 操作人员交接过程中，缺少工具集成导致等待。
- 团队成员对其他人的工作缺少可视化及同步性。
- 跨越团队边界的学习与贡献难以进行。

以上这些是缺少集成交付流水线的典型现象。

如果缺少标准化环境，则会出现以下典型的现象：

- 与生产环境不相似的多个不连通的开发测试环境。
- 独立维护和管理的多个技术栈。
- 欠缺根据风险、价值以及技术匹配度引入最佳平台的能力。

下面是多级应用交付的管理挑战所引发的一系列低效现象：

- 业务线所需要的发布业务功能的能力欠缺，而非单独应用。
- 最低速交付流水线阻碍最高速交付流水线的速度。
- 几乎所有的 IT 资源都用来运行、管理及维护现有的应用程序及工作负载，很少有资源可以用于创新。

2.1.3 选择变革方案

注意：从德克萨斯州的奥斯汀到华盛顿哥伦比亚特区，坐飞机的话大约需要三个多小时，但是如果基于预算限制无法选择飞机出行该如何是好呢？可以开车(22 小时)、坐火车(17 个半小时)或者走路(488 个小时，不推荐给普通人)。基于预算、时间、飞行恐惧以及徒步能力，我只能选择一种方式出行。

本章的剩余部分主要致力于描述需要采用的各种措施，以选择适合的 DevOps 能力去解决识别出的瓶颈问题。辨识并了解每项瓶颈问题产生的根本原因后，下一步就是确定有助于应对这些根本原因的 DevOps 能力。然而，这并非单纯运用 DevOps 能力那么简单。正如旅行案例中所表述的那样，要决定采用哪种 DevOps 能力以及如何实施，需要综合考虑业务、技术以及组织各个方面的因素，其中包括：

- 项目团队对流程、工具变化的接受能力

- 大型组织对于机构变化的接受能力
- 实施投资预算情况
- 项目时间表
- 成果展示时间
- 高层管理人员的认同与支持
- 操作人员的认同情况
- 文化惰性

在后面的章节中讨论具体措施及应用的时候，会对所有这些因素详细说明。

在确定适合的方案及 DevOps 能力时，需要考虑每项方案所对应瓶颈问题的以下四个方面因素：

(1) 流程改进

(2) 工具自动化

(3) 环境与平台

(4) 文化

要想取得成功，每一种方案都应该满足以上所有或是其中多个方面的要求。之后，在综合考虑业务、技术、组织各个方面因素的基础上，主导 DevOps 变革的团队要确保采用适合的方案针对性地处理不同的瓶颈问题。

2.1.4　实施变革方案

一旦选定合适的 DevOps 变革方案，接下来的任务便是推动实施，这需要付出更多的努力。关于大型企业推动 DevOps 实施的方案将在第 6 章详细介绍。然而，任何制定和实施 DevOps 变革路线图的团队都需要解决两个方面的问题，理解这两方面问题并提前做好准备至关重要：

(1) 解决生产率下降问题

(2) 克服文化惰性

现在就来探讨一下如何识别并解决这两方面问题。

1. 控制下降幅度

注意：引入任何变化都会立即导致生产率的下降，这是不可避免的。我在升级手机应用至新版本的时候曾经历过。运动队去外地比赛也会有这样的经历。所以会存在主场优势，因为人们都喜欢在舒适区作业。

在收获成果之前一定会经历生产率的下降。这是引入变化的自然结果，不管变化发生在流程、工具还是团队架构方面。好的变革推行方案会提前做好预案并采取行动将下降幅度控制在最小范围。最终，在生产率实现正增长后，如果流失的生产率(见图 2-3)相比于提升的生产率而言是微不足道的，那么就可称之为成功变革。当然，如果变革没能带来生产率的显著提升，那么这些就都没有意义了。

如果生产率的流失过于显著或者超出了团队的准备程度，那么项目通常无法扭转局面。为了杜绝这种情况发生、将生产率下降幅度控制在最小范围以及尽快实现生产率的提升，并且投入最少的时间与资源，就需要精心策划变革，并且由经验丰富的专业人员进行指导。

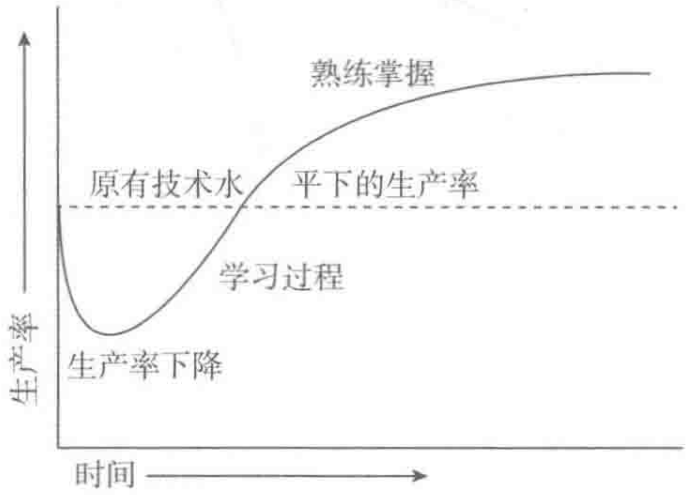


图 2-3　生产率下降图

敏捷教练在推行企业敏捷实施过程中的成功经验已被延伸应用至 DevOps 的实施推广，即利用 DevOps 教练推动 DevOps 的实施。IBM 以及一些专业的 DevOps 咨询机构已经培训出一批 DevOps 教练，加入到正在实施 DevOps 的项目团队或组织中。

DevOps 变革也确定了业务驱动相关的方案，因为这些业务驱动正

是最初用于衡量生产率的关键绩效指标(Key Performance Indicator，KPI)的驱动力。这些关键绩效指标应该做到以下几点：

- 在变革开始前确定(衡量成功)
- 可用作了解现状的基准
- 设定变革结束后的生产率关键绩效指标(结束状态)

将在第 3 章讨论对于不同实施方案如何取舍不同的关键绩效指标。

如果不清楚目标和得分规则，就无法赢得游戏(打比赛也是一样)，同样道理，如果不了解需要改进的关键绩效指标及其需要改进的程度，变革就无法取得成功。同样，正如高绩效的冠军团队离不开好的教练及有效的训练过程一样，要取得 DevOps 变革的成功，也必须要有专业的 DevOps 教练以及精心策划的变革路线图。

完美的计划并不意味着完全实现预期，每次都赢就不会进步。输的时候才会进步，输的时候才有提升，输的时候才更加了解自己。

——林书豪，美国 NBA 球员

---

2. 启动试点实施

实施 DevOps 变革是一个非常复杂的项目，需要投入大量精力。从细微处着手改善 DevOps 实施路线图并展示成果，对于整个企业层面的成功变革是必要前提。第 5 章将讨论企业层面推广的具体方案。然而，项目启动之初，还是需要一份据以在整个企业范围内实现成功推广的行动计划。这就需要选择一系列试点项目来执行 DevOps 实施路线图。

启动企业范围的 DevOps 实施推广，要先从选取的 3-5 个试点项目开始，待实施路线图被证实有效并经过完善后，再基于试点项目的经验在整个企业范围内进行推广。每个项目的目标都是在一个项目上应用一种 DevOps 能力，然后评估其对于解决那些需要特殊能力的特定瓶颈问题所产生的影响。之所以每一个试点项目上只应用一种能力，是为了展示所应用的能力对于要解决的瓶颈问题所产生的直接影响。每个项目上

应用多种能力的话，无法直接评估每一种能力所产生的影响。

最终目标是展示成功以及汲取经验教训，以便在其他应用项目中取得更好成果。如前所述，准确识别既定业务目标的关键绩效指标、启动试点项目前选好基准以及在应用过程中对关键绩效指标进行跟踪，这些工作对于展示成果(或好或坏)以及在项目进程中根据需要调整实施计划而言都是必不可少的。

每个试点项目都需要配备一位专职或是兼职的 DevOps 教练。至于需要的是专职还是兼职的教练，要视项目团队的规模而定。

那些对业务至关重要却非任务导向型的项目是作为试点项目的最佳选择。如果一个项目对组织不具有显著价值，以至于没有合格的团队成员以及相应的资金配备，这样的项目是不能作为试点的。如果一个项目规模太大并且地域分散，或是任务导向性太强以至于项目运行出现问题或是延迟都会危及重要业务，这样的项目也不适合作为试点。一旦试点项目取得成功，并且就其成功与不足之处进行了充分分析，就可以将经验推广应用至其他项目，包括那些任意规模及地域分布的大型的、任务导向型的项目。

3. 克服文化惰性

最后，即使已经完成所有的流程改进与自动化工作，如果无法克服组织内根深蒂固的文化惰性，DevOps 的实施变革也不会取得成功。大多数组织都有一种惰性，那就是对变化的本能抵触。改变很难，尤其是那些大型组织，这些组织中的文化经过多年的沉淀积累并潜移默化地影响着成百上千的操作人员。这些人员，作为独立个体，他们或许认可 DevOps 实施的价值，但作为一个群体，他们就会抗拒变化，进而形成惰性。克服这种惰性是关键所在。

这种惰性是如何表现出来的呢？“我们这里就是这样做的。”“是的，但是要改变 X 不是我所能控制的。”“你需要跟 Y 去谈；我们无法改变他人的工作方式。”这些都是体现文化惰性的言行表现。久而久之，组织发展出了行为规范；团队和群体按照组织划分职责分工；以管理之名设立检查平衡机制，纵然这并非真正意义上的管理；流程存在，却没人

去思考其中的原因，只是视作“本来就有”；报告写出来已无人阅读，却没人愿意提出取消；以往发生的事故通过加强审批防止其再次发生，等等。所有这些行为形成了组织文化的惰性。

要克服文化惰性，需要对可能导致组织低效的每个构建物以及每一项流程进行深入的审查，即审查导致瓶颈问题及浪费的根本原因。要想解决这些瓶颈问题，需要改变的不仅仅是传统的 DevOps 实践，还需要有改变文化的强烈意愿，上至负责并引领变革的高管，下至执行变革的普通员工。管理层需要为改变行为模式的操作人员提供支持与保护，这将对外部那些执行变革的人产生积极影响并将悄然打破传统的管理模式。操作人员本身也需要改变行为习惯，打破职能壁垒，为变革代言。

支持与投入

只有结合自上而下的支持与自下而上的投入去克服文化惰性，DevOps 方案才能有效实施并充分发挥其潜在价值。发起及推动变革需要自上而下的支持。真正实施变革则需要自下而上的参与及投入。

第 6 章将详细讨论管理人员如何支持并推动 DevOps 变革。

利用本章所述技巧为真实客户(匿名)开发的路线图案例，请见附录 A。

2.2　总结

综上所述，与运动队的教练和队员准备、进行以及赢得比赛的情况不同，在实施 DevOps 时，教练及领队需要研究实施条件以及对手情况，需要了解对手的优势及不足，现有条件及竞争力水平。然后从指导手册中选择适合的方案并组织实施。最核心的工作是完善实施方案直至完美，将实施方案导致的生产率下降保持在可接受的范围内，或者克服从之前不同方案中沿袭下来的惰性，直至实际运行方案与实践结果一致。

通常而言，实施 DevOps 需要：

- 确定目标状态
- 理解现状
- 选择适合方案实现由现状到目标状态的转变
- 做好准备解决生产率下降问题
- 结合自上而下的支持与自下而上的投入以克服文化惰性

本书其他章节将对这些内容进行论述。

第 3 章

开发 DevOps 变革的商业案例

美国职业足球大联盟发展的商业案例

如果要寻找一个快速发展的体育项目管理组织，会有一个意想不到的结果：美国职业足球大联盟(Major League Soccer，MLS)。

随着足球运动在美国的不断普及，自 2007 年以来，美国职业足球大联盟已经扩充了 9 家新的俱乐部。与此同时，联赛的上座率现已超过国家曲棍球联盟及国家篮球协会，收视率已较十年前大幅提升，尽管 2013 年曾经出现过下滑。

其成功还体现在财务方面；哥伦布机员的售价超过 6000 万美元，这还只是市场中等价位。尽管与国家棒球联盟及国家橄榄球联盟的价值相比，这个价格显得微不足道，但已创下国家足球大联盟的记录。

伴随着 18 个赛季的纪录，美国职业足球大联盟已跻身美国各运动联盟前列。然而，其仍面临巨大挑战，要去与全世界的足球运动项目竞争以发展更多球迷。总干事唐·嘉伯向公司讲述了联盟现今发展阶段的一些经验及接下来的发展计划。

快速发展总会伴随着失控。为此，保持成功经验并在问题发生的时候进行迅速诊断是至关重要的。美国足球大联盟持续记录其关键绩效指标，嘉伯将其列为以下几项：国家与地方级的电视及媒体覆盖率，足球专用体育场的发展，电视收视率以及体育场馆上座率。

尽管这些指标在过去几年里都保持增长态势，最后两项指标，即收

视率和上座率却在 2013 赛季出现了下滑。为此，嘉伯和美国职业足球大联盟很快确定了导致这种短暂下滑的原因：一个球队的上座率表现特别不好，同时，电视宣传方面还有提升空间。现在的时机很好，因为联赛的电视转播权正面向市场公开发行。

嘉伯指出，在业绩出现下滑的赛季，联盟对其总体规划还是充满信心的。快速发展所带来的偶然性问题并不意味着全盘皆输，反而意味着分析与适时调整。

嘉伯提到："我们制定年度计划并自上而下推行，和俱乐部所有者一起工作，以确保我们拥有相应的资源来实现计划。这种方案在过去的多年间一直有效运行。""过程中难免会遇到一些障碍，这时就需要我们机智、敏捷应对，并集中精力调整计划以适应各种宏观与微观的市场问题。"

——Vaccaro，2014

引领 DevOps 变革的高管人员要负责为推行 DevOps 变革而开发相应的案例，并将其推销给组织(首席执行官、首席财务官以及可能涉及的董事会成员)。在本书中，将列举一些令人无法拒绝的理由，这些理由将成为撬动 DevOps 变革的导火索或是催化剂。第 7 章的内容会对这些理由进行全面论述。然而，适合且有说服力的商业案例还是不可或缺的。商业案例不仅要向组织提供业务方面的价值主张以进行必要的商业投资，还需要保证投资的持续进行以确保变革所必需的时间及资源，从而对业务产生影响。归根结底，DevOps 实施并非一劳永逸的项目，而是一场旷日持久的根本性变革。

3.1 开发商业案例

DevOps 通过优化流程、工具、平台以及文化能够使应用程序交付流水线变得更加精益和高效，这是不争的事实。然而，商业活动还是需要有确切数据支撑的案例来对其投资回报率(Return On Investment，ROI)

进行说明。实施 DevOps 究竟能为组织带来多少商业价值？

要获得商业价值数据，可以利用以下几种技术手段：

- **案例研究。**关于 DevOps 工具、平台以及变革服务所能提供的商业价值及投资回报率，产品及解决方案供应商已经进行了很多的案例研究。可以利用已经实施 DevOps 的同类组织作为案例，研究其所获得的商业价值及实现的投资回报率。如果这些组织愿意分享其在实现商业价值目标过程中的经验与教训，与这些组织进行对话也是很有意义的。
- **分析研究。**外部的分析机构也做了很多的相应研究。比如，福雷斯特公司(Forrester)针对工具与实践的总体经济影响(Total Economic Impact，TEI)研究(Forrester，2013，2015)。这些研究结果可以用来评估组织自身所能获得的商业价值。
- **价值流图。**价值流图(VSM)在第 2 章中有过详细的论述。本书旨在把价值流图视作一种技术手段，用以识别交付流水线中所产生的浪费，并开发出 DevOps 实施路线图来解决浪费问题。然而，价值流图的应用可以进一步深入到时间动作(time and motion)模式的研究，不仅识别出浪费，还要量化出生产过程中确切的损失并确定发生领域。之后便可以创建价值流图去解决那些影响财务回报的浪费问题。
- **商业价值评估。**商业价值评估(Business Value Assessment，BVA)是通过评估这些 DevOps 功能应用所带来的财务回报，选取合适的 DevOps 功能来开发价值流图的常规手段。大部分供应商(产品及解决方案供应商)都会采用的商业价值评估工具，是利用其预设的公式首先计算出当前实践(运用目前采用的工具、平台以及流程)的累计成本，然后再计算出按计划实施 DevOps 工具、平台以及实践所需的成本(采购、部署、培训、使用)。

上述所有技术都可以用于聚焦创新(详见下一章中 DevOps 优化与创新的区别)项目中的 DevOps 实施，但是还有另外一种“曲解”需要在这里提及。多数情况下，创新方案的提出需要经过一系列的实验。这些实验有的针对创新科技方案，有的针对商业模式，更多情况下是同时针

对这两项。其结果是，所开发的商业案例无法做到非常具体。鉴于最初推动实验需求的因素都不确定，确切的商业价值无法计算，这样就需要创建类似“风险投资”的商业模式。即在多个启动项目上分别投入少量资金，通过其中某个项目的发展和成功取得显著收益。这样不仅使成功项目上的投资实现了回报，其他未能成功的项目上所损失的投资也能获得相应的平衡。

很多技术手段已经问世，用来在商业活动前期阶段甚至开发参与之前分析其可能获取的商业价值。这些技术有利于定义创意、确定投资规模以及量化可能带给客户与业务的商业价值。

不谈技术细节，仅就商业模式而言，创新主张对科技组织提出了有趣的问题，那就是该如何确定创意方案的可行性，如何确定其商业价值就更不用说了。有一项技术已经在精益创业领域越来越受欢迎，那就是亚历山大·奥斯特瓦德和伊夫·皮尼厄在著作《商业模式新生代》中所介绍的“商业模式新生代”。不同于其他技术，这本书中利用商业模式画布(Business Model Canvas)来定义、理解以及完善商业模式的内容，其中包括了对于组织的商业价值。

如图 3-1 所示，画布包含 9 大模块内容，每个模块都旨在定义并进一步完善商业模式，确定其可行性，了解相应的风险因素及所需资源，并最终实现商业价值。具体内容如下：

(1) 客户细分：所服务的客户群体

(2) 价值主张：解决不同客户群体各种问题所需的多种价值主张

(3) 渠道通路：传递价值主张的各种方式

(4) 客户关系：与不同客户群体间所确立的客户关系

(5) 收入来源：基于价值主张所产生的收入来源

(6) 核心资源：交付价值主张所必需的资源

(7) 关键业务：利用既有资源交付价值主张所需进行的业务活动

(8) 战略伙伴：为了交付价值主张所需合作的战略伙伴、经销商及供应商。

(9) 成本构成：交付价值主张所需的资金及资源的固定投入

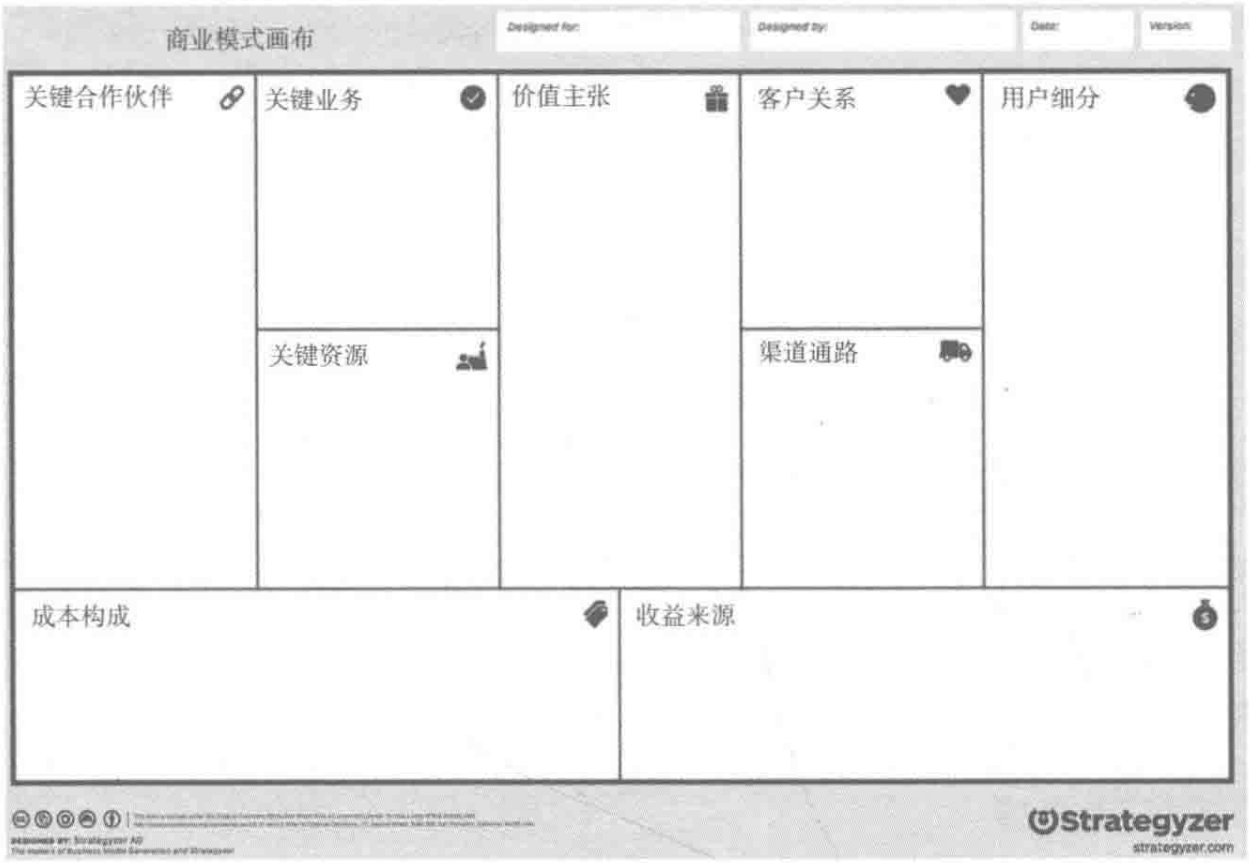


图 3-1　商业模式画布[①](Strategyzer, 2013)

这种方法为组织考虑投资的每个创意提供了全方位评估模型，可以帮助组织做出更明智的投资决策。在 DevOps 环境下，也可以利用画布来创建价值主张商业模式，为推行 DevOps 变革提供决策辅助。

3.2　完成商业模式画布

接下来就可以填充这些模块内容来创建商业模式。这样就可以为 DevOps 变革创建商业模式画布，负责变革的高管人员则可以利用其开发商业案例。可以参考本章后面的内容，填写、绘制不同组织的商业模式画布并开发 DevOps 商业案例。

基于这次实践的目的，我更倾向于把业务线及 IT 组织视为两个不

① 在创作共享协议允许下复制。许可条款请见：https://creativecommons.org/licenses/by-sa/3.0/。商业模式画布可从 https://strategyzer.com/canvas/business-model-canvas 下载。

同维度来理解业务模式画布，并分别创建本质上各不相同的两份画布。

因为这是在 DevOps 变革环境下，所以只涵盖了组织所提供的 IT 功能。倘若组织需要向客户提供与 IT 应用或服务无关的其他服务，这些服务则不在本次实践的考虑之列。例如，银行向组织发放现金用以支付工资，用来申请和管理现金的软件就在所述范围内，但是实际的交付服务，运钞车运送现金的服务则不在讨论之列。

3.3 客户细分

这部分所讨论的是组织所服务的客户群体。如前所述，需要从业务线及 IT 部门两个不同维度来进行分析，二者分别拥有不同的客户群体。

3.3.1 业务线

业务线的最终目标就是向终端用户及客户提交业务能力。所以，其客户细分如下：

1. 终端用户。终端用户指的是所有使用应用程序的人。与“客户”的不同之处在于他们并不一定为组织带来直接收入。例如，Facebook、Twitter 或者 Pinterest 这类社交媒体应用的终端用户并不会为使用这项服务而付费。但是，在终端用户使用其服务的时候，这些公司得以收集到相关数据并可以将这些数据变现，其真正的客户是为其服务付费的那些人。人口结构、地理分布以及使用水平等因素的不同造就了终端用户的多元化。

很多人都会混淆“终端用户”与“客户”的概念。以 YouTube 为例，终端用户指的是浏览或者上传视频的所有人，但 YouTube 还会在这些终端用户中细分出超级用户。这些超级用户上传的视频点击量很高，以至于其可以参与 YouTube 的分红。因此，这些终端用户会被视作特殊客户群体。至于 YouTube 视其为终端用户还是客户，就看他们的划分标准了。

2. 客户/顾客。客户指的是为组织所提供的应用服务付费的人。在社交媒体应用的案例中，客户是那些以终端用户为对象，在应用程序上

付费投放广告的广告商。这里需要特别注意的是，客户也可能再拥有他们自己的客户。这种情况在 B2B(business-to-business)应用中最为典型，当然其他的平台产品也会有类似的情况。组织主要客户的客户是最后的终端用户。比如社交媒体网站，如果一个客户在供他人使用的平台上创建了应用程序，那么这款应用的终端用户就成为应用程序供应商以及社交媒体平台的客户，而应用程序供应商以及应用程序的终端用户同时还是该社交媒体网站的终端用户。可以想象成 Facebook 上运行的 FarmVille 游戏。FarmVille 的用户是游戏厂商 Zynga 的客户，但他们又都是 Facebook 的终端用户。反过来，Zynga 则是 Facebook 的客户。

3. 客户代表。很多情况下，组织与其终端用户及客户之间的接触通过雇员或合作伙伴来实现。客户与组织之间的互动是通过使用数字应用程序及接受服务的代表人物来实现的。在非数字化客户互动中，则是通过雇员来实现的，这些雇员因此成为业务线的细分客户群体。以银行柜员、客户服务代表以及代理(雇员或者外部合作伙伴)为例，作为一个细分的客户群体，他们需要作为非数字化终端用户及客户的代表，并以能够传递给终端用户及客户的方式提供适当的商业价值。

社交媒体的情况有所不同，最接近的情况莫过于知名人士、总经理或者公司聘请公关经理或团队代表其与社交媒体互动合作。然而，对社交媒体而言，这个用户与其所代表的终端用户或者客户不存在差异。但对于自己拥有柜员的银行或者通过特许经营代理开展销售的保险公司，其与组织合作的性质完全不同，这就需要将他们视作单独的客户群体。

3.3.2　IT 组织

对于一个 IT 组织而言，业务线是其真正的关键客户群体。当然，所有业务线的客户群体也是 IT 组织的间接客户群体。此外，IT 组织还拥有自己的内外部客户群体：

1. 内部终端用户。包括组织内部作为终端用户享受 IT 服务的雇员、承包商以及合作伙伴在内。还应该包括利用企业-雇员业务应用程序及服务(e-mail、内部门户网站、人力资源应用程序、薪酬支付系统等)的客户在内。

2. 应用交付利益相关者。其中包括所有在应用交付流水线工作的利益相关者：开发、测试、运维、业务分析、架构以及安保人员，如前面章节所述，所有这些人都是实施 DevOps 的利益相关者。关键应用交付利益相关者包括以下这些：

- 开发人员
- 测试人员
- 运维人员
- IT 主管
- 业务负责人(来自业务线)

3. 客户 IT 组织。指的是隶属于接受 IT 服务的客户的 IT 组织，这些客户利用所交付的 IT 服务去开发及交付其自己的增值应用及服务。客户 IT 组织的关键利益相关者同上所述，只是这些人是服务于客户组织的雇员。

在此前列举的 Facebook 社交媒体案例中，客户 IT 组织指的就是 Zynga 的 IT 组织。

3.4 价值主张

业务线及 IT 组织接下来要做的事情就是，针对此前细分的每一类客户群体，确定能够为其解决多种问题的价值主张。

3.4.1 业务线

针对不同的客户群体，业务线需要理解并确定其想要传递给客户的各种不同价值主张。

1. 终端用户：传递给每一类细分出的终端用户群体的价值主张各不相同。所交付的业务功能不同，价值主张也会有所差异。这些传统功能是否需要优化？这些功能是否标新立异？正确需求、业务模式以及交付模式还有哪些方面有待通过实验去发现、确认以及完善？业务线的价值主张包括：

- 业务功能要确保终端用户能够按照业务线的期望获取相应商业价值
- 业务功能的交付方式要符合不同客户群体的使用需求
- 业务功能需要定期更新以满足不同客户群体不断变化的需求与期望
- 需要通过实验来确定不同客户群体的用户体验
- 需要为新用户群体进行实验以完善用户引入及商业模式
- 业务功能要以优于竞争对手的方式交付给客户群体

仍以社交媒体网站为例。如前所述，YouTube 将客户细分为普遍用户与超级用户，超级用户基于其所上传视频的点击量获得网站分红。对普通用户的价值主张很简单，只需要确保其在地域及法律管辖范围内能够观看视频，同时保证视频的分辨率、画质以及速度。他们还可以订阅其他用户的视频，发表评论，将视频分享至其他社交媒体网站并且可以为视频点赞(社交媒体网站上的最高认可)。此外，他们可以轻松上传各种格式及分辨率的视频文件。除了常规用户的这些价值主张，超级用户还应该收到通过上传视频取得收入的价值主张，而这些收入是基于同样服务于普通客户的广告投放。他们可以上传高质量的视频，可以清楚计算点击量及相应收入，可以控制投放广告的类型以及选择授权广告商，当然，他们还能够以及时精准的方式获取应得收入。

2. 客户：客户或顾客是指为服务实际付费的群体。所以，提供给客户的核心价值主张除了终端用户的所有价值主张外，还应该使其能够获得付费购买的商业价值。传递给客户的价值主张包括：

- 以市场上有竞争力的费率向客户交付业务功能
- 以经过客户同意并明确定义的服务标准交付业务功能。基于不同的定价机制及标准，服务水平也多种多样
- 针对不同客户群体制定不同的定价机制
- 通过与定价以及采购模式相关实验去验证针对不同客户群体的定价机制

对于通过扩展组织所交付的应用与服务来开发自己的应用与服务的终端用户及客户而言，价值主张还包括使用合适的应用程序与服务的能

力，他们利用这种能力向自己的终端用户及客户提供相应的价值主张。

3. 客户代表：因为客户代表实际上代表终端用户以及与之互动的客户，所以价值主张基本一致。除此之外，还涉及客户服务方面角色的相关价值主张。他们能够以超越其他任何工具的最高客户满意度水平向客户提供商业价值。他们还要能够回答终端用户及客户提出的关于组织及其提供的所有服务的相关问题。相比于交付给终端用户或者客户的应用与服务，交付给客户代表的应用与服务需要更广泛的交互功能。

3.4.2 IT 组织

从 DevOps 角度来看，IT 组织所提供的价值主张取决于组织的业务目标，即组织想以何种方式向其客户群体提供商业价值。针对终端用户及客户(内部及外部)的不同价值主张包括：

- 高质量(零缺陷)的应用与服务交付
- 新用户轻松注册
- 不想使用的用户可以轻松关闭及删除账户
- 从其他应用程序导入用户数据的能力
- 导出用户数据至其他应用程序的能力
- 终端用户问题及客户支持请求的快速响应
- 运行稳定——服务中断时间最短，包括维护时间(比如，Facebook 网站从不会出现“定期维护”的情况，即使因为系统升级)
- 用户数据不会丢失
- 符合所有法律法规的规定
- 残疾用户的访问选项
- 支持全球化多种语言及货币
- 用户与客户的数据及隐私保护
- 安全服务
- 高客户满意度

针对作为应用交付流水线利益相关者的 IT 雇员及供应商所提供的价值主张包括：

- 高可用性、稳定及安全的基础设施与平台
- 良好集成工具构成的合适的应用交付工具链
- 交付流水线之间端到端的可追溯性及可视性
- 引入适合的敏捷及 DevOps 实践与流程的能力
- 不同功能的仓筒及交付流水线之间的沟能与协作能力

现在来回顾一下前面部分列出的关键利益相关者并确定其典型的价值主张。

开发人员：

- 利用自动构建、测试及应用部署，在两周的迭代周期内将应用或服务变更交付至生产的能力
- 快速评估应用或服务的性能及功能特性以便于在未来迭代中持续改进的能力

测试人员：

- 向“类生产环境”部署新版应用或服务，利用自动化工具进行测试并验证测试数据的能力
- 向开发人员反馈所发现的问题，以便开发人员能够及时纠正问题的能力

运维人员：

- 通过逐步提升质量与速度，高效交付全自动化 IT 流程与环境，以平衡风险与成本
- 通过使用能显著提升服务质量并降低操作成本的分析和认知操作系统来有效预测并预防生产宕机

IT 主管：

- 访问度量仪表盘，洞察应用程序与服务及其交付环境的总体表现
- 访问度量仪表盘，获取应用交付流水线上每个利益相关者及操作人员的详细业绩标准
- 持续改进应用交付程序、工具、平台，建设不断提升时间价值的团队文化

业务线负责人(来自业务线)：

- 尝试业界新创意，用来打造不一样的客户体验并识别新的商业机会
- 访问业务仪表盘，洞察业务单元的总体表现以不断提升经营业绩(收入与支出)
- 分析用户在使用应用或服务时的行为与情绪反馈，用来决定如何提升客户参与满意度以及提高商业回报的相应举措

对于客户的 IT 组织，价值主张则包括：

- 通过良好定义的 API 与服务提供高效、稳定、安全的服务
- 对问题及客户服务需求进行快速响应
- 对新增需求及增强功能请求进行快速响应
- 清晰的路线图以及对未来应用与服务需要增加、修订或者删除的相关能力的洞察

当然，所有这些价值主张都是非常主观的，而且基于应用类型及客户群体的差异而各不相同。他们通常都表现为对所交付应用与服务的非功能性需求。

3.5 渠道通路

渠道指的是组织向不同客户群体传递价值主张的各种方式。

3.5.1 业务线

对于业务线而言，IT 组织所交付的应用与服务通常是其向客户群体传递前面所述的价值主张的唯一渠道，其中包括：

- 直接可被终端用户及客户(网页、手机、桌面等)所使用的应用与服务
- 由与终端用户及客户互动合作的代表们所使用的应用与服务
- 客户为了开发及交付其自己的应用与服务而使用的服务(API 集合、平台、数据源等)

3.5.2　IT 组织

IT 组织的价值主张传递渠道与业务线一样。IT 组织有责任提供这些通道所需的所有应用及服务来启用这些渠道。因此，IT 负责所有的渠道。

3.6　客户关系

接下来要做的是获取客户关系，在通过渠道向各个客户群体提供各种价值主张的过程中，使这些客户关系得以建立、发展或加强。

3.6.1　业务线

业务线拥有三类客户群体的客户关系：

- 现有关系下的现有客户
- 关系有待进一步探索或发展的现有客户
- 新关系下的新客户

业务线通过更好的客户服务以及用户体验的提升来改善客户关系。

3.6.2　IT 组织

IT 组织的目标是通过其交付的应用和服务与客户建立联系。如前所述，业务线拥有三类客户群体的客户关系，而除了业务线的这些客户关系之外，IT 组织还利用自己建立起来的联系，发展出更多的客户群体。

IT 组织通过提升所有相关人员的体验，授权他们参与应用与服务的不断完善、交付流程、工具、平台以及组织文化的持续改进，来改善所有客户细分中的客户关系。

3.7　收入来源

任何业务向客户提供价值的核心成果都是收入。因此，需要获取每

个客户群体所能带来的各种不同的收入来源。这里需要特别说明的是收入并不一定是组织的最终目标。政府机构向公民提供价值并非为了取得收入(税费除外)，但是也需要确定相应的标准来衡量价值交付所带来的“回报”。

3.7.1 业务线

业务线在收入方面要实现的目标是：

- 提高现有收入来源
- 通过尝试新的能力及新的商业模式发展新的收入来源
- 通过让现有及新的业务能力变现来发展新的收入来源，这要通过利用这些能力来发展其增值服务的合作伙伴来实现

3.7.2 IT 组织

IT 组织在收入方面要实现的目标是：

- 优化现有应用与服务使现有收入来源的提升机会最大化
- 促进业务线去尝试新的能力及商业模式并对实验结果给予快速反馈
- 为合作伙伴提供平台，使其能够利用这些 API 及服务去创建自己的解决方案

3.8 核心资源

组织用来创建及交付价值主张的资源将在下文中进行介绍。这些资源同时也是交付价值主张的投资与成本的衡量标准。

3.8.1 业务线

业务线可用的核心资源包括：

- 包括 IT 组织在内的各个组织的人力资源
- 组织开发及拥有的知识产权

- 可投资于开发及交付应用与服务的财务资源
- 关系资源与合作伙伴

3.8.2　IT 组织

IT 组织可用的核心资源包括：

- 服务于 IT 组织的雇员及外包人员
- 在交付服务的过程中，IT 组织的合作伙伴及供应商
- IT 组织开发及拥有的知识产权
- IT 组织有关开发及维护的流程、工具、平台以及组织文化

3.9　关键业务

下一步要记录的是利用资源交付价值主张而需要开展的各种业务活动。这些业务活动同时也驱动投资与成本控制。

3.9.1　业务线

业务线的关键业务活动包括：

- 通过现有及新开发的应用与服务(渠道)向客户交付业务价值主张
- 尝试新的应用程序、服务以及新的商业模式，以改善客户体验、抢占新的市场份额并通过合作伙伴与供应商开发新的收入模式
- 获取并分析客户群体的反馈意见，以持续提升将要交付的业务能力与客户体验，还有已有的交付渠道以及所利用的商业模式

3.9.2　IT 组织

IT 组织的关键业务活动包括：

- 开发并交付应用与服务以向客户群体提供商业价值
- 基于用户与客户的反馈以及业务线的指导意见持续改进已经交付的应用与服务
- 提供对新的特性、用户体验以及新商业模式进行实验的能力
- 为合作伙伴与供应商提供 API 集合及服务

- 为应用交付流水线上工作的利益相关者提供稳健的流程、工具、平台以及组织文化
- 为应用交付流水线上工作的利益相关者提供反馈，促使他们持续改善即将交付的应用与服务、目标交付环境以及交付所使用的流程、工具、平台及文化
- 向业务线提供应用与服务的功能、性能、用户体验、使用模式、用户情绪方面的反馈意见

3.10 战略伙伴

合作关系包括有助于商业价值交付的所有业务伙伴以及供应商。详细内容见下文记录。

3.10.1 业务线

业务线拥有相应的合作关系以填补客户群体的缺口，扩大客户群体的接触范围，而这些客户群体是依靠其自身能力无法触及的。

- **合作伙伴。**指的是利用组织交付的服务去提供自己的增值服务的任何其他组织。在社交媒体应用的案例中，合作伙伴是利用所交付的服务提供自己的产品与服务的组织。想象一下那些利用 Facebook 或者 Twitter 作为其网站登录方式的公司，他们利用 Facebook 或者 Twitter 的身份认证系统进行用户识别，登录网站时不再需要其他的用户名及登录密码。再比如在这些平台上发布游戏(FarmVille)或者他们自己的应用程序(Periscope)的第三方公司。(的确，客户与合作伙伴间存在着重叠，很多情况下既是客户又是合作伙伴，一些竞争对手也是合作伙伴)
- **供应商。**供应商也发挥着关键作用。是属于客户群体还是属于战略伙伴，或者两者兼而有之，取决于其与组织之间的关系。如果双方是互惠互利的合作关系，供应商专门为组织提供经过改善的核心服务，而组织和供应商均能从共同交付的附加价值中获益，便可将其归属于客户群体。还是以社交媒体为例，如

果一个体育组织(比如美国国家橄榄球联盟)在 Twitter 上转播赛事以开发新的市场，那么其既属于客户群体又属于战略伙伴。

3.10.2　IT 组织

IT 组织同样需要合作关系来填补技术能力与服务的缺口，并扩充获取渠道。

- 应用交付流水线上的相关供应商与合作伙伴。指的是来自组织外部为应用交付流水线提供组件的第三方供应商与合作伙伴，其中包括应用服务供应商(提供 SaaS 及被调用的 API 集合)，基础设施或平台服务供应商(IaaS、PaaS 以及 CaaS)，以及其他的技术供应商。
- 增值合作伙伴。指的是利用组织提供的 API 集合与服务交付自己的增值服务的组织。他们同样归属于组织的客户群体。

3.11　成本结构

和所产生的收入一样，组织还要记录向所有客户群体交付最终价值所需投入的成本与投资。

3.11.1　业务线

业务线的成本结构直接映射到此前所述的核心资源。所有的人力、技术以及 IP 资源都需要成本。一部分是基本建设支出(Capital Expenditures，CapEx)，一部分是日常运营支出(Operational Expenditures，OpEx)。业务线的目标之一就是管理这些成本以实现盈利与投资回报(Return On Investment，ROI)最大化。

3.11.2　IT 组织

对于 IT 组织而言，之前所述的每项核心资源也都需要成本。通常 IT 组织在运营及维护现有应用与服务方面需要投入巨大成本。如今，大多数 IT 组织都致力于优化这些成本，并将释放出的资源投入到创新方面的工作中去。

3.12 总结

附件中的示例为试图创建 DevOps 变革商业案例的组织提供了商业模式画布的模板。在填充画布的时候，商业案例所有的关键模块都会被记录在内。画布还可以作为工具来验证组织向客户交付应用与服务的方式以及所进行的投资。部分验证内容如下：

- 应用与服务的交付方式、相关成本以及所获得的收益流能否证明现有的应用交付模式有效？
- 优化流程及实践是否有利于提高投资回报率？
- 组织是否应当创建或采购新的应用与服务？
- 组织可以释放哪些资源投入创新工作中？
- 改进组织的流程、工具、平台以及文化需要投入什么资源？

如图 3-2 所示，利用商业模式画布，结合第 2 章描述的用以识别交付流水线中浪费发生领域的价值流图，组织便可以构建 DevOps 变革路线图以及商业案例，有了以上工具，就可以开始投资转型工作。

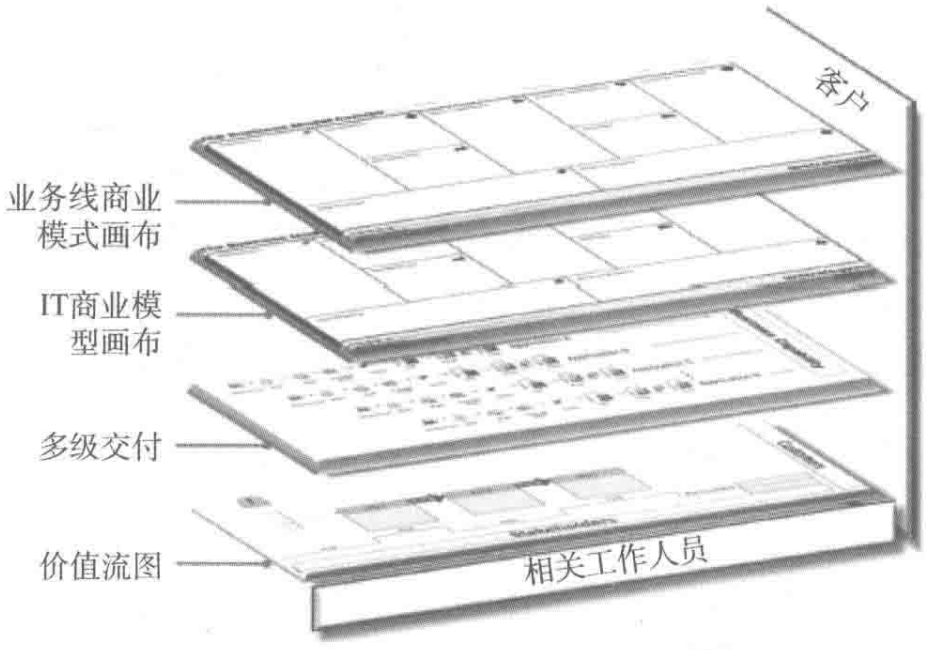


图 3-2　创建 DevOps 实施商业案例

第 4 章

DevOps 方案之优化持续交付流水线

点球成金：优化创新

2002 年奥克兰运动家棒球队的命运已经注定。他们只有 4 000 万美元的预算，而其竞争对手不仅包括工资总额达 12 500 万美元的杨基队，还有其他 28 家平均工资总额均达到 7 000 万美元的球队。作为总经理，比利·比恩(布拉德·皮特饰)在影片中(我凭借记忆复述)提到："有富有的团队，有贫穷的团队，还有我们。"这是一场不公平的角逐。比利告诉他的球员，如果他们打算像杨基队发现人才时表现得那样，那么他们终将在场上输给他们。

在这种形势下想要取得胜利，唯一的办法就是通过创新发掘新的制胜之道。比利找到了出身耶鲁大学经济学专业，突破守旧棒球思维禁锢的彼得·布兰德(乔纳·希尔饰，虚构的人物形象布兰德是以哈佛经济学专业的 Paul DePodesta 为原型的)。布兰德认为，当前的棒球思维已经过时了，关乎球员价值的信念受各种偏见的影响，从而导致对球员价值的误判。市场的低效让奥克兰运动家棒球队有机会通过减少高薪球员来实现在预算的范围内组建有竞争力的团队。

利用统计分析以及赛伯计量学提出者比尔·詹姆斯所推广的理念，

尽管这些理念在棒球业内被广泛忽视，奥克兰运动家棒球队发掘其能负担得起的好球员。他们找到三位看似有缺陷的球员——Scott Hatteberg(手臂有问题)、David Justice(年纪过大)以及 Jeremy Giambi (负面新闻)，这些人拥有一项被市场所低估的技能：上垒。球队找到替补投手 Chad Bradford，他不受欢迎是因为他投球的方式很另类而且速度不够快。但是 Bradford 能够阻止对手上垒，至于他是如何做到的，球队并不在意。

很多人指责《点球成金》，是因为奥克兰运动家棒球队致力于棒球资料统计分析，结果挑战了神圣的棒球传统理念中的很多原则。争论在于，过于关注数字而使比赛失去人性化，并且忽略了只有经过训练的球探才能发现的内涵。尽管棒球资料统计分析是故事元素之一，却不是《点球成金》所真正要表达的内容。

广义上讲，《点球成金》是一个创新成功的故事，或者像比利在电影中所说的那样“要么适应，要么死亡”。比利是一位打破常规大胆创新的企业家。

——Bradbury，2011

4.1 DevOps 作为优化运动

一切都可以归结于在环境约束条件下进行优化创新实现最大产出。举例来说，可以研究一下《点球成金》，看看奥克兰运动家棒球队是如何在薪资水平明显受限(2002 年，奥克兰运动家棒球队的薪资水平在联盟里排名倒数第三，仅高于坦帕湾光芒及华盛顿国民队)的情况下组建获胜团队的。也可以读一下基恩·金的《凤凰项目》，这是一本非常精彩且必读的商业小说，里面介绍的是如何在各方面条件都受限的典型 IT 项目中实施 DevOps；或者可以回过头去研究一下艾利·高德拉特博士的商业小说《目标》的根源(实际上正是这本书激发了《凤凰项目》的创作灵感，里面介绍的是如何在充满各种不可避免的现实约束的环境中实现持续改进)。这其中任何一种方式都能帮助人们领悟到在商业世界取得成功的真谛，那就是，必须创新性的优化操作模式以在受限的操作

环境中最大限度地提高生产效率以及减少浪费。尽管戈德拉特博士将其视为“约束理论”，但这些例子实际上并非理论案例，而是利用优化实现最大产出的实践案例，从球队薪资与技能的优化到在 IT 与制造领域普遍存在的受限环境中实现优化。

在为应用交付以及更广泛的IT领域引入优化创新的一系列尝试中，DevOps 是最新的理论。源于精益思想的实践及经验教训已在制造及其他领域成功应用几十年，将这些实践与经验应用到应用交付及 IT 运维方面，之前也已经有过很多尝试。DevOps 在其他优化活动未见成效的领域取得了成功，是因为之前所有的尝试都是集中在应用交付流水线的一个、至多两个领域，比如 Dev-test(敏捷)、运维(信息技术基础设施库，或 ITIL)、项目管理(六西格玛)。DevOps 取得成功是因为采取了整体分析方法，对整个交付流水线的各个功能领域进行优化。事实上，优化活动最初的前提和意图是要将沟通协作的文化传递至各个功能仓筒并最终消除现存各种用途的仓筒(这些仓筒可能仍会存在于组织汇报架构中，关于这点本章及后续章节会进行讨论)。DevOps 之所以能取得成功，是因为它将影响到各功能与组织仓筒的文化变革置于流程与工具之上。DevOps 首先是一项文化运动。实际上，大型组织实施 DevOps 之所以困难重重，是因为这些组织把体制与流程置于文化之上，关于这点将在第 6 章进行讨论。

在本章以及接下来的两章中，将从体育运动队打赢比赛的视角来看待 DevOps，即基于运维所受的各种限制，试图优化应用交付流水线，该采用何种 DevOps 方案去实施文化、流程以及自动化实践。我既会审视人人都可以按需使用的普遍技术方案，也会审视为专有技术(技术驱动需求与限制)而特别设计的特有技术方案。而且，还会根据其背后的业务目标是否是实现优化或创新来将这些方案进行分类。每一项以创新为核心的应用交付工作都需要优化，但反之则不然。当谈到 DevOps 的实施时，将创新作为主要目标的应用需要特别对待(这个问题将在下一章详细讨论)。本章内容主要关于优化。不过首先，要充分了解应用优化与创新之间的区别。

业务目标：优化与创新

在审视可以用来推进 DevOps 变革进程的各种方案之前，有一点必须了解，那就是选择方案时不仅要考虑正在解决的瓶颈问题，还需要考虑到执行方案的应用其背后的核心业务目标。组织有两类业务目标：一类以创新为核心(速度与敏捷)，另一类以优化为核心(稳定性、可持续性，同时也兼具速度与敏捷)。按照创新或优化的不同业务目标将应用进行分类有助于选择更有可能运行良好的 DevOps 方案。

基于这两种业务目标，组织内的应用可以划分为两类：

(1) 核心业务，提供保持业务可持续发展的核心业务能力。主要业务目标在这里指的是优化。

(2) 边缘创新，为了开发和识别新的商业模式以及用户参与方法而进行实验。主要业务目标在这里指的是创新。

业界曾做过很多尝试，将应用或一般意义上的信息技术作为双速或双峰进行分类。

为分类所做的这些努力或基于速度，或基于需求变化程度。双速 IT 意味着一些应用变更得更快，而另一些则较慢。然而，速度并非最好的分类标准。另一方面，双峰意味着应用需求是稳定的还是不断变化的。这两种分类方法都有局限性。实际情况更倾向于正态分布而非双峰分布。

- 很少单独依据速度对应用进行分类，因为一些非常稳定的应用也能够快速交付变更。已存在几十年且极其稳定的大型机应用，拥有非常了解情况的开发团队，也可以进行快速更新。另一方面，使用敏捷以及现代语言开发的创新型应用，虽然利用平台即服务(PaaS)，但如果团队对平台或者技术堆栈的情况不熟悉，交付速度也可能非常缓慢。
- 需求变更的频率和程度也不是很好的分类标准，虽然它优于速度标准。即便是像银行核心业务系统这种很好理解的业务系统，在当今行业规定日新月异、地缘政治动荡不安的环境下，也可能产生千变万化的需求。另一方面，有着极其明确的需求，比

如吸引更多用户利用炫酷滤镜发布更多自拍的需求，这样的新生社交媒体应用却很少变化。

业务目标是一种更好的分类标准。最终，IT 组织要做的就是交付商业价值。那么，不考虑速度或者需求变更频率的话，还有比需要交付的商业价值更好的分类标准吗？

要分析优化与创新的区别，还可以从风险角度进行审视。提供核心业务服务的应用风险承受能力较弱，因此需要归类于优化。另一方面，创新型应用具有较强的抗风险能力。为了进行实验，他们的交付方式是允许失败的。

分析“核心业务”与“边缘创新”这两种模式时，可以用美式橄榄球运动来做比喻。在大多数连续比赛的体育运动中(比如足球、篮球、排球、曲棍球等)，同一组队员在进攻和防守模式间切换。但橄榄球比赛却有所不同，当球队由进攻转为防守时，场上会换上完全不同的两组阵容，反之亦然。这些不同的球员打法不同、教练不同、训练模式不同，尤其是遵循的指导手册不同。的确，在大多数体育运动中，都有同时负责进攻与防守的专业队员，只有在美式橄榄球运动中，进攻和防守是不同的队员负责。

1. 核心业务

隶属于核心业务分类范畴中的应用是组织中的那些关键系统。这些关键系统能保持业务持续发展，用他们自己的话说，能让灯一直亮着。这里所提交的业务模式都是易于理解的。用户参与的方法也很好掌握。因为已存在有一段时间，所以这些应用通常是大型的而且复杂程度高。他们具有某些特征以使得彼此之间在性质上类似。这些特征包括：

- 业务模式以及用户参与模式都易于理解，尽管具体的需求可能会不断完善或更替
- 通常按照稳定、明确的节奏，以能够让业务适应的更新速度进行提交
- 稳定性及在线时间是评价成功的衡量标准

- 因为存在已有一段时间，通常采用单块架构
- 所使用的技术各不相同，随着技术的进步，新的应用采用新的技术，但原有的技术仍然保留并继续提供运维支持
- 这些模式通常分别拥有开发及运维团队，而且有正式的运维交接程序，除此之外，运维还控制应用的运行
- 运维在这里绝对权威，IT 服务管理及信息技术基础设施库(ITIL)驱动日常运行
- 如果生产环境中发生问题，运维团队负责恢复应用
- 环境性质多样，混合着各种技术与平台，包括组织可能想要停止使用的遗留系统
- 优化是一场实现敏捷、速度及效率的比赛

2. 边缘创新

组成边缘创新的应用曾经或正在被开发用以驱动创新，通过尝试新的客户参与模式，利用新的技术及现代应用交付架构去体验新的业务模式，占领新的市场。这些应用是组织的发展动力，通常是小型的而且以现代技术及平台为基础。利用精益创业的技术方法，比如最小化可行产品(Minimum Viable Product，MVP)以及 A/B 测试，对业务模式进行尝试，基于实时用户反馈，利用设计思维去提升用户体验及开发用户参与模式，即持续创新。这些应用同样具有某些特征以使得彼此之间在性质上类似，其中包括：

- 业务模式及用户参与模式不易理解，需求改进要通过实验及快速获取用户反馈来实现
- 持续交付以利用诸如 A/B 测试这样的技术去尝试新的特性及客户体验
- 评价快速实现变更以及响应用户反馈的能力
- 通常利用多种编程语言构建，根据功能需要选择适合的语言
- 使用云原生及微服务这类的现代架构
- 开发团队绝对权威
- 目标是利用 PaaS 抽离出开发人员对基础设施的关注，使用 PaaS,

开发人员可以利用云服务去完成非功能性需求(Nonfunctional Requirement，NFR)

- 开发人员不仅负责构建应用，而且还负责运维应用。如果应用宕机，他们会负责修复
- 创新就像一场竞赛，组织尝试新创意、新特性以及新业务模式

决定应用归类的绝不是技术。分类标准应该仅限于业务目标，而在决定业务目标时，还应该对应用的风险与商业价值进行评估。一个应用如果提供高度规范的系统，并伴随着较高的价值与风险，就通常可以归类为核心业务，即使这个应用是利用微服务架构在 PaaS 上交付，并且利用诸如 Node.js 及 Go 这类的现代语言编写的。相反，用 Java 编写的服务可以作为创新型应用的其中一个组件，这些应用被设计成最小可行产品去验证新的业务模式，并且通常属于边缘创新。

边缘创新之所以被称为边缘，是因为在典型的大型组织中，创新只在“边缘”出现。事实上，很多组织可能只有一个创新点。这是因为 IT 组织通常把大部分预算用于运行现有系统，而非创新。正因为如此，优化核心业务对驱动创新至关重要，因为可以释放资源并投资创新，而且优化可以确保核心业务中那些运行缓慢、缺乏弹性的系统不会阻碍高速创新能力的发展。

在美式橄榄球运动中，要充分了解如何训练同一球队中进攻和防守阵容上的队员，而教练需要为进攻和防守队员分别开发不同的战术，IT 行业也是如此，组织要将其交付的应用及系统归类至核心业务或边缘创新，然后基于每个应用类别的需要来组建相应团队。平台与环境也要与应用类型合理匹配。如前所述，核心业务类别具有各种各样的环境，从传统的 IT 系统到云，从大型机到移动端，其目标就是优化环境与平台。对于边缘创新，基于平台的环境则是必不可少的。原生云应用能够为实验提供最快的速度以及最大的灵活性，还能保证所需的创新速度。熟悉这些现代架构以及相关现代开发实践的团队不可或缺。下一章将讨论平台即服务(PaaS)、容器以及利用微服务开发的原生云。

交付业务能力的这两类应用实际上是密切关联的，认识到这一点至关重要。 除了没有预设核心系统的启动之外，边缘创新上交付的系统

依赖于核心业务所提供的服务去交付开发的创新功能。如前所述，因为在典型的大型组织中，核心业务通常消耗大部分的预算，所以需要进行优化以释放资源用于创新。

以典型的移动应用为例(比如手机银行)，要交付这款应用，需要有一个负责前端开发的团队。这类应用都在边缘创新开发及交付。当使用这款应用时，是否所有的数据与业务逻辑都在手机上运行？当然不是。更多情况下，业务逻辑及数据只是核心业务中的预设系统或服务所交付的应用上的部分组件。所以，移动应用是由两个方面运行的组件混合构成的。即使只是交付移动应用的部分实验特性，公司也要结合边缘创新及核心业务所交付的组件来发布复合的应用。

所以，核心业务与边缘创新并非两个完全独立、拥有各自团队的同层仓筒。这会改变实施 DevOps 的真实目的。即使在美式橄榄球运动中，进攻和防守阵容终究还是一个团队，拥有同一个目标，而且需要互相成就。他们总是会有一个主导一切的总教练、一个负责业务运营的总经理以及一个不断赢得比赛的团队目标。与之类似，在核心业务与边缘创新模式中，二者之间存在着真正的关联性。他们相互依存、拥有共同的总体业务目标并且需要不断调整运行模式以互相匹配。

有些人试图发现这项运动的奥秘，但是美式橄榄球只有拦截与抢断这两件事情，别无其他。

—— 文斯·隆巴迪，美国橄榄球运动员及传奇教练

4.2 核心主题

一系列源于精益的核心主题构成本书中列举的多数 DevOps 方案的基础。在讨论方案本身之前，有必要先理解这些主题并认可其作为 DevOps 核心的重要性。如前所述，这些主题也是相互交织在一起的。

它们就是 DevOps 的“拦截与抢断”，具体包括：

- 最小化周期时间
- 缩小批次规模
- 建设正确文化理念

4.2.1　缩短周期时间

缩短训练“周期时间”

国际女子职业网球协会能够在赛事中向教练员提供实时深度分析，使得教练可以在比赛过程中与球员实时互动，而不必等到赛后再讨论这些深度分析与策略。

教练员手里有包含着赢得比赛所需分析内容的平板电脑，还可以把这些信息带到场上与队员一起讨论。

应用程序提供赛程趋势，可以显示发球的速度与方向、击球速度、击球动作模式以及场地范围。教练及球员能够看到比赛实况、场上的成功与失败等，所有这些都在比赛过程中实时发生以助于改变比赛成绩。

—— Donato, 2016

对于软件交付而言，除了生产中运行的代码之外，没有其他可以衡量真正进程的标准。这并不是指所交付的全部软件应用，反而是构建最终产品的小部分能力。软件的持续交付提供了相应的机会以从真正的客户，有时是客户代理那里获取反馈，这些客户或者客户代理使用软件产品并提供相应的反馈。这些反馈可用于三个方面的改进(如第 2 章所述)：

(1) 交付的软件

(2) 软件交付环境

(3) 软件交付流程

如图 4-1 所示，持续的交付会带来持续的反馈，持续的反馈会产生持续的改善。如此一来，尽可能缩短反馈周期便成为焦点，即在最短的时间内把反馈数据和分析结果提供给操作人员，以便其对软件、环境以及交付程序实施快速改进。

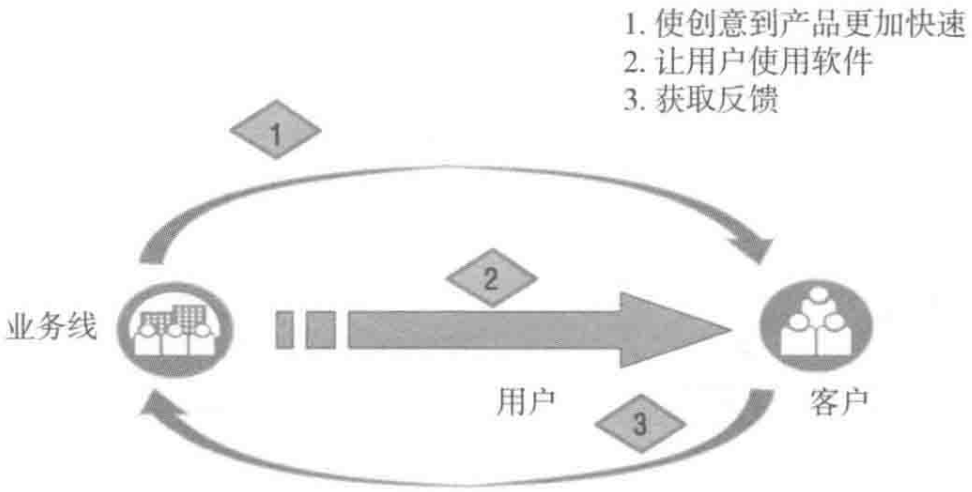


图 4-1　缩短周期时间驱动快速反馈

无论是全新的组件、现有组件的增强版本，还是一套组件的漏洞修复，每个软件组件从启动到生产中运行的代码以及反馈意见从用户到团队所用的时间被定义为交付周期时间(也可以称作开发周期)。交付周期时间将各种其他周期时间包含在内。

- 开发测试周期指的是开发人员从测试团队获得新交付组件的测试结果所用的时间
- 部署周期指的是把应用部署到相应环境中并开始利用其进行测试或生产所用的时间
- 运维周期时间指的是运维团队收到新环境请求，提供相应环境并使其可供需求团队利用所用的时间
- 数据中心延迟周期指的是从远程数据中心返回 Ping 响应的时间
- 项目审批周期指的是各种审批董事会及委员会批准新项目所用的时间
- 变更管理周期指的是变更控制委员会批准变更所用的时间
- 财务审批周期指的是首席财务官批准财务请求所用的时间
- 采购周期指的是采购办公室批准采购请求所用的时间
- 管理审批周期指的是管理层审批各项事务所用的时间

目标就是要缩短交付周期时间。而要实现这样的目标，唯一的方法就是要缩短每一项内嵌的周期时间，即上面列举的那些以及更多其他项构成整个交付流水线的周期时间。

缩短周期时间是 DevOps 的核心目标。使利益相关者之间的互动更高效、更省时是一种精益原则。在本书中所描述的几乎所有 DevOps 实施方案中，缩短周期时间是贯穿始终的基础主题。

- 交付构建物的速度有多快？
- 项目成员响应新需求的速度有多快？
- 分派任务完成的速度有多快？
- 应用部署的速度有多快？
- 新的项目成员掌握项目能力的速度有多快？
- 提供及配置环境的速度有多快？
- 安全变更审批的速度有多快？
- 开发环境中缺陷再现的速度有多快？
- 从生产数据中获取测试数据的速度有多快？
- 需求审批或拒绝的速度有多快？
- 进行复查的速度有多快？
- 出具状态报告的速度有多快？
- 应用回滚的速度有多快？
- 事件重建的速度有多快？
- 故障解决的速度有多快？
- 客户的反馈应用到新生需求或是需求改进的速度有多快？

缩短周期时间要求所有的 DevOps 原则都发挥作用，通过减少浪费及加强合作实现流程精益化。实现流程精益化并减少浪费，进而达到缩短流程周期目标的关键是缩小批次规模，接下来要讨论的就是这个问题。

4.2.2　缩小批次规模

如何通过 1%的绩效提升实现奥运金牌

戴夫·布雷尔斯福德先生于 2002 年出任英国自行车队总监时，这个团队几乎没有成功的纪录，英国自行车队在 76 年的历史中只赢得过一枚金牌。在戴夫先生的领导下，情况迅速发生改变。2008 年的北京奥运会，他的车队赢得了场地自行车赛十枚金牌中的七枚，在四年后的伦

> 敦奥运会上又取得了同样的成绩。戴夫现领导着英国有史以来第一支专业自行车队，这支车队已经赢得了最后四项环法自行车赛事中的三项比赛。
>
> 戴夫，一位获得 MBA 学位的前专业自行车选手，将边际收益理论应用到了自行车赛事当中，他孤注一掷，认为如果团队能打破常规，便能更深入地思考自行车竞赛的相关问题，对每个元素进行 1%的改进，其综合业绩便能突飞猛进。
>
> —— Harrell, 2015

要实现流程的精益、高效，增量交付必不可少。以工厂的装配生产线为例(之前用过的比喻)，装配线要实现速度与质量目标，就要持续交付相同的组件。每个组件的尺寸越小、相似度越高，生产或装配线的运行速度越快；流程也会更加高效，因为这些流程是由很多不断重复的小步骤组成的。如果工厂的操作工人或者机械手臂要做的只是重复添加同样的铆钉，其工作效率便可实现最大化。在需要执行多重复杂步骤的情况下，则刚好相反。每个步骤所承担的工作量就代表批次规模。要获得最大产出、实现周期时间最小化，就需要合理管理批次规模。

再回过头来看看精益的基本原则，其主要应用于三个方面(Reinertsen, 2009)：

(1) 库存最小化

(2) 积压明确管理

(3) 根据团队能力匹配批次规模

通过传统工厂、物流以及后台程序中所实施的精益流程，这些原则已被应用多年。而今，这些原则通过敏捷与 DevOps 被应用于软件交付领域。敏捷方法已经解决了库存最小化和积压管理的问题，所有的敏捷方法论都指向清理积压以及将工作项(软件交付库存)映射至敏捷迭代。DevOps 通过管理批次规模解决第三方面的问题，首先做的就是将批次规模与团队能力相匹配。在敏捷方法论中，批次规模与团队能力的匹配通过了解团队速度，即每一次敏捷迭代中团队所能够提交的工作单元来实现。基于团队规模、团队成员技能以及历史交付情况，可以相当准确

地判断出针对特定类型工作项目的团队速度。团队在固定数量的敏捷迭代中所能完全交付的工作项将决定理想的批次规模。要实现最大产出，批次规模需要最小化并且与团队速度相匹配。批次太大或太小都会降低团队的工作效率。

除了提升团队效率，缩小批次规模还有其他的益处：

- **缩短周期时间**。交付的产品越小，交付及获取反馈速度就越快。
- **降低交付风险**。引入变化量越小，影响力越低，引入变化的风险也就越低。
- **降低集成风险**。如果是微小变化间持续集成然后测试，集成问题就能更快识别及处理，因为都是些微小变化所造成的问题。
- **降低架构复杂性**。提交小批次变化的需要迫使应用改变架构以便于按小批次进行开发及交付。由此产生的是小型的高度解藕组件而非大型的整体组件所构成的架构。
- **提高测试质量**。测试团队对小批次变化进行更频繁的测试，而不是只能对大型的复杂变化进行较低频率的反复测试，产品质量也会随之提高。这使得问题及缺陷可以被快速识别并减少对小批次变化所造成的影响。这点适用于所有类型的测试，包括功能测试、集成测试、性能测试以及安全测试。
- **减少过度开发**。提交的变化越小，获取反馈速度越快，也就能更快确定所提交变化的正确性。
- **减少浪费**。对比大型复杂的变化，小批次变化相关的浪费更容易识别及减少。大型复杂的变化还会削弱直接识别浪费产生原因的能力。
- **降低可见性复杂度**。更容易看清交付流水线上各个团队或组件的真实状态，是因为批次里的每组工作项都是由几个小型任务组成的，这些小型任务使构件物发生很小变化，也因此易于跟踪及管理。
- **改善环境**。环境的变化越小，变化管理越容易。更容易识别出问题及事件发生的根本原因，是因为这些原因都可以归结为一小部分的变化。

- **改善流程**。流程的小规模增量变化相对容易实现。因引入新流程所导致的效率下降得以缓解。还可以更明显以及更容易地识别变化所产生的影响。
- **改善文档**。大型的复杂变化的文档难于交付，而且通常是在所有变化完成之后才能提交。通过缩小批次规模，可以在每个批次提交少量变化，这就使得变化的持续记录变得更加容易。
- **实现持续改善**。根据定义，只有通过不断进行小规模的循序渐进的改善，才能实现持续的改善。作为改善的核心，DevOps 的三个方面，即软件、环境以及流程都需要进行改善。本章的后面部分会详细讨论持续改善问题。

要实现小批次的目标并不简单。需要克服人员、流程、架构以及工具各个方面的挑战。

- **人员**：谈及缩小批次规模，解决人员问题最为关键的是处理好团队的规模与架构。传统的团队架构是为了大型项目而设计的，拥有大型的功能团队，他们从事自身职能范围内的系列工作，然后再交接给从事下一系列工作的功能团队。要将批次缩小至最优规模，就需要组建跨职能团队，从事贯穿于从需求到生产全过程的全部工作。本章后面部分以及后面的章节中将讨论各种不同的团队模式。
- **流程**：谈及缩小批次规模，有两种方式来处理流程。
  - 首先，该如何构建各种流程用以处理小规模的、频繁的开发、测试与组件交付？

    要把流程及相关治理机制拆分成小型的任务，以便操作人员能够独立操作，然后再交接给团队中具备处理后续任务所需相关技能的操作人员。
  - 其次，如何对流程本身进行持续的、循序渐进的改善？相对于大型的彻底的改变，对流程进行小规模改进更加容易。如前所述，引进变化会导致工作效率的下降(烟斗曲线)，引进的变化越小，工作效率下降幅度越小。
- **架构**：应用的架构必须可以拆分成能够进行开发、测试以及独

立部署的小型组件。要将现有的应用重构成这样的架构并非易事。关于这点，在下一章有关微服务的部分将详细进行讨论。不是基于微服务的架构也是可以按照小批次工作的，但是如果是对大型代码库进行循序渐进的改变，则需要整体交付。团队的组织结构与软件架构之间密切相关。本章后面部分将讨论反映这种关系的康威定律(Conw's law)(Conway,1967)。

- **工具**：尽可能减少交付流程中的非自动化步骤，将交付流水线的工具链结合起来，使用高度集成的工具处理小批次需求的快速交付。这是实现所需的精益、高效生产能力的唯一途径。工具的作用是使流程自动化。相应地，利用手工操作所不具备的工具能力，流程会变得更加高效。工具擅长处理自动化、重复性任务并使流程更加规范化。

最后，是向客户发布的整套发行版本。尽管交付团队可能从小批次交付开始，进而实现持续交付，但沿用这种思维模式去频繁发布变化较小的新版本，对终端用户却可能不适用。终端用户可能并没准备好使用频繁发布的小批次变化或更新。在无法向用户频繁发布新版本的情况下，就需要在预生产环境实施小批次变更。预生产阶段的交付可以进行测试，可以让客户做好准备，然后再正式以用户可接受的方式，按较低的频率向客户发布更新。本章后面在“实施发布管理”部分将对发布管理的流程进行讨论。

4.2.3　建设正确文化理念

球队的化学反应

对于我而言，化学反应是比赛最重要的方面。化学反应剧烈的球队会赢得冠军，再伟大的球队，缺少化学反应也会失利。这就是骑士队今年有机会获胜的原因。

湖人队深受众人喜爱。然而，由于傲慢以及缺少化学反应，使得他们有时候看起来很糟糕。骑士队在才智方面无法与湖人队相匹敌，但化学反应将把他们推至巅峰。

——Roberts，2009

康威定律：团队与架构

当梅尔文·康威 1967 年首次向哈佛商业评论(Harvard Business Review，HBR)提交论文提出其定律时，基于证据不足而被拒绝了。然而，弗雷德·布鲁克斯和其他几位专家给予了其所需的支持，使其赢得相应的关注。认定一项定律需要确切的证据，对于确切证据分量的界定决定了定律是否成立。并没有这种程度的确切证据来证明康威定律。然而，在软件领域却有充足证据启发性地揭示软件架构的本质并将其映射到软件交付团队的组织结构中，以此推论康威定律的确存在。

康威定律：任何组织在设计一套系统(广义概念上的系统)时，所交付的设计在结构上与组织的沟通结构保持一致。(Conway，1967)

从 DevOps 团队及组织的角度而言，康威定律通常用来作为团队重组的基础，使得流程、应用及系统架构更加高效。如果能够通过精益、高效管理以及移除障碍与仓筒，使团队的沟通与协作结构更加高效，那么依照康威定律，这些变化将会反映在团队所交付的应用及系统的设计与架构上。与之相反，而且大多数情况下是反模式的，如果执行的是为了交付小批次变化而专门设计的固定架构模式，而且团队可以围绕此类架构的应用与系统的需求进行自组织，那么依照康威定律，由此产生的团队模式就是能够实现最高效沟通与协作的正确组织模式。为最佳沟通与协作而设计的团队模式以及这样的团队所交付用以实现最大产出的架构模式，这二者连锁产生的两个新兴领域是针对团队模式的“跨功能团队”(squads)与针对架构模式的“十二要素应用”(12-factor apps)。下一章将对这两个领域进行深入探讨。

DevOps 作为文化运动

DevOps 首先是一场文化变革，始于文化运动并将其作为核心贯穿始终。如果参与人员不能彻底改变其组织、沟通及协作的方式，那么几分钟内可配置完成的所有改进流程、工具自动化以及云计算环境都无法实现 DevOps 的目标以及承诺，直到他们打破那些阻碍效率及产

出最大化、降低信任度的组织与文化仓筒。而且他们还必须克服组织的文化惰性。

要克服文化惰性，需要组织认同并有意愿去改变组织的各个层面，而非只是操作人员层面。

- **组织**：在组织层面，需要高层管理人员认同并主导组织变革，包括：设立 DevOps 卓越中心、提供 DevOps 启动资金、创建 DevOps 教练团队以及投资交付流水线的工具、平台与环境，发起对传统流程与管理体制的变革，甚至为了促进组织仓筒的最小化而重组团队的汇报架构。后面的章节中将详细介绍高层管理人员该如何领导变革推动者，以使 DevOps 在大型组织中得以大范围应用。
- **团队**：任何一款规模与复杂程度适当的应用，从项目计划开始，到需求、架构与设计、开发与测试、运维、安全以及风险管理，其间每一件事情都是由团队来完成的。团队通常依据功能被组建成功能仓筒。DevOps 核心原则之一就是要打破这些仓筒，建设跨职能领域的团队成员间的信任、沟通与协作的文化氛围，使所有为项目或应用开发与交付付出努力的操作人员成为促进项目成功的“利益相关者”。他们享受相应的权利，同时要承担相应责任，要确保应用能够按照设计初衷交付商业价值，而不是仅仅负责成功完成自己职能领域或仓筒内的工作任务。

从团队组建的角度已提出几种团队模式来实现这一目标。从在 Netflix 网站上流行开来的 NoOps 到声田(Spotify)所推广的称为“跨功能团队”完全独立的跨职能团队，所有这些模式都是用来打破仓筒并使团队成为一个整体组织。这些团队的结构中包括负责每个领域工作的利益相关者，与其他团队的职责边界也是不固定的。第 6 章将讨论各类团队模式及其优势与劣势。有一点必须了解的是，不管团队以何种方式组建，通过彻底重组将运维人员纳入跨职能团队或是创建跨职能团队矩阵，如果还保留着传统汇报结构，目标就是移除那些阻碍团队成员实现合作、沟通效率最大化的组织障碍。

- 个人：个人可以是最终的推动者，也可能成为最大的瓶颈。个人只有具备正确的意图，并且也更重要的是愿意去改变，才能进行变革所必需的那些改变。最终，即使团队组织良好，能够实现信任、协作与沟通，如果某些人选择不参与或者无法克服文化惰性，那么改变也不可能发生。

多么巧合！你离开时我们发现了瓶颈的根源。

——愤怒的老板对员工如是说

对人们的评价方式(以及回报)决定着其行为表现。因此，组织进行变革，对团队中个人的衡量、评价以及奖励方式也要随之改变，这点至关重要。要真正创建权责共享的文化，需要对所有相关个人采用相同的成功衡量标准。不能太过关注个人绩效的评价与奖励，也不要期望个人会把团队利益置于个人利益之上。在团队运动项目中，奖杯或金牌是给整个团队的，而非实现最多目标或得分的个人。的确，他们或许是最有价值队员(Most Valuable Player，MVP)，或者是最佳队员奖得主，但这些都无法取代主体团队的奖励。

奥运会中没有最有价值队员，团队运动项目中也没有个人贡献奖。团队中的每位队员都获得相同的奖牌。2016 年奥运会足球决赛中，巴西足球明星内马尔队长进球得分所获得的金牌与其他团队成员的金牌是一样的。团队中的替补队员也是同样情形。在奥运会中，基于真正的团队精神，只要替补队员在整个奥运会期间上场并至少打过一场比赛，即使只是资格赛而非决赛，也会获得团队所获得的奖牌。

无论如何安排整个组织，各个团队，甚至个人的行为模式，只有将人(团队)、流程以及自动化相结合才能使 DevOps 的真正潜力得以发挥。如果遵循效率低下的流程，严格管理负担过重，不具备启动及推广自动化的合适工具，或者要在遗留环境中交付应用，即使是组织良好、高度协作的团队也终将失败。真正的变革，正如 DevOps 实施一样，要求进行三个方面的变革：人、流程以及工具。不分主次。不能按顺序实施，

也不能忽略其中任何一项。当然，团队是变革的焦点，他们是工作的执行者以及变革的推动者。流程指导团队如何工作。工具使流程可复制、可伸缩、无误差。

4.3　DevOps 实施方案

现在开始讨论实施方案。有一点要特别注意的是，尽管这些方案以顺序的方式呈现，但并不需要按照这个顺序来实施。大多数可以同时采用。有些方案依赖于其他方案，这种依赖关系是不言而喻的，但如有必要，也会特别指出。

> 特殊的投篮或运球方式可以成为球员的个人标签，但是行动效率却是球队获胜的关键。
>
> ——帕特·莱利，前 NBA 球员、教练以及球队总经理

4.3.1　方案：建设衡量标准与关键绩效指标

如第 3 章内容所述，要识别出所需的正确 DevOps 方案，需要做到以下几点：

- 明确界定目标状态(业务目标及驱动力)
- 理解现状(能力及成熟度)
- 识别交付流水线中效率低下领域的瓶颈问题(通过实施价值流程图)

这三个需求代表了需要实现的优化目标以及实现目标所受到的限制。

同任何一项优化运动一样，首先需要评估现时的生产率，然后制定目标生产率。生产率的定义以及目标提升空间因组织间所受限制条件的差异而各不相同。因此，首先要做的就是确定衡量标准。哪些关

键绩效指标需要调整？正是用于优化交付流水线的这些关键绩效指标需要改进。

因为这些关键绩效指标将决定并衡量改进的内容与方式，所以这个方案应该最先执行。这是实施其他方案的前提条件，正如在运动赛事中需要首先了解如何赢得比赛一样。

永远别把忙碌与成就混为一谈。

——约翰·伍登，大学篮球队员及教练

下面介绍大多数组织在实施 DevOps 时通常采用哪些关键绩效指标。

1. 项目关键绩效指标

有两个核心的项目关键绩效指标：

- 速度
- 成本

的确，速度与成本或许是每个组织衡量及改进关键绩效指标的最低共同标准。速度的衡量标准有很多，下面是一些速度相关的关键绩效指标：

- 项目总工期
- 人/小时(月/年)
- 上线时间/价值实现时间
- 平均处理时间(修复)
- 试运行次数(针对边缘创新项目，详见第 5 章)
- 交付速度(每个版本交付的特性或用户故事的数量)

成本比较简单，但依据项目管理及所使用的评估办法，也有几种不同的衡量标准。具体如下：

- 项目总成本
- 利润值
- 成本绩效指数
- 成本差异
- 成本比率

- 每次部署成本(初次部署与重复部署)
- 每次问题修复成本/停机成本
- 客户获取成本与客户终身价值

已经实施 DevOps 的组织在速度与成本关键绩效指标上通常都做哪些改进呢？与 IBM 合作的大型通讯组织实施 DevOps 考察以下指标(kagan，2015)：

- 维护，小型项目(少于 15 人-年)
- 交付速度快于竞争项目 30%～40%
- 总成本降低 20%～25%
- 维护，中型项目(少于 100 人-年)
- 交付速度快于竞争项目 20%～30%
- 总成本降低 15%～20%
- 维护，大型项目(超过 100 人-年)
- 交付速度快于竞争项目 10%～15%
- 总成本降低 4%～8%

2. 关键绩效指标组合

考虑到项目之间的相互依赖性以及项目类型的不断变化，多数组织还应该衡量应用组合中这些相关领域的关键绩效指标。应用组合管理关键绩效指标包括：

- 整理组合中的项目，依据应用类型归类(大型机应用、布局应用、移动应用、原生云应用、打包应用、SaaS 应用等)
- 按记录系统与引入系统划分的应用组合
- 应用组合，依据架构类型划分(单体架构、SOA 架构、微服务架构、无服务器架构等)
- 组件和/或利用 API 架构点对点集成的应用之间相互依赖的数量
- 组合中代码和/或架构组件的重用
- 依照成本、风险及商业价值划分的应用分布
- 按维护模式、即将退役还是新开发状态划分的应用组合

3. 质量关键绩效指标

当进行质量评价，进而实施优化以提升质量的时候，选择适当的衡量标准至关重要。再次重申，合适的关键绩效指标因项目及行业的不同而异。制造医疗设备的组织，其质量目标要明显高于创建照片共享应用的组织，即便这些应用带有酷炫的滤镜。

质量关键绩效指标有两类功能性衡量标准：测试，这是应用开发流程的一部分；运维，用来评价生产环节中的质量。

针对测试的关键绩效指标，通常会提出以下 DevOps 相关问题：

- 多少比例测试(单元测试、功能测试、集成测试、性能测试及安全测试)是自动化的？
- 多少比例的服务和应用能够通过虚拟化或打桩方式自动测试？
- 多少比例的测试是在类生产环境中利用类生产数据进行的？
- 测试人员的时间有多少比例用于测试环境准备、配置以及应用部署这些非测试相关的任务上？
- 开发人员的时间有多少比例用于解决测试人员发现的缺陷？

典型的测试关键绩效指标涵盖多个方面的评价工作量与执行时间，比如：

- 探讨测试准备工作
- 准备测试数据
- 检查测试环境(冒烟测试)
- 审查测试准备程度
- 选择测试用例
- 执行测度用例
- 分析测试结果
- 创建缺陷
- 重新测试缺陷
- 准备测试总结报告
- 沟通测试总结报告

实施 DevOps 以后，很多方面的周期时间与工作量减少 40%～50% 的情况司空见惯(Kagan，2015)。

业务应该关注的核心衡量指标有以下几点，这些指标主要由运维团队负责评价：

- 严重程度 1、2 级事故的数量
- 严重程度 1、2 级事故的平均处理时间
- 严重程度 1、2 级事故的平均成本

DevOps 致力于减少这些 1、2 级事故的发生以及事故发生后的平均处理时间，最终减少这类事故的成本费用(Quirk,2004)。DevOps 通过持续监控来观察生产环节中发生的错误及问题的类型与根源，以实现此目标。这类关键绩效指标包括：

- 软件失效
- 应用失效
- 数据错误
- 数据传输错误
- 架构引起的问题
- 服务告警状态/停止状态
- 高使用率
- 配置错误

在运维方面，针对 IBM 客户所进行的有关实施质量的改进包括以下几点(Kagan，2015)：

- 宕机和应用性能下降方面减少 50%
- 可用性提升 60%～90%
- 应用问题根本原因的分析速度提高 90%

4. 交付流水线优化关键绩效指标

浪费(MUDA)

Muda 是一个日本术语，指的是宝贵资源的浪费。丰田生产方式创始人大野耐一(Taiichi Ohno)认为，浪费分为以下 7 种：

- 运输

- 存货
- 请求
- 等待
- 过度生产
- 过度加工
- 缺陷

——Pereira, 2009

DevOps 的最终目标是减少浪费以优化整条交付流水线，即从构思(创意阶段)到生产环境的全过程。要实现这样的目标，需要有相应的关键绩效指标，能够用以评价应用交付流程的复杂程度并进行简化。下面是一些很好的用以评价交付流水线复杂程度的关键绩效指标样例。这些关键绩效指标即可以评价初次部署，也可以用来评价应用的重复部署：

- 每个交付周期的成本
- 交付周期的持续时间(前置时间)
- 交付周期中审批步骤的数量
- 交付周期中管理评审(检验)次数
- 通过不同的评审步骤(安全、法律及法规；企业架构；费用控制委员会；标准委员会等)执行审批的非项目团队利益相关者的数量

下面是一些可以用来进行项目层面的评价以决定流程改进的关键绩效指标：

- 项目启动时间
- 梳理出的待办事项(针对敏捷项目)
- 总体开发时间
- 复合建构时间
- 迭代测试时间
- 冒烟测试(Build Verification Test，BVT)可用性
- 总体部署时间

- 总体生产时间
- 发布时间间隔
- 运维人员用在开发与维护上的时间比例

5. 文化关键绩效指标

一个人可以是一个团体的关键要素，但一个人永远不能成为一个团队。

——卡里姆·阿布杜尔·贾巴尔，前篮球明星

曾经有人建议，找到合适的关键绩效指标来评价文化的人应该同时获得诺贝尔经济学奖和诺贝尔和平奖(因为阻止了关于文化评价标准的战争)。该如何评价文化惰性？又要如何去衡量文化是否有所改进？常见的方式是衡量团队士气。团队士气是否有所提升？但是直接从文化变革到提升士气是很难实现的。事实上，即使组织拥有杰出的组织文化，团队仍会士气低落。想想创业公司酷炫的开放式办公计划、喝酒游戏、午间按摩以及荒唐的股票期权。然而，如果公司的技术无法实现预期的成果，那么任何文化都无法挽救；士气也会随着股价的下跌而日渐低落。

审视 DevOps 的首要原则，即创建信任、协作与沟通的文化，可以衡量出这些方面是否有所改进。如果跨仓筒的 DevOps 预沟通只需要通过任务单，或者运维人员平均每天花费超过一个小时的时间参加状态会议，或者随时更新状态报告，那么文化改进就已经实现了。评价文化，所选择的评价标准要能够衡量出团队内部以及各团队之间发生了多少合作与沟通。

这里是一些可衡量的关键绩效指标，这些指标间接表明了文化的成熟度：

- 不需要修改或返工，就可以为操作人员制定所用的构建物比例(完成比率及准确度)
- 操作人员用于参加会议与从事生产性工作的时间比例

- 操作人员在不可见的等待状态中，等待他人对需求做出反应的时间比例
- 操作人员之间非实时沟通(例如电子邮件与信息)的比例
- 操作人员创建与更新的对最终交付不能产生附加价值的构建物数量
- 在状态会议以外，操作人员需要每周互动的其他功能领域的人员数量
- 不需要管理层参与，团队可以自行决策的程度(团队授权)
- 通过会议或状态报告进行汇报与利用仪表盘汇报的比例
- 项目衡量标准及关键绩效指标通过仪表盘向操作人员公示
- 团队成员认为的其个人的贡献与更广泛的业务及组织目标的契合程度
- 操作人员流失率
- 操作人员对公司的知识产权(Intellectual Property，IP)和/或开源项目所做的贡献(取得的成绩)

有趣的是，这里的关键点是团队授权与对知识产权或开源项目的贡献。这两项衡量指标在当今世界至关重要，尤其是对于拥有千禧一代成员的团队而言。对于操作人员能够感受并建立多少自尊与自信，这两个指标是很好的衡量标准。

如果把文化单纯理解为人的行为与互动，那么确保团队拥有正确的人终将成为最重要的衡量标准以及正确文化的根源。团队成员是否富有成效？他们是否快乐？他们是否甘愿承担更多职责，以使团队取得成功？他们是对团队士气与文化产生了积极影响还是导致了文化惰性？采取措施确保团队拥有正确的人，这方面最极端的案例当属亚马逊及其子公司 Zappos。这些公司会对由于不开心而离开的人给予补偿。亚马逊对其执行中心的雇员每年提出一次邀约，如果因为工作不开心而选择离职，公司可以支付 2000 到 5000 美元的补偿。Zappos 在严格苛刻、深入的新员工培训结束之后也采取同样的措施。他们宁愿付钱让这些人辞职，也不想他们留在团队里危害公司以及文化。(Taylor, 2014)

4.3.2　方案：敏捷实施

> 我认为，在比赛开始之前，用类似“我们真的需要赢，否则就出局了”这样的表达来激励团队不失为一个好主意。然而结果恰恰相反，这样做并没有帮助他们表现更好。最初我也觉得奇怪，现在我明白了。问题在于球员没有机会凭借其个人力量影响比赛的结果。即使球员本身已做到最好，但还有太多无法控制的其他因素会决定比赛的结果。缺少控制使他们紧张焦虑，从而阻碍最优水平的发挥。我意识到输赢只不过是我们比赛方式的结果。好消息是每个球员都**能够**影响其个人的比赛方式。所以，比赛前不再谈论输赢，取而代之的是仔细重复我们的战略以及每个球员必须注意的个人事项。这是具体的，也更好控制。通过这个方法，他们更加放松，进而呈现个人最佳状态。我们的结果证明了这一点。
>
> ——Marc Lammers，曲棍球球员和主教练，荷兰(Peter，2008)

大多数组织把已经实施敏捷的项目作为实施 DevOps 的试点。这本来就是 DevOps 实施的先决条件。有些人甚至争论说，DevOps 将敏捷实施一方面延伸到项目管理、项目立项以及项目的实施阶段，另一方面则扩展至运维。

DevOps 始于敏捷，伴随这一观点而来的问题是，执行瀑布式开发流程的团队能否实施 DevOps。当然，在瀑布方式下，不存在持续集成与持续交付的实践，因为没有什么是持续发生的。如果事情只发生一次，在定义上就不能称为“持续”。尽管确实不可能实施所有的 DevOps 实践，也当然无法在使用瀑布式应用交付程序时获得 DevOps 的所有益处，但是仍可以从个别几个 DevOps 实践中获益，这些实践甚至可以使基于瀑布式流程的交付流水线更加高效。这类实践包括：

- **自动化部署**。手工部署是在浪费时间与精力，而且容易出错。即使并非持续部署，自动化部署在节约时间以及提升部署质量方面也能产生巨大价值。
- **进入类生产环境**。开发与测试人员工作的开发-测试环境与实际运行环境不一致的话，会降低质量并产生返工。提供所有应用交付

状态下类生产环境的访问权限，便可以很容易地解决这个问题。

- **自动化测试**。再次重申，自动化提升速度、减少错误。
- **监控与反馈**。向业务线及开发人员提供生产绩效评价标准有利于提高其能力去改进应用及运行环境。

以上这些以及 DevOps 架构中的其他实践，还有本书中所讨论的实践，都不需要明确应用交付流程是敏捷的还是瀑布式的，任何项目都可以实施。

从流程角度而言，敏捷项目最适合 DevOps 实施，这是不争的事实。除此之外，由于敏捷的局限性，这些项目成为滋养 DevOps 实施的土壤。敏捷的主要局限在于其范围仅限于开发测试。的确，敏捷包括开发测试周期中的业务，但业务的作用仅限于参与每一次迭代的开发测试以确保所开发产品的正确性。Scrum 以及大规模敏捷框架(Scaled Agile Framework，SAFe)这类的敏捷方法并不会改变业务线的运行方式以使其更加敏捷，也不会扩展到运维以使其更加敏捷。利用其余按瀑布式运行的交付流水线，运维使用定义好的固定窗口期，或通过手工任务单系统参与开发测试，这个方法就会导致第 2 章中所提及的瀑布式敏捷开发(water-Scrum-fall)。

在这类瀑布式敏捷开发环境中，敏捷团队很快就会达到生产力极限。随着速度的加快，如果执行输入的利益相关者速度跟不上，或者使用输出的利益相关者不能与其速度相匹配，那么阻抗失衡会限制其速度与敏捷性，如图 4-2 所示。这成为其实施 DevOps 强有力的原因，以保证交付流水线的速度与敏捷性、消灭失衡。

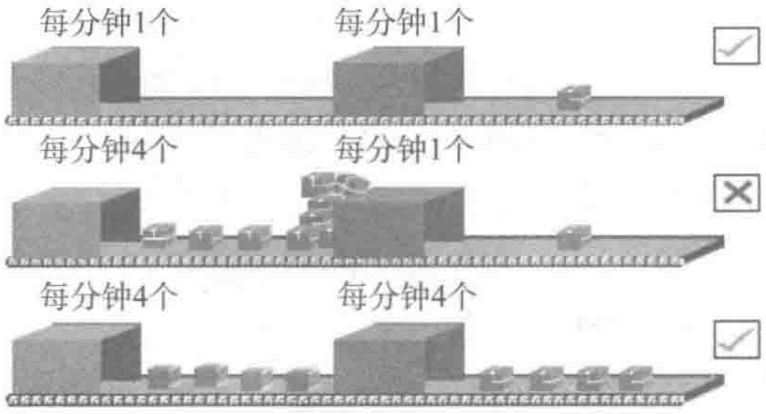


图 4-2 解决瀑布式敏捷开发导致的阻抗失衡

为消除阻抗失衡所做的工作也同时清除了一系列障碍，或者按本书此前的说法，称为瓶颈，这种情况并不少见。例如：

- 如果开发人员执行的是持续集成，每天工作结束时都做好测试准备，但测试组织无法按这个速度进行测试，这个问题就需要解决
- 如果开发测试团队每两周运行一次开发迭代，而运维团队却需要三周的时间来准备新的测试服务器，这个问题就需要解决
- 如果开发测试团队可以持续运行，能够通过每两周一次迭代快速交付代码，而业务分析人员却只能每季度一次性提交大批量的新用户故事，这个速度要低于交付速度，这个问题就需要解决
- 如果安全团队只能在新的应用上运行安全测试，每个发布周期只能运行一次，而且每次测试需要 5～10 天的时间，这个问题就需要解决
- 如果集成构建需要数个小时才能完成是因为每个组件及模块都需要构建，即使这些模块和组件根本没发生变化，那么这个问题就需要解决
- 如果业务线期望项目按确定的按部就班的时间表依照计划进行，而事实上他们的需求并未被充分理解，项目计划就需要调整。

这些瓶颈是可以事先识别的，也可以通过第 2 章提及的价值流程图制定缓解计划来解决。

敏捷团队不断碰壁，这使得 DevOps 实施得以最大限度地开发这些团队的潜力以及敏捷本身的潜力。他们需要引进 DevOps 实践以将敏捷性在交付生命周期中从左至右全方位扩展。通过把 DevOps 实践带给生命周期前端的利益相关者，利用诸如敏捷产品管理、设计思维以及精益创业这类的技术与实践，DevOps 将敏捷性向右扩展。向右扩展敏捷性是 DevOps 的灵魂所在。将开发与运维拉近，便形成了 DevOps 的起源(以及名字由来)。这样使得持续交付、基础设施即代码、软件定义环境以及持续反馈这类的实践得以应用。接下来的方案中将讨论所有这些实践

与技术，从中可以发现他们在从构思到生产的整条交付流水线中如何驱动敏捷与高效。

所以，从 DevOps 方案的角度而言，敏捷方案的实施有以下两种方式：

(1) 敏捷团队：如果有已经实施了敏捷的团队或者项目，那么找出足够成熟以至于可以开始突破生产力极限的团队，其生产力达到极限是因为瀑布式敏捷开发所带来的各种限制。如图 4-3 所示，这些团队为推进 DevOps 的实施已准备就绪。

(2) 非敏捷团队：如本章前面内容所述，即使这些团队不打算改用敏捷开发方法，他们通过实施相应的 DevOps 实践也一定会获益匪浅。

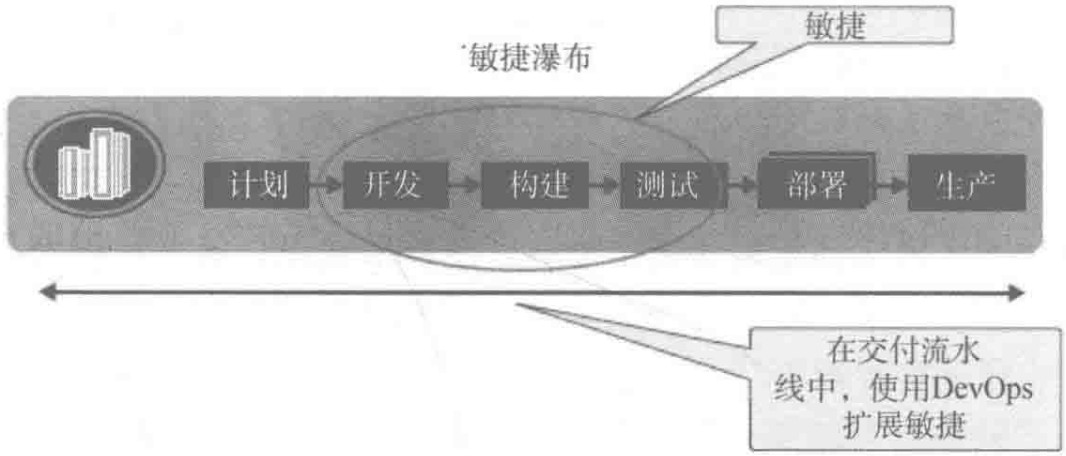


图 4-3 实现整个交付流水线的敏捷性

4.3.3 方案：集成的交付流水线

要实现整个交付流水线的效率最大化，就需要消除或者最大限度地减少流水线中那些影响到每一个利益相关者的瓶颈及低效。大多数瓶颈问题都发生在跨职能领域的利益相关者之间在互动过程中的接触点以及他们传输构建物的时候。手工任务是导致效率低下的首要原因。工具对于流程自动化、缩短等待时间以及减少人工失误必不可少。话虽如此，如果不能彼此集成，工具反而会增加复杂性和低效性。Pre-DevOps 致力于优化交付流水线的各个职能领域。每个职能团队为其职能领域寻找最佳的工具。然而，如同利益相关者一样，工具不可能真空运行，构建物不是利用工具创建的，就是从其他利益相关者团队所使用的工具输入

的。利用自己选择的工具集，创建或修正的构建物需要传输给其他的利益相关者团队。如果工具无法集成，不能创建无缝连接的工具链，那么在传递过程中的各个接触点会出现明显的效率低下。

操作人员从其他利益相关者团队收到的构建物格式用自己的工具无法识别，这种情况并不少见。操作人员获得一份文档，之后还需要手工输入至自己及团队所使用的工具，例如，从数据分析团队获取一个数据模型，在导入到应用架构工具之前，需要先转换成 XML 或者 CVS(Comma-Separated Values，字符分隔值)。再比如，开发人员利用 Jetty 或 Tomcat 服务器小程序容器(servlet container)创建代码，之后还要再部署到 WebSphere 应用服务器。所有这些情况都会产生或手动或自动的额外工作，从而导致效率低下。

真正的解决办法就是集成工具链。每个构建物配置单一数据源，即单一存储库，任何有权限的利益相关者都可以访问。不管是通过成熟的点对点工具集成，还是利用构建物的标准化格式进行，构建物可以在工具之间无缝传输，这就是工具链。已经完成大量的工作去构建工具数据库之间的数据级链接，提供数据存储级集成，这样就不再需要在工具间来回移动构建物。开放式生命周期协作服务(Open Services for Lifecycle Collaboration，OSLC)开放标准社群就致力于解决数据级链接的标准问题。

图 4-4 是交付流水线的样例。事实上，交付流水线的范围更加宽泛，还包括其他的利益相关者及其所使用的工具，比如需求管理、架构、应用设计、项目管理、安全、发布管理等。当然，实际情况中的利益相关者会因组织甚至项目的差异而各不相同。所需要的环境也会有所不同。举例来说，针对功能、集成、性能、压力、系统集成测试(System Integration Testing，SIT)、用户验收测试(User Acceptance Testing，UAT)以及安全测试，有多种不同的测试及测试环境。还会有多重部署环境。一些组织既有预生产环境，又有后期生产环境。针对这些环境，还会有相应的监控和管理工具。工具及环境的数量必须在合理范围内，但如前所述，更重要的是要将工具集成为工具链。



图 4-4 集成交付流水线

1. 实现端到端的可追溯性

集成交付流水线的其中一个重要优势就是可以实现构建物之间、交付流水线之间的端到端追溯。这种端到端的可追溯性促进了单一数据源的产生，以供所有跨职能领域的操作人员与利益相关者使用。端到端的可追溯性最明显的益处就是赋予操作人员及利益相关者相应的可见性能力，使其能够了解构建物之间的关系和访问正确版本构建物的权限，满足工作任务所需。

图 4-5 显示了缺陷相关的变更请求端到端可追溯性。举个典型的例子，开发人员更改代码以满足新的变更请求，将变更集交付至集成流并创建持续集成构建，然后将持续集成构建提交给测试人员，他们运行测试套件，其中一个测试脚本失败，缺陷被创建用于记录失败。在没有端

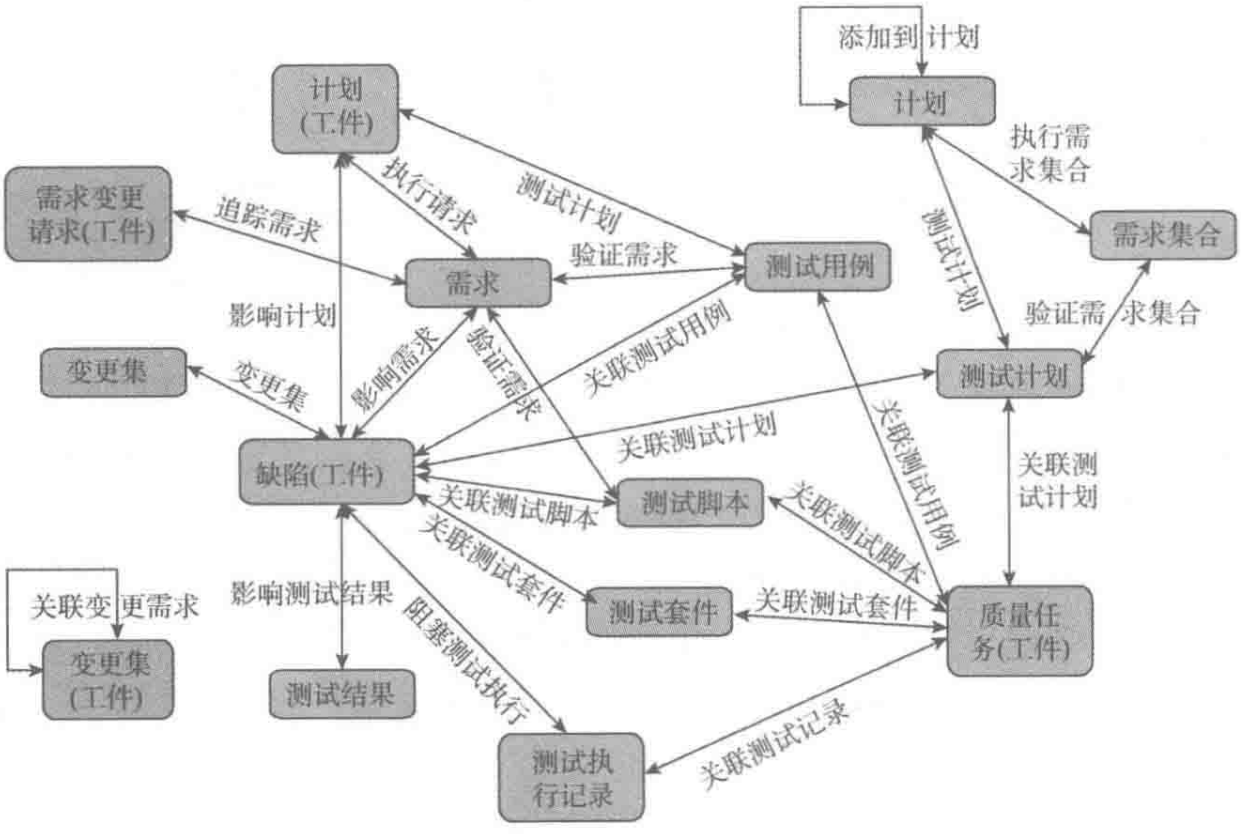


图 4-5 缺陷的端到端追溯

到端追溯的情况下，多个开发与测试人员要解决这样的缺陷问题，就不得不来回多次以识别测试所运行的变更集并妥善区分代码中导致缺陷的根本原因，这种情况并不少见。如果要进行适当的影响分析并估算减少这种缺陷所需的工作量，那就更麻烦了。有了端到端追溯，所有的构建物都可以相互关联，从而使缺陷识别、影响分析以及实际的解决方案更加高效。

让我们浏览一下可追溯性图表(注意此图表只是完整的端到端可追溯模型的一个子集)。

- 通过运行测试脚本发现缺陷
- 测试脚本是测试套件的一部分
- 测试套件代表测试计划
- 为了验证一组需求而制定测试计划
- 失效的测试脚本有相关的测试执行记录
- 失效的测试脚本有测试结果
- 测试结果用以创建缺陷
- 缺陷可以追溯至代码变更集，这是持续集成构建中最后一组代码变更
- 变更集用以满足变更请求
- 变更请求影响着一组需求

没有贯穿整个交付流水线的集成工具链，这种程度的追溯几乎是不可能实现的。

除了可见性以外，端到端追溯的益处还包括以下方面：

- **影响分析**。通过评估变化及缺陷产生等方面的影响分析结果，利益相关者可以降低风险
- **变更管理**。利用变更集以及工作项，可追溯性使构建物的所有变化得以被追溯，从而能够对构建物进行合适的变更管理
- **减少过度生产**。可追溯特性能够识别出无用代码、无法追溯至活动需求的代码以及不能追踪需求的测试用例等
- **操作人员效率**。操作人员能够确保其所处理构建物的正确性，并且可以对工作进行上下游追溯，不需要执行手工任务

- **测试覆盖率**。确保所有的需求得以测试，而且所有测试测的都是活动需求
- **状态报告**。端到端的可追溯性实现了状态报告的自动化，不再需要更多的电子表格去关联跨职能团队的状态报告
- **合规性及可审计性**。端到端的可追溯性使审计轨迹得以创建，即构建物、利益相关者以及交付流水线每一次变化发生时的人物、内容、原因以及时间

端到端的可追溯性并不容易实现。构建集成交付流水线时，应该力求确保构建物维持适当的可追溯性，这些构建物在添加新的集成之前是不可追溯的。

2. 带有多重交付流水线的多级 IT

关于“多级 IT”这个术语，业界已反复进行了许多探讨。组织应用的分类源于 Gartner 分析公司在论文中所提及的“两级 IT”。但这种说法很快被“双模式 IT”所取代，打破了所有事物只按快慢进行分类的观念。双模式 IT 对应用的分类是基于其需求的稳定性，还有在开发中的进化或实验属性。双模式似乎也无法完全满足行业所需的分类模式要求，现已被“多级”的概念取代，因为只用两种模式去区分所有的 IT 工作及项目是不可能的。现存事物实际上都是统一连续的，特别是涉及速度因素时。

而今，业界已达成共识，所以这些模式(双模式、多级以及任何可能出现的新模式)都能及时发布。全美互惠保险公司的 Carmen DeArdo 早就提出未来的世界将是“变速”的。每一项应用都将依据业务意图及目标去选择其速度与模式。

就如同不会根据速度，也不会因为司机每天上下班都走这条路或是因为这是司机首次去一个新的目的地而指定高速公路的车道一样，应用交付的速度或模式也不需要指定。交付团队会根据业务需要及团队成熟度去选择相应的速度与模式。刚取得驾照的青少年不会(也不应该)在高速公路上以最快速度驾驶。同样，救护车去急救时也不必注意限速。

多级背后的核心驱动是不可能只有一个交付流水线的事实(除非处

于启动阶段，只有一个应用)，如图 4-6 所示，将会存在多条流水线。基于技术堆栈、所使用的开发语言、交付环境、操作人员成熟度、不同操作人员的地域分布、核心业务与边缘创新的业务意图，应用的风险价值评估，当然还有业务所需速度的差异，这些交付流水线也不尽相同。所有这些因素，加上其他更多的方面，使得这些交付流水线以不同的速度移动。现存的以多重速度运行的交付流水线称作“多级 IT”。这些交付流水线不是独立运行的，而是彼此之间相互依存，产生实施多级 IT 需要解决的挑战。第 6 章将详细讨论这种相互依存关系以及在企业范围内推广 DevOps 实施所必需的多级 IT 处理方式。

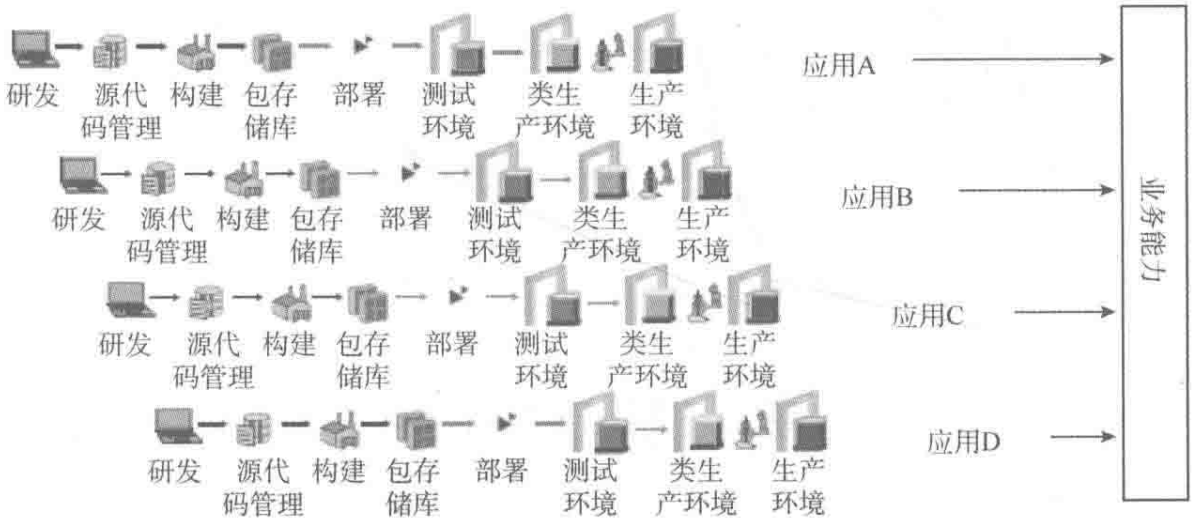


图 4-6　带有多重交付流水线的多级 IT

从构建集成交付流水线的角度而言，多级 IT 意味着需要有多个这样的集成交付流水线。这些流水线或许都拥有不同的工具，与交付流水线平台、技术以及交付速度实现最佳匹配。重要的是这些流水线都是横向集成的，但这并不意味着混乱无序以及每个交付流水线拥有独一无二的工具组合，即使可以集成。标准化是必需的。工具标准化带来流程与操作人员能力的标准化，使其可以在拥有相同工具链的几个交付流水线间互换。标准化还有助于创建定义明确的评价标准体系，以实现持续改进。这个方案的最终目标是为每一个技术领域创建一条至多两条集成交付流水线。COBOL/CICS、Java、.NET、iOS/Android 移动应用、认知应用、大数据、区块链应用，每一项可以有 1-2 个的标准工具链。一个组

织中不能有太多这样的交付流水线工具链，否则会造成混乱。

没有标准，就没有改善(Kaizen)[①]。

——大野耐一，丰田生产系统创始人

除了交付流水线内的集成需要，多重交付流水线间还存在着垂直集成的需要。这些集成以及相应的工具标准化在五个具体的领域势在必行：

- **计划**：通过分析彼此之间的相互依存关系，协调交付流水线的工作计划与项目计划
- **架构与 API(应用编程接口)**：通过建立良好架构的应用接触点，利用 API 来实现尽可能地解耦依赖项
- **部署自动化与编排**：根据需要提供、编排以及部署环境、中间件与应用的能力
- **服务虚拟化与测试环境**：在依赖的所有环境、服务以及应用都不具备时，测试应用或服务的能力
- **发布管理**：执行发布计划，即预防和处理资源冲突、延迟以及集成挑战

这些集成还使多重交付流水线间的可追溯性得以实现。之后的方案中会详细讨论多重交付流水线间的这些集成点。

4.3.4 方案：持续集成

Sprint(敏捷迭代)是一个 Scrum(迭代式敏捷软件开发)术语。在 IT 行业许多不成功的命名中，这是其中一个。在田径或游泳比赛中，冲刺(Sprint)意味着运动员在临近终点时以其最快的速度进行的短距离、快速跑(或者游泳)。而且，赛跑中的冲刺是有固定距离的，在这个距离内，参赛者尽可能比竞争对手跑得更快。另一方面，Scrum 中的敏捷迭代通常也是为了固定时间周期的迭代而构建，比如两或三周的敏捷迭代，在

---

① Kaizen(改善)：日本持续改进工作实践、个人效率等的企业经营理念。

此期间，团队基于自身的速度，尽可能完成更多的工作单元。同 Scrum 一样，速度(velocity)也是敏捷方法的一个术语，用来衡量团队在一次敏捷迭代中通常可以交付多少工作单元。

事实上，开发与应用交付更像是一场马拉松，而非冲刺。同马拉松运动员一样，团队需要建立一种节奏，据以完成交付工作单元的固定节奏。团队之后还要对这种节奏进行持续优化以实现其速度最大化。马拉松运动员在训练过程中找到其最优的跑步节奏，通常每分钟跑 160～180 步才能坚持跑完全程。同样，应用交付团队的成员在实施持续集成与持续交付时也要找到交付的节奏。不需要耗尽能量拼命前行，也不需要英勇的付出去推进项目完成，而是要以可行且可以承受的节奏让团队坚持到整个项目周期结束。

持续集成(CI)已作为一项实践在第 2 章有详细介绍，不仅介绍，而且还深入探讨了其核心特征。创建持续集成实践是必要的，但一经建立，每个团队就必须要建立相应的持续集成节奏。持续集成节奏指的是构建的固定频率，开发人员将据此交付测试并可能最终发布给运维人员。这种节奏为所有团队设定了期望。典型的持续集成节奏是每日构建以实现面向生产的持续交付。这并不意味着每天都要进行生产交付，这对大多数企业而言都是不合理的，但是至少要每天进行测试环境的交付。

每个组织的持续集成节奏各不相同。按日进行并非唯一选择，但节奏一定要稳定，即使时间周期较长。另外，组织内不同项目团队的持续集成节奏也可能有所差异。基于流程、工具以及团队合作的舒适程度，团队自行决定其集成节奏，而这种节奏又会反过来决定其速度。要实现持续集成，团队至少应该每天进行构建，而且构建应该自动化而且快速进行，集成测试的反馈周期时间也要短。如果构建需要数小时，测试周期又是数天，那么持续集成节奏就无法保持稳定，就没有节奏可言。

实现持续集成是每一位开发人员的责任。每一位开发人员都必须按照团队采用的持续集成节奏将代码交付至集成流。如果无法按照至少每天一次的节奏将代码交付至每日构建中(最低限度)，那么其代码将不会被纳入持续集成构建，从而破坏整个团队的持续集成。

> 成功的每日构建是软件项目的心跳。缺少成功的每日构建，就失去了心跳，项目就会死掉。
>
> ——吉姆 • 麦卡锡，微软 Visual C++ 产品经理

持续集成的实现要求团队及项目要有良好的分支方案，使开发人员得以独立工作，在多个工作项上同时创建多版本代码，并允许其轻松地将代码交付至集成流。关于分支方案，即开发人员究竟该交付至主线还是交付至专用集成分支(本书中称为“集成流”)，存在着很多的原理。团队应该依据开发人员、团队规模以及团队速度来选择适合其组件尺寸需要的一种。要避免分支策略或源代码管理工具，对开发人员同时开发多版本代码能力和从任何分支向集成流交付代码能力的限制。

构建持续集成工具是为了支持团队范围内的持续集成节奏。像 Jenkins、UrbanCode Build、TeamCity 还有 Travis CI 这类的现代持续集成工具都有能力定期查询源代码管理系统(SCM)，使开发人员可以校验其交付至集成流的代码。一旦开发人员交付了代码，工具就会启动，生成代码并集成到集成流中已有代码的其余部分中。当然，如果构建中断，添加代码的开发人员需要立即得到通知，以缩短反馈周期时间。如果构建成功，就会进入适当运行及集成测试阶段。这些测试需要自动化进行。每次运行至关重要的关键测试是集成测试，以验证和识别集成错误。如果集成测试通过，诸如每天结束时进行的关键构建(每日构建)应该被提升并部署到测试环境来运行其他测试，即功能、安全、性能测试等，然后再向更高级的环境部署。如此一来，持续集成工具就需要与自动化部署工具进行集成，以实现持续交付，下一个方案中将讨论这部分内容。

如前面的集成交付流水线方案中所述，典型的组织不会只有一个交付流水线，而是有多个相互依存的流水线。同样地，多数项目团队不会单独开发与交付代码。多个团队可能处理同时生成的代码以交付单个的组件。反过来，所开发的组件与其他组件之间也是相互依存的。而且，几个这样的组件会共同交付服务或应用。如此一来，持续集成至少发生在两个层级(详见图 4-7)：

(1) 组件级的持续集成。在同一组件上工作的多个个人或团队需要持续不断地将自己开发的单个代码交付到集成流，并启动集成构建。这类构建最好在每次团队成员向集成流交付新代码时进行，或者至少每天进行一次。

(2) 应用级的持续集成。所有交付应用的组件通过集成构建统一集成。这类构建没有组件层面构建进行得频繁，但是也必须至少每天一次。

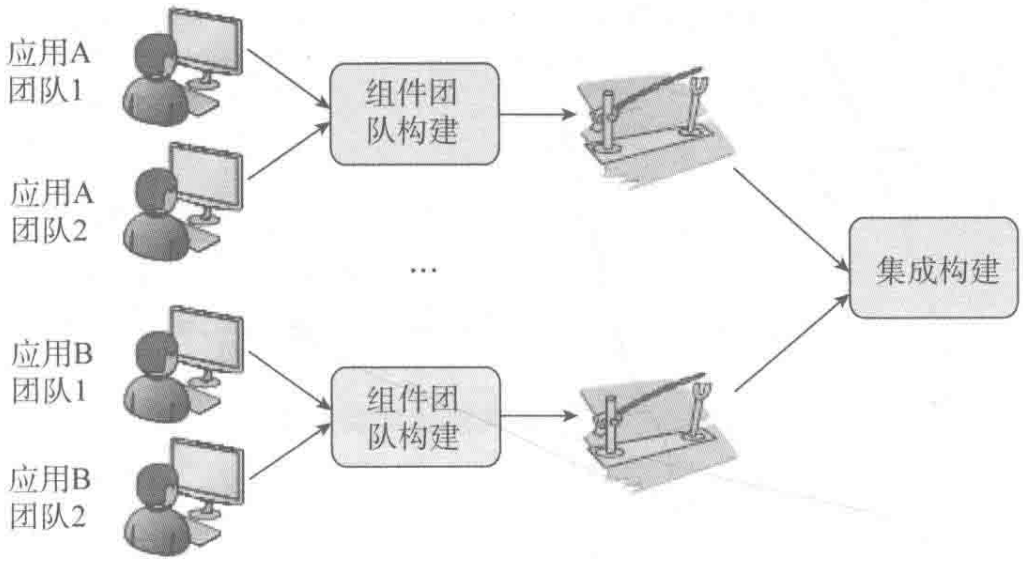


图 4-7　组件及应用级的持续集成

还有系统级的集成，多个服务或应用一起集成。这个规模上的集成不是持续集成，但仍然尽可能频繁地进行。并不存在真正的构建，因为这些服务与应用都是自由存在及独立运行的。他们不需要与其他服务及应用进行集成。这种集成应该属于 API 级(之后的方案中会对 API 进行详细讨论)。

在所有这些层级上，每一次持续集成构建之后都应该进行集成测试，以在构建完成后识别集成错误与缺陷。只做持续集成构建，却不进行集成测试去验证及识别缺陷，只能说明具备持续集成构建的能力，除了持续构建再无其他能力。

随着组织在团队、项目及应用中实施持续集成，构建过程管理对于跨企业的推广开始变得必不可少。大多数持续集成工具可以覆盖拥有上百组件的超大型项目及应用。然而，每个项目执行自己的工具实例，组织内构建与持续集成工具杂乱无序随意扩展的情况并不少见。这就产生

了对标准化的挑战并导致构建质量出现较大偏差。标准化对于企业范围内的推广至关重要，关于这点将在第 6 章详述。

持续集成是 DevOps 实施的一个关键方案，可以驱动其他几项方案。就是持续集成将代码推进至交付流水线的持续交付与其他阶段(如图 4-8 所示)。持续集成节奏设定了代码流通过整个交付流水线的速度。正确的持续集成是 DevOps 的成功基础。

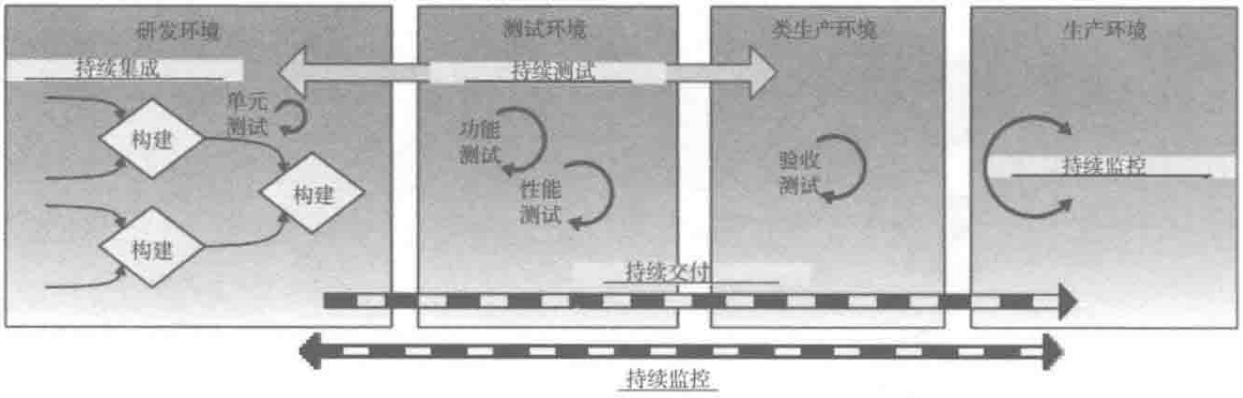


图 4-8 持续集成驱动交付流水线

4.3.5 方案：持续交付

> 这是一个循序渐进的过程，每周跑四天。长期来看，在训练第一周跑 6 英里是相对容易的。每个周末，这个距离会增加一些，在马拉松前三周达到 20 英里。赛前减量期使运动员得以为比赛积蓄能量。调整周避免运动员训练过度。交叉训练及充足的休息要混合进行。
>
> ——霍尔·希格登，美国作家、长跑运动员及教练(Higdon，2011)

和持续集成一样，持续交付也在第 2 章介绍过了，两个容易混淆的术语“持续交付”与“持续部署”的区别也解释清楚了。概括来说，持续交付是持续不断地将应用部署到交付流水线各种环境中的能力。为了实现可重复性、可靠性以及可伸缩性，部署过程必须自动化。

1. 自动化部署

开发人员开发代码时，需要部署到开发服务器以确保其能够运行。代码编写完成后，开发人员要为持续集成提交代码。持续集成构建需要

部署至测试服务器进行测试，并且升级至其他测试及测试环境，所有的步骤都是在非生产环境进行的。如此一来，持续交付就涵盖了多重环境下的多重部署，从代码模块的简单部署，到开发人员将其构建为可部署的二进制文件，再到多组件复杂的部署编排。代码实现集成并内置到组件、服务及应用中时，就不仅是需要部署到服务器的简单二进制文件。任何项目都可能进行部署，从简单的配置变化，到面向新特性的增量代码变化、数据库模型变化、环境变化，再到整个环境。

将这些组件部署到多个节点及中间件服务器已成为一项编排。不同组件的部署可能需要按照适当的顺序进行。进程或服务器可能需要停止，并开始执行代码模块或配置更改。数据库模型变化以及数据库配置都要进行管理。需要在服务器集群的所有实例中进行更新和应用的配置变化总会产生相应的中间件组件。这就要求自动化部署工具能够创建复杂的部署流程，并且可以编排以及更新多个中间件服务器与进程，如图 4-9 所示。

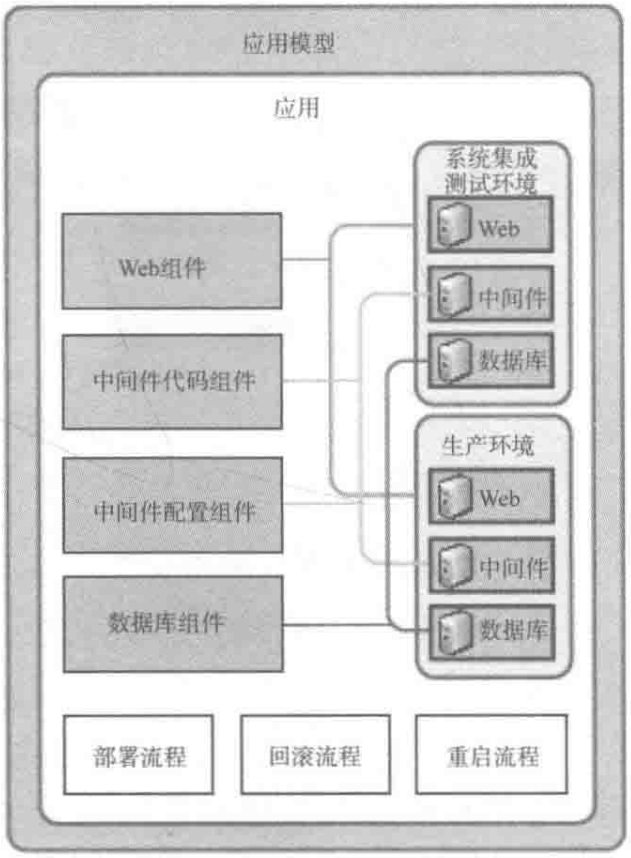


图 4-9　部署应用组件

除了部署流程之外，创建适当的回滚流程也至关重要，如图 4-10 所示。多种原因可能导致回滚需求的产生。首先，如果部署本身失败，那么无论部分部署了什么，都需要进行回滚。已完成部署的应用如果识别出关键缺陷也可能需要回滚。回滚流程很复杂，部分部署的应用(要追溯哪些已经部署，哪些还未进行部署)是这样，已完成部署的应用整体回滚亦是如此，其中数据库的变化是最大症结之一。有一种方法显示的是应用根本就没有回滚，只是向前推进。这种方法通过部署应用的旧

工作版本并以此替代需要回滚的应用来实现回滚的向前推进。数据库的状态及数据变化仍然是个问题，需要谨慎处理。

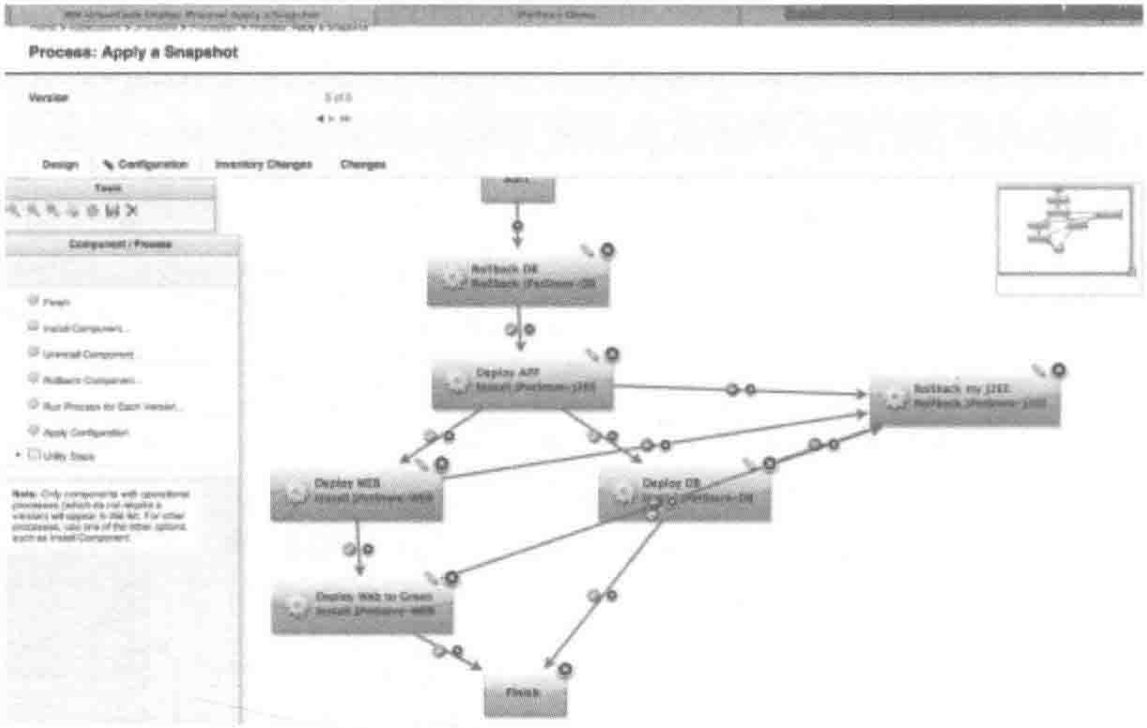


图 4-10 带回滚的部署流程(在 IBM 的 UrbanCode Deploy 工具中)

2. 数据库部署

每一项数据库所特有的具体挑战都会带来数据库组件的自动化部署：

- 数据库的更新必须是增量的。虽然应用组件可以完全由全新版本替代，但这对于数据库来说是不可行的
- 数据库的更新必须以有序方式进行。每一次改变都要基于此前的变化。举例来说，将多个应用 WAR 文件更新到应用服务器，可以同时，也可以按任意顺序进行。数据库的更新却只能按规定的顺序进行，以保证表中的终端数据集
- 错误地部署两次应用组件不是问题。然而，添加相同的数据库记录两次导致重复记录，却是不可接受的
- 数据库更改是不可逆的。应用部署可以回滚或覆盖。数据库要实现回滚，只能用备份整体替换

基于数据库部署的特殊需要，已开发出相应的工具。传统的数据库管理员使用的是需要手动管理而且容易出错的脚本。为了达到 DevOps

所需的规模与速度，自动化是必需的。Liquibase、dbdeploy 这类的开源工具以及 Datical 与 DBmaestro 的商业工具实现了持续交付的数据库部署自动化。

数据库部署还存在相关的文化挑战。数据库的更新由数据库管理员控制。在传统的非 DevOps 驱动的团队中，数据库管理员处在独立于开发测试的仓筒中。他们与开发人员没有联系，而且可能无法理解开发人员为实现持续交付而对数据库进行的快速更改。此外，他们还把自动化推回到没有工具控制的地方。随着集成交付流水线的向右延伸，这些集成交付流水线中持续交付工具链包含了数据库部署工具，挑战应运而生。为了应对这些挑战，应该使数据库管理员成为项目组中具有平等地位的利益相关者，消除其所在的独立仓筒。

3. 部署内容、方式以及环境

全能选手

在板球运动的历史上，很少人有足够的天赋能同时驾驭球和球棒。只有少数运动员可以声称属于这个特殊群体，而这些人中又只有一小部分会成为历史书中前所未有的伟大人物。

更为罕见的还是真正的全能选手，他们对球和球棒都能很好驾驭，不论是作为投球手还是作为击球手，都能成就团队。

—— Cheshire，2012

真正的部署需要自动化并且持续进行，谈及真正的部署，就需要理解部署的三个方面：

(1) 部署的是什么？

(2) 如何进行部署？

(3) 在哪里进行部署？

在统一建模语言的概念(UML，2005)中，图 4-11 的模型显示的是部署这三个方面之间的关系。这个模型展示了在 IBM UrbanCode Deploy Blueprint 中如何获取部署的这三个方面。其他部署工具使用的也是类似的概念。

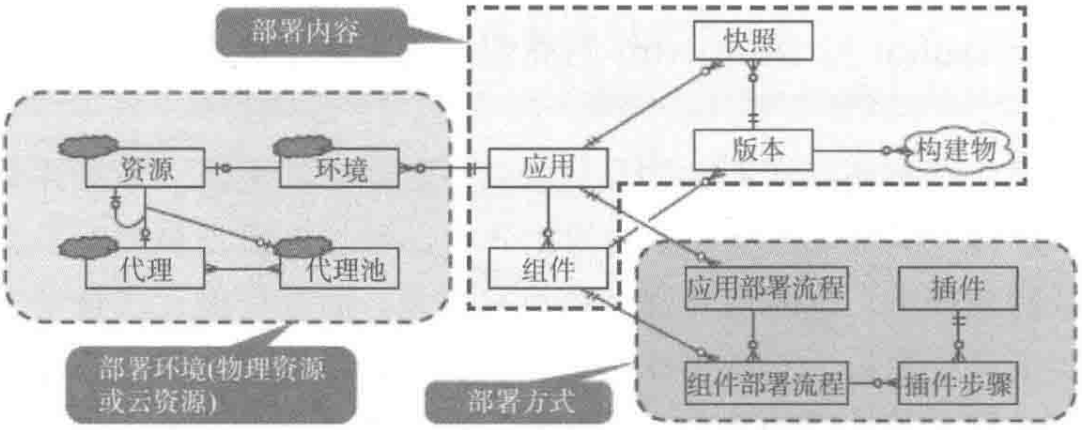


图 4-11 自动化部署“蓝图”

部署内容(什么)：部署内容指的是需要部署的构建物，其中包括代码、配置、数据库模型、数据、内容、网页等，如模型所述：

- 构建物指的是所有能够部署的、版本化的并且能存储在构建物库中的有形产物。
- 组件是构建物版本的集合
- 应用由组件构成
- 组件各版本一起组成快照版本，即一组需要互相部署的版本组件
- 所有的应用、快照、组件以及构建物都有相应的版本

因此，进行部署的时候，部署的内容可以是整个应用，也可以只是一些组件或是一个快照(获取一组组件版本，代表整个应用或只是其中一部分)。

部署方式(怎样)：部署任何一项内容，都需要执行一系列步骤，这些步骤构成部署流程，或称为“部署方式”(如何部署)。这些步骤可能是技术性的，需要进行编排，即根据步骤之间的逻辑按特定顺序进行。某些步骤可以与其他步骤同时执行，其他则可能有依赖项及先决条件。此外，流程中可能还有一些步骤是用来处理单个步骤或是整个过程中的失败情况，如模型所述：

- 应用具有部署流程
- 组成应用的每个组件也都有部署流程
- 应用的部署流程调用各个组件的部署流程
- 每个组件流程包含多个步骤，其中有一些或者可能全部都是由

技术性插件执行的技术性步骤

- 部署流程也可能取决于部署的目标环境(哪里)，是要将组件部署到应用服务器或是数据库，还是需要安装到操作系统级别才可执行
- 部署流程可能还包括对应用本身、中间件或者部署环境进行配置的步骤

执行的是整个应用的部署流程还是只是一个组件的部署流程，这取决于部署的内容是全部应用还是只是一些组件。

部署环境(哪里)：指的是部署的目标环境。每个部署环境都是由几个服务器所构成的，每个服务器上安装着不同的应用服务器及其他的中间件。作为部署流程的一部分，这些服务器及中间件可能需要更新配置，如模型所述：

- 环境由节点或资源组成
- 每项资源上都运行着一个代理，由其进行实际的部署以及管理库存
- 节点集可能属于代理池，即相同的节点集，需要部署同样的组件，进行同样的配置
- 应用组件与资源间存在着映射。这种映射告诉部署流程哪个应用组件需要部署到哪个节点
- 部署流程还指出每个节点需要管理的配置设置

应用部署环境有两类：

(1) 静态。静态环境指的是状态稳定的环境。静态环境中的节点(服务器)数量固定，节点之间有着预设的静态网络连接，形成静态拓扑。

将应用部署至静态环境时，部署流程唯一要做的只是管理各种服务器的配置，然后部署正确的应用组件到适当的服务器。

(2) 动态。顾名思义，动态环境不是静止不变的。基于影响状态的各种不同因素，节点的状态不断变化。根据需要来决定配置还是取消节点。尽管节点间的网络是预设的，拓扑在本质上却更加动态。

将应用部署至动态环境时，节点的配置并不依赖部署流程，而应用组件配置完成后只需要部署至节点即可。另外，基于嵌入至流程中的规

则与条件，应用部署流程本身可以提供节点。整个堆栈的节点与应用就是这样部署的，接下来的部分将对此进行详细讨论。

云托管环境的出现使动态环境成为常态。这些动态环境被设计成云模式。云模式的定义与标准多种多样，其中包括 OpenStack Heat Orchestration Template (HOT)，IBM Virtual System Pattern(vSys)，Amazon AWS CloudFormation template 以及 Docker Swarm。除了虚拟机映像(节点)及其网络连接(拓扑)的定义之外，模式还捕捉环境中的行为，即基于流程流以及模式中所包含的规则如何进行编排。

作为一种利用 IBM Urban Code Deploy 所设计的 OpenStack Heat 模板，图 4-12 显示的就是这样的模式。关于云环境以及持续交付中运行的角色模式，下一章将进行详细讨论。

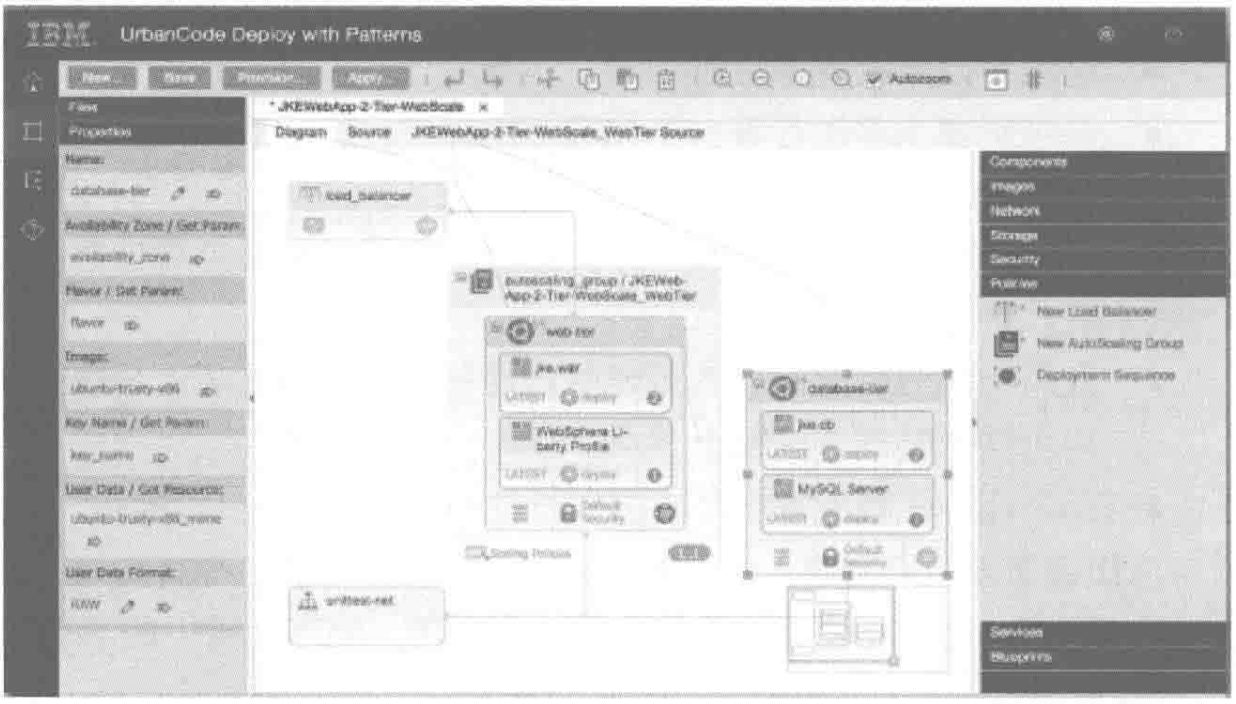


图 4-12 IBM UrbanCode Deploy's Blueprint Designer 中的 OpenStack Heat 模式

从将应用组件(是什么)映射到环境(到哪里)这个角度来看，这只是静态环境下的一个简单过程。对于动态环境，则需要对应架构进行水平扩展，以支持环境的动态扩展，即基于规模及功能的需要而进行弹性的容量的上下调整。

类生产环境 DevOps 的其中一个核心原则就是为开发测试人员提供类生产环境。交付流水线中的关键挑战，也是低效率的来源之一便是

开发与测试人员无法访问环境，即便这些环境与真正的生产环境并不类似。这就导致应用在最终部署的生产环境中的功能与行为与其在开发测试环境中的功能与行为完全不同。如果从未在类生产环境中进行过测试，那么实际的部署过程会有很多错误产生。

理想的状态是开发、测试及其他生产环境都与实际的生产环境完全一致。从经济学的角度而言，这是不可行的，因为所需的基础设施成本太高。下一个最好的解决方案是提供在功能及性能方面与生产环境极其类似的环境。或许无法具备与生产一样的伸缩性以及相同数量的计算节点、内存、存储空间或者网络带宽，但是，应该确保存在着相同的节点拓扑，使用的是相同版本的操作系统及中间件，而且操作系统与中间件所用的是类似的配置设置。

典型的反模式包括以下情况：

- 开发测试环境利用开源应用服务器，生产环境却利用商业应用服务器
- 不同环境使用不同版本的服务器操作系统
- 利用定制的、自行开发方式打桩(stub)来代替昂贵的测试服务，却不利用可以正确模拟这种服务的服务虚拟化
- 使用与生产数据模式不相类似的测试数据
- 不去管理较低配环境(测试环境)的中间件的配置，使之与生产配置相匹配
- 多个应用共享测试环境，导致配置及行为漂移

全栈部署　全栈部署流程究竟部署的是什么？先来看看全栈的构成，如图 4-13 所示。

- 应用(包括应用代码和应用配置)
- 数据
- 中间件配置
- 中间件
- 操作系统(OS)
- 虚拟化



图 4-13　全栈

- 服务器(计算节点)
- 存储
- 网络

谈及全栈部署，指的是部署所有这些层级。在非全栈部署中，所部署的只有较高层级部分，即应用数据、应用配置、应用以及中间件配置。其余低层级的内容是作为环境的一部分，在应用部署之前配置的。关于哪些部分是应用部署流程部署的，哪些是环境配置的，这二者的界定是主观的，而这取决于所部署的应用。很多情况下，应用部署流程包括了中间件本身的部署，环境的配置也是仅仅利用节点上的操作系统来实现。

4. 持续集成到持续交付

> 4×100 米接力赛既是技能比赛也是速度比赛。四个一般选手组成的队伍利用在接力区的优势击败速度更快的对手，就能跑赢四个较好选手所组成的队伍。这项比赛的关键是在接力区传递接力棒所花费的时间。
>
> —— 罗森鲍姆，未注明日期

在构建交付流水线工具链时，集成的关键点之一便是可部署的资产从构建和持续集成工具向持续交付工具的传递。持续集成的节奏应该驱动持续交付的节奏。此外，如果目标是要实现周期时间最小化，那么构建与交付流程都应该是快速的。理想状态下，从持续集成到持续交付应该是一个持续的、无缝的过程，而且周期时间最小化(详见图 4-14)。



图 4-14 持续集成到持续交付过程

多数情况下，在交付流水线中，除了构建工具以及自动化交付工具以外，还建议创建包或者构建物存储库。构建物存储库用来存储所有的可部署资产、配置及其相关依赖包。包仓库成为真理的唯一来源，每个利益相关者都可以在这里获取部署应用或组件所需的适当资产。存储库可以是顺序的，也可以与构建工具并行工作，使之成为交付工具集成点。

也就是说，包仓库可能位于两个工具之间，即构建工具将资产推送至存储库，部署工具从存储库提取出来(顺序方式)，也可能位于工具旁边，即构建工具每次将资产传递至自动化部署工具时都同时将其发布至存储库(并行方式)。

在持续交付周期中，构建物存储库只是一个通道。然而，当需要周期外的部署，比如需要重现缺陷时，构建物存储库就变得必不可少，以确保所部署的都是正确资产的正确版本。因为拥有所有可部署资产的所有版本，存储库还成为端到端追溯的必要驱动力。

从工具的角度来看，构建物存储库有很多种。有 JFrog Artifactory 或 Sonatype Nexus 这类的通用存储库管理工具。大多数自动化部署工具都有自己的内嵌的构建物仓库，仅用来存储可部署的资产。IBM UrbanCode Deploy 的内嵌仓库称为 CodeStation。嵌入式存储库与通用存储库同时使用的情况也不少见。内嵌的自动化部署仓库存储的是所有已部署资产的副本。

5. 推送与拉取

接力棒的三种传递方式

传递接力棒的三种方式各有利弊。德拉蒙德喜欢推入式，尽管并不总是最快，却通常被认为是最安全的方式。

推入式：接棒选手的手掌朝向跑过来的选手，跑过来的选手将接力棒垂直握着并直接推入接棒选手的手掌。

上曲式：接棒选手的手掌朝下，跑过来的选手将接力棒向上挥动至接棒选手的拇指与其他手指之间。

上手传递：接棒选手手掌朝上，跑过来的选手将接力棒放在接棒选手的手褶位置。

——乔恩·德拉蒙德美国奥运男子 4×100 米接力赛　金牌得主(2000)及教练(2012)　(Ward，2012)

从持续集成到持续交付，有两种传递方式：

(1) 拉取机制。这种方式下，自动化部署工具将可交付资产从构建

工具或者构建物存储库提拉出来。

(2) 推送机制。这种方式下，构建工具或者构建物存储库将资产推送至自动化部署工具。

执行持续交付时，推送是首选的操作方法。构建工具完成构建并推送，启动部署。拉取方式可能会存在问题，因为在这种模式下，自动化部署工具无从知晓构建是否完成，或者所有需要部署组件的可部署资产是否可用。这就导致只有部分资产完成部署。只有在部署资产已预先构建，甚至已提前部署(也就是说，不是在一个连续的交付周期中)的情况下，拉取方式才能很好地运行。因为只有构建工具知道构建流程什么时候完成，所以最好由构建工具在完成构建后的第一时间启动部署。

6. 实施持续交付

产品发布过程中会遭遇挑战的最主要原因之一就是，项目没有执行持续交付，应用首次部署到生产环境，就是在产品发布期间。实际部署应用的时候，各种组件的部署经常失败，或者遭遇挑战，是因为这是部署流程第一次在生产环境中实施，即便是那些自动化的部署流程也一样。它们从未经过真正的测试和验证。传统项目在发布软件的时候会有一个发布周末，现场的整个团队利用整个周末(或更长时间)去修复各种组件部署过程中产生的问题，在这里，以及本章中所讨论的其他几项挑战就是其原因所在。

在实践应用的持续交付时，不仅要验证所交付组件的功能与性能以及交付环境，还要验证组件的部署流程。软件的部署不像在 FTP 服务器上复制一些二进制文件那么简单。既涉及具有潜在复杂性的节点集群中多个路径的文件传输，也涉及操作系统、数据库以及中间件的配置更改。同时还会涉及步骤的编排。不能简单地以机械的、线性的方式执行部署步骤。中间件流程可能需要在配置更改后重新启动。服务可能需要在文件传输之前停止，然后重新启动，所有这些都要以协调有序、精心安排的方式进行。持续交付使这些流程得以进行测试和改进，从而确保进行最终的生产部署时，团队不是首次实施这些流程。随着多项部署在多重

周期或迭代中进行，可以对流程进行持续测试与调试，从而确保其在任何时候都可以良好运行。此外，这些部署流程还应该包括环境部署流程在内，而非只有应用部署流程。

这里的关键是要从项目一开始就启动持续交付，从迭代 0 开始一直贯穿整个项目。刚开始，部署可能很简单，但到了项目后期，会变得复杂得多，而且需要精心策划。持续交付将应用、中间件、配置、数据以及环境碎片化，利用适当的自动化工具，对自动化、部署流程、配置变化、部署环境，当然还有所部署的应用进行验证，以降低风险。

部署始于迭代 0。可是，迭代 0 能部署什么呢？连代码都没有。这个问题很简单，那就是部署环境。获取 Linux(如确有必要，也可以是 Windows)发行版本并安装好，然后在某处启动。

7. 持续交付平台

> 给我一个平台。让我们摇滚，摇滚吧，就在今天。
>
> ——杜威·费恩(杰克·布莱克饰)，《摇滚校园》，派拉蒙影业，2003

最终的目标是为操作人员提供一个平台，使其能够持续高效地将应用部署到开发、测试、系统集成测试、用户验收测试以及最终的生产环境中。该平台的目标是在基础设施和应用交付流水线功能之间引入一层抽象层。操作人员不需要担心基础设施的细微差别；硬件、系统管理程序、虚拟机、数据中心以及网络都会从操作人员那里抽象出来。他们需要专注于自己的目标，开发、测试并交付应用与服务，而不用去关心所配置的基础设施的种类与位置。操作人员的环境应该是定义明确的、架构良好的、无缝的、始终可用的、弹性的、可升级的环境，而且是根据应用与服务的具体需要调整好的。这种抽象或者发生在基础设施层，交付基础设施即服务(Infrastructure as a Service，IaaS)，或者发生在平台层，交付平台即服务(Platform as a Service，PaaS)，也可能通过利用容器，在交付的应用或服务与基础设施服务之间创建基于 API 的隔

离。这三种选择各有利弊，下一章将进行详细讨论。

无论是哪个层级的抽象，都有必要为应用交付操作人员创建相应的平台。该平台跨越交付流水线内的各种环境。这样的平台通过构建工具链完成，工具链提供了支持交付流水线所需的各种服务或功能。工具链通常包括提供以下功能的工具或服务：

(1) 源代码管理
(2) 构建
(3) 持续集成
(4) 自动化部署
(5) 中间件配置
(6) 环境配置
(7) 环境供应

还可以将其他几项服务添加到以上列表以形成完整的功能平台，但这些是创建功能平台的最低要求。如果正在进行全栈部署，包括了从应用到计算节点全部堆栈并作为一个流程配置的单一环境模式中会涵盖第 4～7 项功能。如果进行的不是全栈部署，那么每项功能都是单独的服务，而且不需要每次部署应用时都需要配置环境。但每次部署新版本的应用时，仍然需要配置中间件和环境。

实现这类平台的工具链如下：

- Git、GitHub、Subversion 或者 Rational Team Concert 这类的源代码管理工具
- Jenkins 或 IBM UrbanCode Build 这类的构建与持续集成工具
- IBM UrbanCode Build 或 XebiaLabs Deployit 这类的自动化部署与中间件配置工具
- Chef、Puppet 或 Salt 这类的环境管理工具
- VMware vRealize、IBM Cloud Orchestrator 或 Amazon Web Services EC2 这类的环境管理工具或环境供应商

利用标准化的、集成的工具链至关重要。为了确保可移植性和防止供应商锁定，这些工具应该基于开放的标准。

4.3.6　方案：测试前移

防守移位在棒球运动中如何发挥作用

防守移位的第一部分是显而易见的：三个内野手转身移向场地的一端。不太明显的部分是击球手如何被投中；投变速球或内快球，迫使击球手拉球。

如果击球手是极端易拉型，可以将那些快球投至任何位置，依靠击球手把球拉至过度防守。

——裁判，2016

在 DevOps 交付流水线中，测试前移促使测试在交付流水线中尽早进行。这适用于所有类型的测试：单元、功能、回归、集成、性能、压力、安全性等。通常情况下，除了由开发人员进行的单元测试外，其他所有类型的测试都在交付流水线的后期进行。有一些类型的测试，比如性能测试、负载及压力测试或者安全性测试，通常只在最终发布前才进行。在拥有多家供应商的大型项目中，甚至集成测试都会留到项目最后阶段进行的情况并不少见。测试的延迟使得缺陷与问题不能及时发现，从而带来高风险，并导致修复缺陷与问题的费用过高。相信大家都还记得 U.S.healthcare.gov 网站在 2013 年 10 月发布时所出现的问题。大多数问题都可以归因于测试在发布后期进行，从而导致关键缺陷在网站发布时未能及时发现(Pollock，2013)。

生命周期早期进行的测试前移具有以下两点优势：

(1) 较早进行测试意味着可以早些发现缺陷与问题，从而降低解决问题的成本以及整个项目的风险

(2) 较早进行测试意味着可以更频繁地进行，从而减少每次的变化幅度(小批次)。单次变化较小的高频率测试能够显著降低相关的风险，并提高交付的综合质量，因为交付的内容已经过更多的测试。

测试前移要求创建类生产环境，使更多的实际模拟测试得以在交付周期的早期进行。较低负载水平的服务虚拟化、生产数据仿真以及类生产环境下的负载模拟这类的技术常被用于类生产环境的创建。本模块后

面的部分将对服务虚拟化进行更详细的讨论。

1. 自动化测试与持续测试

> 障碍并非一定会使你停下来。如果你跑到了一堵墙边，不要转身也不要放弃。请找出办法去翻越它、穿过它或解决它。
>
> ——迈克尔·乔丹，篮球偶像

持续集成的目标是什么？是将开发人员的代码持续交付至生产吗？是的，最终是这样的。但是，在此之前，则是要实现代码的持续测试与验证。是要启动所需的测试，去验证代码是根据设计生产的，并且是按照设计与其他开发人员的代码集成，与应用功能及性能的其他组件集成。没有持续测试，持续集成与交付(几乎)都是毫无意义的。如果只有通过不满客户的投诉才能发现应用功能或性能低于标准，那么拥有最先进的持续交付流程还有什么用处呢？

在开发人员创建代码去添加新功能、加强现有功能或者解决缺陷问题时，他们持续不断地将自己的代码与那些开发完成并交付至集成流的其他组件代码进行集成。在这个过程中，他们在自己的代码上运行单元测试。集成完毕后，他们就在集成代码上进行集成测试。他们可能还会运行诸如白盒测试或者代码级安全测试、代码性能测试等的其他测试。这项工作之后会被交付至各个团队的共同集成区域，集成为项目服务的所有团队的工作，以及构成服务、应用或开发系统的所有代码组件。这里需要特别注意的是，持续集成流程之后紧接着要做的一定是去验证所有层级的集成代码准确无误，并且开发人员运行的所有测试都运行无误。因此，作为持续集成的一部分，持续测试实际上是开发人员开始的。

经过整套应用(或服务或系统)构建并验证无误后，应用会被交付至测试域。从开发人员或开发环境到测试环境的这种代码交付是持续交付中的首个主要步骤。当开发人员将其代码交付至团队的集成空间以及项目的集成空间时，持续交付就发生了，但这仅限于开发空间内，并不针对新的环境。当交付给测试时，才称之为从一个环境到另一个环境的完全过渡。测试会在其自己的类生产环境中运行一系列的功能及性能测

试。此外，测试有可能还需要在每次运行系列测试时添加新的数据集。这就意味着持续交付流程不仅要求从开发到测试的代码转换过程，还要包括为测试准备新的类生产环境的步骤，并利用正确的配置以及相关测试数据运行测试。这使得持续交付流程更加复杂，而并非仅仅是 FTP(文件传输协议)代码那么简单。关键是持续交付的目标就是使代码做好测试准备，并持续不断地将应用配置到合适的环境，以便应用得以进行持续测试。

如果将此处描述的流程扩展到将服务、应用或者系统交付至类生产环境以及最终生产环境的过程中，那么其过程和目标也是相同的。通常情况下，尽管在交付生命周期的早期已进行了所有的测试，但将应用交付到必须尽全力保障稳定的生产环境之前，运维团队还是想要运行自己的一系列冒烟测试、验收测试以及系统稳定性测试。安全团队也想要在类生产环境中运行其自己的一系列安全性及合规性测试。就如同测试环境一样，这些都是需要准备的类生产环境，这些环境需要为所运行的各种测试提供必要的测试自动化以及测试数据。只有完成最终阶段的持续测试后，应用才会交付到生产环境。

简而言之，如果需要前移所有这些不同的测试并且使其运行时间更早、运行频率更高，则需要具备四项关键能力：

(1) 根据需要按时提供类生产环境的能力

(2) 测试自动化的能力，以提高测试运行速度以及可重复性

(3) 对不具备的或者会显著增加测试成本的服务以及环境进行虚拟化的能力

(4) 针对所运行的不同类型测试的多重版本，以更早时间、更高频率持续提供所需新版测试数据的能力

类生产环境的概念已铺天盖地。自动化测试的需求也是不言而喻的。启动测试前移需要快速且重复的测试，不可能使测试人员都能满足这种实验需要。如果运行功能测试需要 3-4 天的迭代周期，那么为期 2 周的迭代有效周期有什么益处呢？高效交付应用有什么好处呢，只是滞留三周时间以满足安全团队运行其测试及验证所需？因此，所有的测试都有必要自动化进行。自动化提升测试速度、实现测试模块化及可重复

性，并且使测试成为组成部分参与持续集成与持续交付流程。单元测试、集成测试、功能测试、性能测试、负载及压力测试、安全测试、合规测试等所有形式的测试都有相应的自动化工具。集成交付流水线中都应该有这些工具，以确保组件、应用以及服务的持续交付得以更早、更频繁地进行全面测试。

现在要讨论的是最后两项能力，即启动测试前移所需的服务虚拟化以及测试数据管理能力。

2. 测试服务及环境虚拟化

> 部署准备好的，虚拟其余的。
>
> ——IBM 测试实践

在准备将组件、服务或整套应用部署到测试环境时，为了运行某些测试，会遇到测试该应用所需的所有服务、应用、环境以及数据源都不可用的情况。除去可用性问题，用于测试目的，特别是在频繁测试小批量变更集时，可能成本过高。在这种情况下，等待所有服务都如期准备好再运行测试是不现实的。这可能成为交付流水线的一个主要瓶颈问题。在任何一种情况下，测试应用的过程中为实现测试功能所消耗或者利用的应用、服务、环境或者数据源，都需要虚拟化。通常情况下，开发人员会编写打桩代码来替代测试所不具备的那些应用或服务。编写及维护这类打桩代码耗时久、成本高，并且很容易出错。通过将自动化和弹性伸缩引入流程，利用服务虚拟化工具，可以解决这个问题。这类工具包括 IBM Rational Test Virtualization Server(之前是 Green Hat) 以及 CA Service Virtualization(之前是 ITKO LISA)。通过在测试虚拟服务上运行的虚拟代理，这些工具可以模拟应用、服务以及数据库。详见图 4-15。

首先利用几个必需的应用、服务以及虚拟数据库运行测试。随着项目日渐成熟，有越来越多的应用、服务以及数据库可供使用。虚拟资源做好准备并可以使用后，会逐渐为现实资源所取代。最后，在进行测试的应用发布至生产之前，需要利用所有的现实资源进行测试。

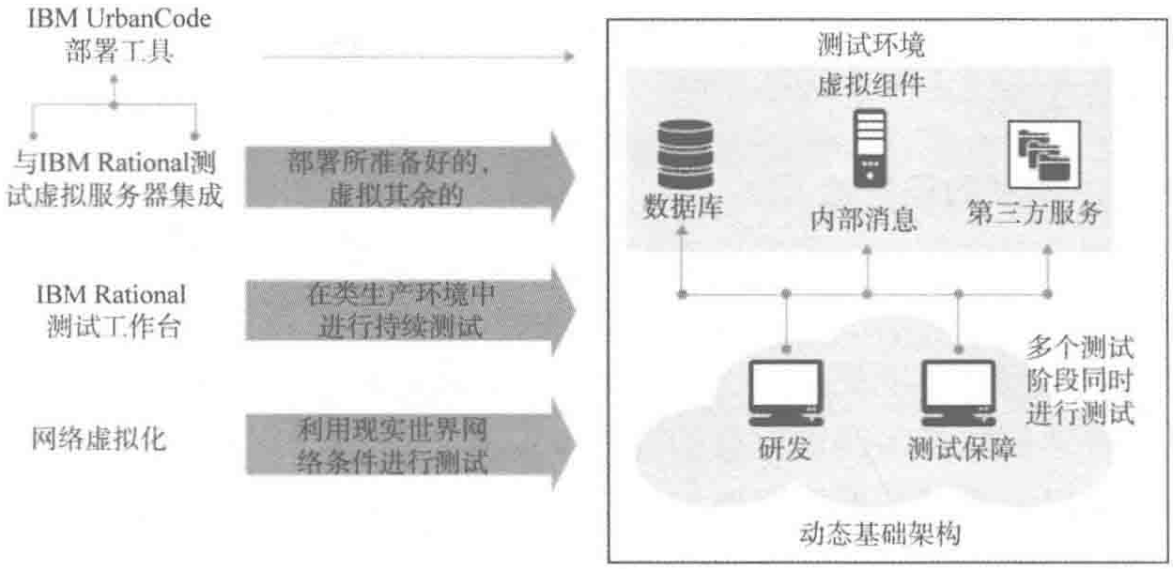


图 4-15　测试服务虚拟化

除了应用、服务以及数据存储之外，环境可用性也可能成为瓶颈。例如，在大型机环境下，一个逻辑分区专门用于测试并由多个应用共享的情况并不少见。相反，在分布式环境下，服务器专属于少数的测试试验的情况则屡见不鲜。出现这种情况是因为测试环境的获取、提供以及配置过程太烦琐，以至于项目一旦获得了所需环境，就会保留并在大多时候保持空转，以便在测试需要时随时使用。大型机面临的挑战是独特的，本章后面部分专门针对大型机的方案中会详细讨论。在测试实验室中引入云托管环境对于解决这些可用性及应用问题至关重要。这些环境可能位于私有或者公有云上。云托管环境使得测试环境能够在需要的时候可用，不需要的时候关闭。

测试环境即服务的特性是持续测试领域中一类新的云服务。通过导入所有现有环境的虚拟机，这些特性提供弹性伸缩的云托管环境，而这些环境可以完全克隆现有的本地或云端环境用于开发测试的目的。其中为 DevOps 而创建的一些环境也有能力进行环境的完整克隆，包括状态及网络(互联网端口 IP 地址)。测试用例对于实现持续测试是很理想的选择。测试人员发现了缺陷，可以创建完整的克隆环境，捕捉发现缺陷时的系统状态。然后发送给开发人员一个指向克隆的 URL，之后利用合适的环境、工具、状态以及网络地址，在适当的云数据中心为开发人员重新分配环境。这就消除了一个瓶颈，开发人员不必再努力重建测试人员所发现的缺陷。利用 Skytap 公司技术的 IBM 开发及测试环境服务(IBM

Development and Test Environment Service，IDTES)，便是这类服务的一个很好案例。

3. 测试数据管理

测试数据管理指的是迅速创建针对性的、规模适当的测试数据库，而非克隆整个的生产环境。不需要手动创建及维护测试数据，持续测试的开发与测试环境更加易于管理。简而言之，测试数据管理是在测试需要时快速创建真实测试数据的过程。

所以，测试数据管理要如何与 DevOps 相契合呢？如前所述，为了验证应用的集成、功能性以及性能，DevOps 要求开发及测试人员要定期进行应用部署。DevOps 测试的目标就是，通过执行适当的集成、功能以及性能测试去执行这些验证。这意味着应用在每次集成及部署的时候都应该进行测试，要求每次都有测试数据集。提供良好的测试数据集具有绝对的挑战性。开发人员每次部署新版本应用时，都需要使用全新以及更新的测试数据进行测试，这进一步加强了这种挑战性。为了应对挑战，测试数据管理已然成为 DevOps 生存的先决条件。

实现 DevOps 持续集成与交付的测试数据管理的最佳实践列表如下(Moran，2013)：

- **发现测试数据**。测试用例需要与相应的测试数据相关联，为每个测试用例找到正确的测试数据是至关重要的。某些情况下，数据可能位于多个生产数据库中。比如，一个应用既要使用客户关系管理(CRM)数据库中的客户档案，同时还要使用另外一个库存管理系统数据库中相关的采购商品明细。目标是要获取端到端的业务流程以及相关的测试数据，无论这些数据存储在什么位置。这使得团队能够将适当的数据提取到测试用例所需的子集当中。
- **自动创建真实的规模适当的测试数据**。组织要么手动创建测试数据，要么就是通过克隆整个生产系统来获取测试数据，而非仅仅提取支持测试用例所需的测试数据子集。这些手动流程无法提供 DevOps 持续集成与交付所需的敏捷性。测试数据的自动生成能够根据需要为各种类型的测试快速创建测试数据库。

- **基于合规和保护目的屏蔽敏感数据**。保护数据隐私已不再是可以选择的事情，而是法律规定。组织必须有相应的程序来屏蔽非生产环境中的数据，以符合数据隐私规则并避免数据泄露。数据屏蔽可以为开发团队提供有意义的测试数据，而不需要暴露诸如个人识别信息(Personally Identifiable Information，PII)以及受保护的健康信息(Protected Health Information，PHI)这类敏感的私人信息。屏蔽能够提取真实的数据，确保数据是真实的而非捏造，以便敏感数据不遭到泄露。
- **为持续交付更新测试数据**。为了每次交付新版本应用时能够运行测试，并在下一版本发布时再次运行，测试人员及开发人员需要持续访问相应的测试数据，以实现持续交付。组织可以让测试人员及开发人员不需要通过数据库管理员而直接利用工具与流程来更新测试数据，以此简化测试数据的交付。这提高了工作效率，提供了更多的时间用于测试，使发布得以更加快速及持续地进行。
- **分析测试数据结果**。在功能测试确认应用的运行状态时，测试数据管理使组织能够对测试数据中的变化进行评估以确认成败。通过对比测试前后的数据来分析测试数据结果，有助于评估测试通过与否。此最佳实践可以解决任何隐藏的失误，使组织得以快速识别并解决持续集成与交付缺陷。

4.3.7　方案：运维参与前移

撤回守门员

体育运动中最激动人心的 90 秒是由不在现场的人创造的。随着时钟滴答滴答地走向两位数，冰球队还落后一两个球，教练撤回了守门员，派出另外一名滑冰运动员进行孤注一掷的冲刺。多半情况下，球会从球棍、身体、墙壁戏剧性地反弹回来或者射入无人防守的球门。但是这种六对五的赌博往往足以值得冒险，特别是在别无选择，即使不这样做也注定失败的时候。

—— Powers，2013

开发与运维团队之间一直是良性的紧张关系。在多数组织中，他们隶属于不同的汇报线，通常情况下由不同的负责人进行管理，使其组织各自独立。“紧张关系”背后的原因是合理的，在性质上也并非完全消极。开发想要掌握其新开发的或是更新的应用与服务并持续不断地将其部署到运维人员提供的稳定且快速的环境中。另一方面，运维想要的是稳定与控制。他们希望运行的所有应用可以共存，却不会彼此干扰或是影响到所运行的系统。他们希望开发团队构建的应用，其系统能够符合规范，而且一旦部署就不需要投入更多精力。

开发将运维视作为其运送贵重货物的渡船人。而运维将开发视作为其珍爱的货运船提供生意的供货人。(就如同美国海军与海军陆战队对彼此的看法，二者完全巧合)

——DevOps 会议上无意间听到

正如在“方案：敏捷实施”部分所述，敏捷开发实践的出现将这种紧张关系又提升了一个数量级。实施持续集成的开发希望运维提供能够进行持续交付的环境。测试团队希望能根据需要、按照测试团队的规范提供测试环境，还要构建能够反映生产环境的理想测试环境。测试团队完成应用的测试并通过后，最好利用的是自动化测试，持续交付程序会将应用交付至更高级的环境做进一步的测试并最终部署到生产。所有这些环境都应该自动提供并且能够真实映射生产环境。

理想情况下，所有这些都是自动且持续发生的。环境(开发、测试等直到生产)会根据需要提供并为相关应用进行配置。一旦应用交付至下一环境，当前环境也可以自动销毁。

对运维而言，这显然是文化与技术的重要改变。突然之间，他们不再只是每隔几个月就部署一个新版本的应用，现在则不得不处理开发人员每周甚至每天生产的数百个潜在构建。如果他们对这些构建信心很小或是根本没有信心，就更复杂了。此外，现在还要求他们启动新的环境去测试及验证多数的构建。

因此，他们需要改变开发-测试的参与方式、处理变化的方式、管

理环境的方式以及实现自动化的方式。要充分实施 DevOps，运维团队需要做好准备并愿意进行改变。最重要的是，运维与开发之间需要建立信任。采取以下步骤，可以增强这种信任：

- **参与前移**：DevOps 的一个关键目标就是使运维在交付生命周期的早期就参与进来并贯穿始终。在传统的瀑布项目中，只在有应用需要部署的时候，运维才会参与进来。在需求阶段，运维可能参与开发系统设计、规格及需求定义，之后就会脱离出来，直到开发团队开始部署工作代码(构建)才重新参与。对于 DevOps 实施，运维团队需要作为团队不可分割的一部分，全程参与整个流程。可以通过让运维团队成员参加开发团队的每日例会(敏捷开发及极限编程概念)这样简单的步骤开始运维参与前移。将运维纳入团队的不同合作模式将在本书后面部分进行讨论。
- **获取敏捷**：运维需要实施(部分实施)那些开发已经在实施的敏捷实践。他们需要开始使用迭代结构、工作项管理策略、采集工作积压的工作燃尽图及仪表盘等。最好与开发使用相同的工具及资料库。这并不意味着运维开始采用诸如敏捷开发的方法，进行双周迭代，但这会使其变得更加敏捷，以具有相匹配的能力按照开发-测试团队的速度实施改变。这就减少了他们之间的阻抗失衡。
- **虚拟化及自动化**：采用软件定义的环境为运维提供了最高程度的敏捷。后面的方案中将详细讨论这些环境。
- **变更管理**：运维需要采用他们的变更管理实践。这并不意味着要放弃 ITIL 这样已经验证过的实践，而是对其进行扩展及调整，去处理数量级更高的变更管理任务，针对所有配置的以及弃用的环境还有部署的构建，现在都需要执行这些变更管理任务。这就要求所有的运维任务实现自动化，从配置与管理环境到实施政策，到利用软件定义环境，再到标准化用以采集所有相关参数以及运营关键绩效指标的仪表盘。

1. 转变运维角色

如同多数变革一样，DevOps 变革也使运维团队的角色发生了转变。负责基础设施或平台的工程师们受到角色转换的影响最大。DevOps 变革之前，他们负责在服务器上执行所有的管理任务，比如提供或回收服务器，编排及配置服务器运行模式，安装及为操作系统和中间件打补丁。利用 DevOps，他们不再执行重复性的任务，角色转换至更高的抽象层面，在抽象层面他们设计、创建以及管理各种模式，也会管理这些模式所提供的运行环境。

运维团队提供这些模式，从开发到测试再到生产，通过自助服务目录，使其可以为操作人员所用。模式本身是环境拓扑，具有构建完成的镜像、相关的管理政策、根据编排预先定义好的流程进度以及配置脚本。运维人员可以提供这些模式，以按需获得环境，在定义好的政策约束下进行配置，利用流程进度安排部署，利用配置自动化脚本进行配置，并根据需要部署应用与数据。当环境不再使用时回收环境。这一切都发生在运维团队没有直接人工参与的情况下。

如果运维人员需要做出政策不允许的变更，这是从运维人员的角度定义的，那么他们可以就此改变向运维团队提出要求，运维团队将为其创建新的模式或者更新现在模式的版本。

运维的下一步工作是，提供更高的生产率，与此同时，为利用开发-测试-生产环境工作的运维人员创造更加精益、高效的工作环境。

当然，这样的机制最好是在云基础环境中启用。然而，运维所工作的抽象层面也是可以针对非云计算的环境进行提升。因为不使用云基础环境时会有一些限制，所以运维团队在开发-测试中将多少控制力移交给运维人员，也是取决于技术水平。例如，只有在使用云基础的环境时，利用 OpenStack Heat Orchestration Templates 这样的技术，才可以实现获取全栈环境的模式的设计与配置。也就是说，从计算、存储、网络到操作系统及中间件，还有最终的应用与数据，环境包括了所有的这些层面，所有这些都集中在一个模式中，提供完整的软件定义环境。

在大型机这样的遗留环境中，开发-测试环境通常位于专用的逻辑分区(Logical Partition，LPAR)，在这样的环境中操作时，运维团队是不

想将任何的环境控制权让给开发及测试的。比如，运维团队之外的人是不允许在测试或者生产专用逻辑分区上配置中间件的。运维团队会自己操作，并且只允许开发-测试运维人员将应用部署至专用逻辑分区或者中间件服务器。

在另外一种使用容器的情况下，只要容器本身是来源于运维团队所批准的镜像库中的镜像，运维团队就不会在意开发-测试运维人员在容器内进行部署或配置。当然，容器的运维隔离性是容器价值的重要保证。

进入基于向平台交付应用的模式时，运维团队的角色进一步转变。平台可能完全由供应商管理(比如 IBM Bluemix PaaS)，此外，所交付的应用还有可能需要使用来自多家供应商的第三方服务。这里运维的角色就变成一个服务中间商及管理者，管理内部以及供应商提供的服务、处理合约、计量、安全以及服务等级协议。他们要确保所有的应用服务以及 PaaS 提供的云服务可以使用，功能以及性能都符合设计预期，以使组织自行开发的应用及服务保持良好运行状态。如果组织也向其客户及合作伙伴提供服务，并且也相应地利用服务开发自己的应用，那么运维的职责也包括管理这些对外服务的合约、计量、安全以及服务等级协议，确保服务可以满足使用者的期望。

运维角色的转变要求实现良好的、全面的、跨平台的监控以及持续不断的反馈。这是运维团队不需要动手操作每项事务，就能确保环境及系统实现设计的运行状态及表现的唯一途径。

2. IT 服务管理与 DevOps

当 DevOps 的实施涉及与诸如信息技术基础架构库(ITIL)、业务流程架构(Business Process Framework，eTOM)、信息系统和技术控制目标(Control Objectives for Information and Related Technology，COBIT)这类的 IT 服务管理(IT Service Management，ITSM)架构相匹配的问题时，就出现一个共同的话题，并且经常被推翻。其中，信息技术基础架构库(ITIL)已经成为历年来多数 IT 组织的首选架构，用来构建可存档、可重复的流程以管理及治理交付给组织的 IT 服务，向客户与用户交付商业价值的应用在此基础上部署、运行及管理。应用要以可伸缩、稳定以及可预

测的方式运行，这些 IT 服务的质量、伸缩性、稳定性以及可预测性是至关重要的。ITIL 提供了 IT 组织可以用来支撑这些需要的架构。ITIL 倾向于严格的变更管理及服务管理流程，引入持续交付这样的 DevOps 实践似乎与 ITIL 的做法冲突。事实上，DevOps 实践倾向于提供高效、可重复、自动化以及可伸缩的流程，与 ITIL 的目标完全一致。尽管提高部署频率似乎增加了变更管理与服务管理流程所必须处理的复杂性，但缩小的批次规模以及更短的周期时间实际上降低了每次部署的相关风险。同样，IT 服务管理实践现在是以更高的频率处理更小的变化，而不是以较低的频率处理大型的、复杂的变化。

源自 ITIL 这类架构的 IT 服务管理，提供了相应的功能，在四个核心领域支持 DevOps(Hodges，2015)：

- **配置管理。**保证具有类似架构的开发、测试及生产环境的一致性
- **事件管理。**对任何环境中识别出的问题及事件，能够及时采取处理措施
- **基础设施及应用性能管理。**提供持续监控，确保应用质量稳定。这在“方案：持续监控与反馈”部分将进行更多讨论
- **业务服务管理。**提供有分析数据支持的业务仪表盘，为所有利益相关者提供持续业务反馈，使其得以在必要时调整方案

IT 服务管理实践及工具由此实现并确保了连续可靠的操作，同时为交付流水线中的所有利益相关者提供反馈。他们促使运维参与前移至生命周期的更早阶段，并且提供了持续反馈的能力，从而实现应用、环境以及交付流程的持续改进。

使 ITIL 实践与 DevOps 相匹配的方案中包括三部分关键内容：

- 使 ITIL 实践更加精益、高效，以缩短诸如变更管理或者相关的事件管理这类的 IT 需求的处理周期时间
- 通过引入政策以及基于规则的自动化，减少 IT 审批流程中的手动步骤
- 自动获取参数并审计集成交付流水线中的追踪数据作为符合 ITIL 限制的证明

自动化可显著提升 IT 服务管理流程价值的领域如下：

- 利用基于业务流程管理(BPM)的编排工具，比如 IBM Cloud Orchestrator(ICO) 或者 VMware vRealize，实现服务与环境的编排自动化
- 日志与告警自动化，及使用 Splunk 或 IBM Operations Analytics 进行数据相关性分析
- 利用诸如 ServiceNow 或 IBM Control Desk 这类的工具实现事件管理的流程自动化

4.3.8　方案：持续监控与反馈

足球比赛业绩反馈

2008 年 9 月，当曼彻斯特城足球俱乐部被阿布扎比王室成员所拥有的私募股权机构(阿布扎比联合开发投资集团)收购的时候，球队突然间发现自己拥有了挑战英超所必需的资源……

每场比赛过后，他们就团队的业绩数据编制详尽的报告，重点放在其认为最为相关的统计数据上。所列内容范围广泛。例如，他们分析突破防线的次数，突破防线是借用自橄榄球的一个术语，指的是穿过对手的中场，或者更为关键的后卫线向前传球。他们审视团队赢球或输球后的 20 秒内发生了什么。关注球队在球场最后三分之一部分的控球率，这是其发现的与赢得比赛高度相关的衡量指标。

—— Medeiros，2014

由于 DevOps 的其中一个目标就是为交付流水线上的所有利益相关者提供快速的参数反馈与分析，因此建立持续监控与反馈的能力至关重要。DevOps 专家 Gene Kim 在其博客中发表了《实现 DevOps 的三种方法》，其中以放大反馈环路(Kim，2013)为目标对此进行了描述。他指出："几乎所有的流程改进初衷都是要缩短并放大反馈环路，以便持续进行必要的修正。"反馈是对应用、交付环境以及用来交付应用的流程进行改进所必需的输入。

提到持续监控，人们总倾向于仅仅将精力集中放在生产环境的监控

上。然而，持续监控要包括交付流水线中开发、测试以及生产的每一项环境的监控。同样，持续反馈意味着来自交付流水线中每个流程的反馈，包括测试运行、缺陷识别、工作项积压或延期、需求改变以及生产事故在内。

要确保以目标利益相关者所需要的形式提供反馈，这点至关重要。为业务分析提供日志，对其而言意义不大。然而，关于特殊功能使用量剧增的根本原因分析，配置或用户界面变化而带来用户行为改变的原因分析却与业务分析密切相关。

1. 提供监控与反馈

谈及监控与反馈，指的并不仅仅是生产中的监控，而是要贯穿整个交付流水线。监控非生产环境并从中收集反馈，可以在生命周期的早期，在应用或服务被部署到生产之前很长时间发现性能相关的潜在问题，从而使问题得以发现及处理。在交付流水线中，监控源于五个方面：

(1) 应用监控。收集参数与分析数据，了解应用的功能及性能表现是否符合设计预期。如果答案是否定的，那么是在什么条件存在问题，又是哪种类型的问题？

(2) 系统监控。包括基础设施在内的整套系统在功能与性能表现上是否符合设计预期？如果不是，什么样的负载或基础设施(计算、记忆、存储、网络等)条件导致了问题的发生？

这也包括事件管理。

(3) 应用用户行为。应用是如何被使用的？某些特性及功能是用得很多还是根本没用？用户把时间用在应用的什么地方？他们在做什么，是正在使用特性还是在努力解决应用使用过程中遇到的问题？

这也包括进行 A/B 测试，即将应用的多个不同版本交付给不同的用户群，以测试新的特性与功能。下一章中将详细介绍 A/B 测试。

(4) 用户情绪。确保用户体验(UX)是积极的和建设性的已经成为多数交互型系统应用的关键核心。这些应用直接向终端用户提供业务价值。最近已经开发出一些技术与架构，以提供高品质的用户体验，包括设计思维、精益用户体验等。评估用户体验以确定人性化的用户体验究竟该如何实现，这已然成为用户体验设计师的一项必需评估以及反馈来

源。这就导致整套的实践与工具被开发出来用于评估及获取用户情绪，即用户的感觉如何。工具可用于评估用户与应用的真实反映，他们使用的是哪一部分，是否正在某些领域痛苦挣扎，用户是否有建设性，或者看起来已不知所措？也有一些工具能确切捕捉直接用户的反映。IBM 的移动测试(Mobile Quality Assurance，MQA)工具就是很好的例子。

用户情绪反馈的另一重要来源是社交媒体。这不容忽视，因为许多消费者都倾向于根据其在社交媒体上的所见所闻做出决定。像 IBM 沃森社交媒体分析这类的工具与服务能够通过各种渠道获取社交媒体上的帖子并进行分析，以提供用户基础情感的整体视图。

(5) 交付流水线度量指标。所有的参数收集与分析功能都不必局限于终端用户及生产系统。交付流水线及所有服务于构建物的利益相关者都是数据的丰富来源。这些参数能够反映出所交付的项目、工具、环境、构建物以及工作的真实状态。这点将在下一章讨论。

交付流水线度量指标

介绍 HYGIEIA

在 Capital One 公司，我们相信，尽管工具、自动化以及协作都非常重要，持续反馈环路对于 DevOps 的成功也是非常关键的……

所以在评估了许多这类的仪表盘产品之后，我们决定是时候创造自己的产品了，因为 Capital One 公司需要一个单独的仪表盘来实现全交付流水线在任意给定时间点的可视化。

设计和制造仪表盘的时候，我们致力于使其配置简单、易于使用。另外，我们知道这款产品对于其他人也有用处，所以我们本着与全世界共享的目的进行构建，并提供一个开源版本。

我们的 DevOps 仪表盘，现在被称为 Hygieia[SM](如图 4-16 所示)，已经在 Capital One 广泛应用，很荣幸现在提供给你……

这款仪表盘的主要用途是使交付流水线中的任何障碍都易于发现，以便团队成员可以立即采取行动进行移除。

——Tapabrata “Topo” Pal，总监，下一代基础设施
Capital One(Pal, 2015)



图 4-16　Hygieia，源自 Capital One

要介绍提供一款仪表盘以呈现并可视化交付流水线的端到端参数的意义，没人能比我的朋友，来自 Capital One 的“Topo” Pal 做得更好。阅读 Hygieia 的介绍(以及原网页上更完整的版本)，可以看出，正是通过利用这样一款仪表盘实现交付流水线数据与分析的持续反馈来提供核心业务价值点：

- 持续反馈的回环对于 DevOps 是必不可少的。交付流水线上的每一位操作人员都应该能够看到交付流水线的相关度量指标，并根据这些度量指标调整其当前工作以及构建物相关的后续工作。
- 利用单一的仪表盘实现整条交付流水线的可视化，这是一个理想的解决方案。仪表盘要有能力处理来自交付流水线中每一个工具的参数与数据。
- 仪表盘应该持续不断地揭示瓶颈问题。如果说价值流图的目标是发现交付流水线中已经存在的瓶颈，仪表盘则是实时的瓶颈问题检测系统。
- 这样一款仪表盘应该易于配置与使用。安装及配置仪表盘不应该将新的瓶颈问题带入交付流水线。此外，作为直观的呈现，度量指标与分析数据应该便于使用。

Hygieia 是 Capital One 启动的一个开源项目，以创造一款仪表盘，实现整条交付流水线的可视化。图 4-16 和图 4-17 是 Hygieia 仪表盘的截屏。IBM、HP、XebiaLabs、Jenkins 这几个供应商已经将代码贡献到 Hygieia，使源自他们自己工具的参数得以在仪表盘上显示(Capital One GitHub，2015)

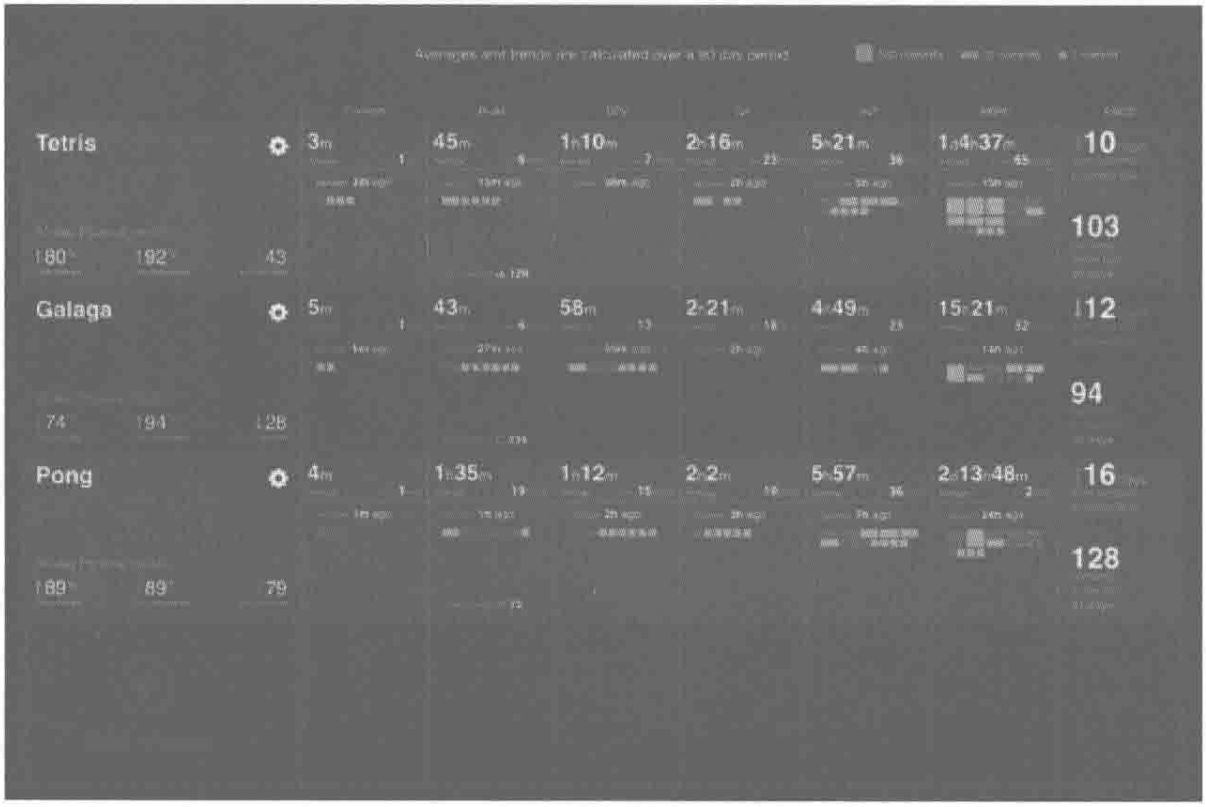


图 4-17　Hygieia，流水线视图

2. 持续改进

在当今的体育运动中，成就取决于投入。运动天赋能力很重要，但那只是进一步提升竞技水平的基础。训练的付出在四十年前似乎是难以想象的复杂与难以完成，而现在努力训练却是不被淘汰的必要条件。运动员不只是训练比之前更加努力。正如 Mark McCLusky 在其精彩的新书，Hudson Street 所出版的 *Faster*，*Higher*，*Stronger* 中所描述的那样，他们还要更加聪明地工作，利用科学技术来提高训练和成绩。仅仅适量饮食和努力训练已不足够。

—— Surowiecki，2014

尽管迈阿密海豚队创造了 1972 年的传奇，但记录终将被打破。索罗维基所要表达的是，在体育运动中，运动员每个赛季都在成长。他们越来越强壮、训练不断加强、使用更好的设备、配备更好的教练、利用更好的技术不断学习如何进步。他们努力奋斗，要超越的并非是前辈队友或者目前的竞争对手，而是前一天的自己。

实施 DevOps 需要将这种持续改进的文化引入项目团队。

- 如何改进刚刚交付最新版本的应用或服务？
- 如何改进刚刚用以交付应用或服务的基础设施和环境？
- 如何能比上次迭代做得更好，以便能够以更低的成本，更快速、更高效的方式交付更高质量的产品？

改进只能通过与奥运或世界级选手提升训练相同的方式来实现，即通过检查日常训练的每个方面，通过可视化大量的、详尽的生产力及性能相关参数指标，通过发掘可以改进的地方，即使只是非常轻微的改进。毕竟，在某些奥林匹克比赛项目中，即使只是一毫秒的差别也可能导致与金牌失之交臂。

游泳项目中存在那么多平局的原因

在 50 米长的奥林匹克泳池中，按照当今男子 50 米游泳项目世界纪录的速度，每毫秒可行进 2.39 毫米的距离。国际泳联关于泳池尺寸的规定中允许每条泳道存在 3 厘米的误差，相当于这个数值 10 倍还多。能将游泳运动员的速度计算到毫秒吗？当然可以，但是无法保证获胜选手比赛的赛道短上毫秒级的赛程。(要修建误差范围更小的混凝土泳池几乎是不可能的。泳池的有效长度是根据环境温度、水温甚至泳池里是否有人发生变化的。)

选手使用相同的赛道的运动项目，比如滑雪橇，可以用毫秒作单位，因为这个问题无关紧要。尽管考虑到比赛时的发令方式以及声音传播难以置信的缓慢速度，速滑也使用了毫秒的标准，但是或许不应该这样选择。

—— Burke，2016

提到持续改进，清楚知道在哪里进行投资也是至关重要的。正如游

泳比赛，节省一毫秒的时间或许不能产生足够差距以赢得更高奖项，但是竞争对手却必须要节省百分之一秒才行。同样的，如果真正的瓶颈存在于项目变更审批或者需求获取流程中，聚焦与投资开发-测试周期以提高效率就不会产生显著影响。清楚改进所交付的应用与环境时需要关注些什么，这对于确保合适的关注与投资领域至关重要。高效完整的度量体系能够实现从项目开始到维护整条交付流水线的持续监控与反馈，还可以为应用交付流水线的交付流程获取关键的绩效指标。

4.3.9　方案：发布管理

> 不要在比赛当天炫耀或期许伟大。伟大来源于不断的训练。当你成功之前，就必须对自己有所期待。
>
> ——迈克尔·乔丹，篮球传奇

发布日就是比赛日。团队在软件中努力做到极致的每件事情都将发布至生产中。只有那时用户才开始使用应用来获取商业价值。

这是陈旧的思想，是大批量、完整应用发布的思想。通过实施 DevOps 实践，发布日不应该是持续时间很长的非常受重视的事件，而每个项目发布周期中仅发生一次，情况将不再如此。利用 DevOps，生产发布变得很简单，就如同转换配置设置以开放已经交付至预生产甚至生产环境的应用的新版本，或者打开特性开关在已交付应用中启用新的功能集。交付流程也应该很平常，因为已经通过交付流水线向各种环境交付过小批次的应用，即便没有上百种，也有几十种的环境。用户发布或许不经常发生，但生产(或预生产)交付应该是持续进行的。

1. 发布管理流程

让我们回过头来看看为什么组织会拥有如此烦琐与复杂的发布管理流程。这都出于对质量管理及成功发布的考虑。

发布的定义是什么？发布是将经过测试并引入生产环境的全新或更改后的配置项目或组件组合起来。可能包括应用软件、系统软件、硬件以及相关文档的组合。由于不同的团队可能同时交付这些组件的不同

版本，所以需要通过具有明确定义权限的正式测试与治理步骤，以确保通过相应测试并正确集成的正确版本组件得以成功部署。鉴于应用任务关键型的性质以及运行所遵循的监管与合规要求，发布管理流程可能相当严格。例如，比起发布怪物猎人游戏的下一版本(不管青少年们认为发布新的怪物猎人游戏有多重要)，发布支付处理系统的发布管理流程要更为严格。

发布管理系统中一旦包括来自多重应用交付流水线的多重组件或服务时就会变得更加复杂。关于这点已经在前面的“多级 IT”部分介绍过了。这里，构建物流转的协调需要在这些多重交付流水线间进行。需要正确理解并记录各种不同组件与服务之间的依存关系，以充分理解组件及服务变更所导致的影响以及发布计划中各种延迟所带来的影响：

- 组件可以单独发布还是需要同时发布？
- 什么样的依存关系要求组件或服务必须按特定顺序发布？
- 某些组件或服务的新版本能否与还未更新的其他旧版本兼容？
- 每个组件的质量门禁有多严格？
- 某些组件及服务在测试、安全/合规方面是否有特殊要求？
- 是否存在必定阻碍组件发布的严重缺陷？相反，是否存在不会阻碍发布的轻微缺陷？
- 如何处理发布计划的改变？
- 为了确保所发布的是组件及服务的正确版本，要如何进行资产及构建物的版本管理？
- 致使发布中断的部署失败是怎样的？如何给失败定级以及如何解决？
- 如何管理与记录发布清单？
- 如何处理发布回滚？
- 发布审核的要求是什么？什么级别的发布记录需要保留，保留时间多久？
- 发布完成的标准是什么？
- 发布是否有维护期，在此期间该版本发布不会标注为“完成”？

上述这些以及许多其他的问题描述了完整的发布管理流程。

鉴于发布计划要求的复杂程度，背后流程的自动化以及配置可视仪表盘，来提供流程和发布流程中可能会出现的问题与瓶颈的可视化，这点至关重要。IBM UrbanCode Release (如图 4-18 所示)及 XebiaLabs XL Release 是两款很好的发布管理工具。

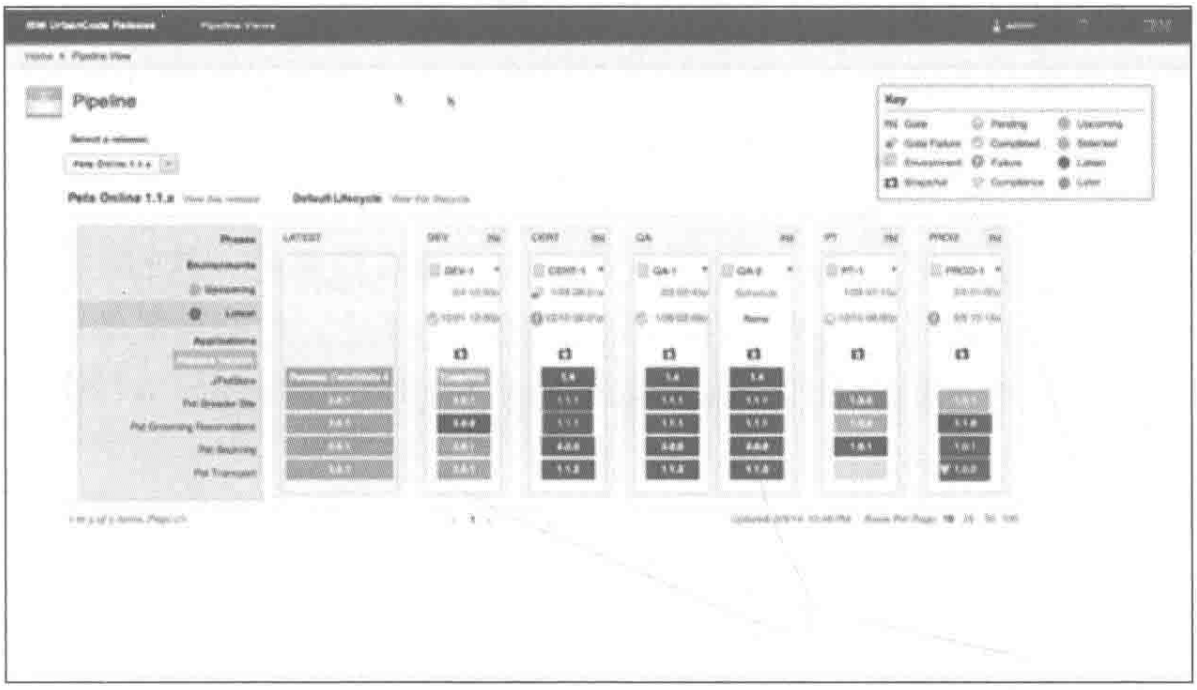


图 4-18　IBM UrbanCode Release 中多重交付流水线发布管理

2. 非持续发布周期的持续交付

尽管正式的发布可能按诸如每月、每季度、每年等这样固定的频率进行，DevOps 实践仍可用于预生产环境的持续交付。如今，所有的发布管理流程都适用于预生产环境中的组件及服务。在预生产持续交付过程中，发布管理流程或许和以往一样严格，但是自动化使其速度变得更快。此外，以更高的频率对预生产环境进行小批次部署，缩短了通过所有质量门禁所需的时间，降低了流程的繁杂程度。在向用户实际发布时，发布就成为从预生产到生产环境部署的一个几乎微不足道的步骤集。有些组织的预生产环境与生产环境完全一致。发布的时候，他们只是转换网络(域名服务器，DNS)设置去实现两个环境的转变，使预生产环境转变成新的生产环境。原有的生产环境重新构建成下一个预生产环境，用于下一次发布。

4.4 专注核心方案

游泳运动员参加游泳比赛的多个比赛项目，这并不罕见。迈克尔·菲尔普斯曾多次在奥运会上独自包揽多达八个独立项目的奖牌。在游泳比赛中，不仅按照距离(100 米、200 米等)可以划分为多种项目，依照泳姿(自由泳、蛙泳、仰泳、蝶泳以及包括所有这四种姿势的混合泳)也可以划分为很多单独的比赛项目。尽管游泳的基础原理都是一致的，但是运动员需要针对不同的比赛进行不同的训练。基本上，他们会采取核心战术，然后根据每场比赛的性质是按距离划分还是按泳姿划分进行微调或加强。

接下来的几个方案是技术性的。到目前为止，本书中所列的方案都是进行 DevOps 变革的组织可以并且应该采用的通用的、无关技术的方案。这些是任何一本 DevOps 变革手册中都需要包括的核心方案。然而，技术的细微差别使得有些方案在实施时，需要针对项目以及在此技术平台上工作的团队进行具体的微调或改进。本章的剩余部分介绍这些技术性的方案。

4.4.1 方案：移动设备 DevOps

实际上没有单独的移动设备 DevOps 这类东西。作为一种方法，DevOps 适用于所有的应用与组件，从前端的移动应用，到中间件，再到后端的服务器组件以及数据库。目标是应用贯穿于企业的所有开发与运维团队的 DevOps 实践及原则去实现包括移动应用在内的所有这些组件的持续交付。

移动应用确实有一些特定的需求及挑战需要解决。本方案中将介绍利用 DevOps 的能力去处理这些移动设备特有需求的最佳实践，正如将 DevOps 用于移动应用的交付一样。应用这些最佳实践的目标是要使移动应用的开发、测试以及运维实践与其他实施 DevOps 的企业应用相匹配。这些最佳实践的应用使企业得以在其移动开发团队间实施 DevOps，

交付更高质量的移动应用，并实现持续的改进与创新。

1. 移动应用特有的 DevOps 挑战

尽管企业与移动应用的 DevOps 基本原则是相同的，移动应用却对 DevOps 提出了特别的挑战。这些挑战包括(Williamson，2014)：

- **多平台支持**。大多数移动应用程序针对多个设备，这意味着要处理各种技术规范、操作系统版本以及硬件规格。安卓是众所周知的碎片化操作系统，因为每一个设备供应商都将操作系统植入了自己的设备(比如 Nexus 的安卓系统、Kindle Fire 的安卓系统以及 Nook 的安卓系统)。像 BlackBerry、Windows、Ubuntu 以及 Firefox 这样的商家正进一步分化移动操作系统市场。即使是苹果的所有移动设备中曾经标准化配备的 iOS 系统，如今也出现了很多变体。一个 iOS 应用需要支持不同的 iOS 版本：iPhone 4S 及以下规格；常规 iPhone 及 Plus 规格；iPad、iPad Pro 还有 iPad mini 规格。
- **作为企业前端的移动应用**。移动应用，特别是企业对消费者(Business-to-Consumer，B2C)或者企业对雇员(Business-to-Employee，B2E)类的移动应用，通常很少有业务逻辑代码在其本身移动设备上运行。相反，B2C 或者 B2E 的移动应用是作为多种企业应用及服务的前端来提供服务的，这些企业应用及服务为组织提供相应的业务功能，比如业务处理系统、人力资源管理系统或者客户信息采集系统。图 4-19 突出显示了这样一款本身业务逻辑有限的应用。这个例子就是流行的 LinkedIn 移动应用。(LinkedIn，技术博客，2011)

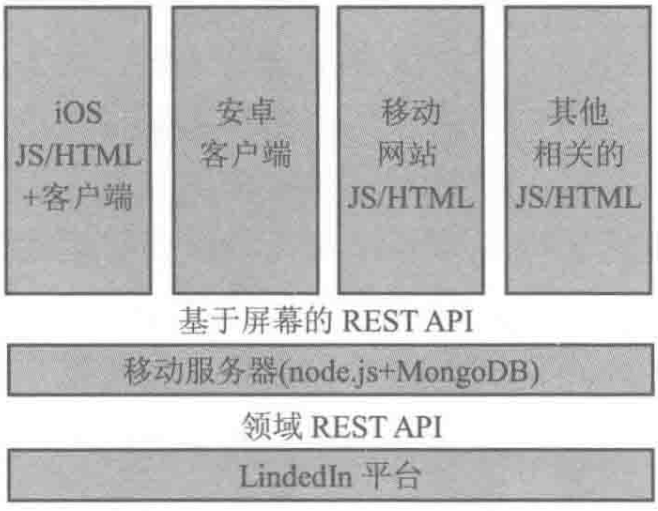


图 4-19　LinkedIn 移动应用架构

LinkedIn 移动应用实际上是 LinkedIn 向其用户提供核心业务服务的从前端至后端的一个平台。作为客户端或混合应用交付至多个平台的移动应用需要与后端 LinkedIn 平台服务一起开发及交付。如前所述，实施 DevOps 的挑战是要综合考虑企业中为应用提供功能和性能的所有应用与服务，并协调其构建、发布流程及生命周期。

- **交付速度与周期时间**。基于移动应用快速面世、获取反馈并且迅速迭代的强大商业动机，移动开发项目的时间表通常非常紧迫。历时几个月甚至几周的产品周期比比皆是。移动应用快速交付的压力使得敏捷开发的方法得以实施，以便移动项目取得成功。这就产生了利用极短的周期时间完成持续集成与持续交付的需求。

对于移动应用的持续集成与持续交付，开发人员所交付的应用变更需要针对所有的目标移动平台进行集成与构建。如果该应用是混合或原生的移动应用，则每次当开发人员交付应用变更集时，都需要同时启动几种不同的应用构建。因为所支持的每一个移动环境，其构建与配置都各不相同。为了提升构建速度，可能需要配置一个小型的构建服务器集群，用于处理多重目标的移动操作系统的持续集成构建。

- **应用商店**。通常情况下，移动应用无法直接部署到设备上，而是要通过应用商店。苹果公司推出了应用分布模型并锁定其设备，以防止应用开发人员或供应商直接安装应用。所有移动平台供应商都纷纷效仿。因为开发人员无法按需部署应用更新，所以应用商店在部署过程中增加了一个额外的异步步骤。即使是针对关键缺陷的修复，新的应用版本也要通过应用商店提交及审核流程。持续交付由此变成“提交与等待”。
- **“拉取”而非“推送”***部署*。传统的部署是在推送模式下进行的，这样操作可以按需推出新版本应用，无论是网页应用还是其他任何基于服务器的应用。更新移动应用的流程是一个“拉取”过程，然而，大多数情况下，用户必须自行选择去更新其应用。移动应用开发人员几乎无法控制既定用户在其设备上所使用的应用版本。从 DevOps 的角度而言，这意味着所部署的

与应用匹配的后端服务必须为移动应用的多个发布版本提供持续支持，无论是现在还是以前的版本。

- **负面评级的长期影响**。对于一个品牌来说，一个一星级的应用具有无与伦比的伤害性，尤其是当这个评级通过应用商店的媒体对外公布时。无论是否免费使用，消费者移动应用的不满意用户很快会公诸于众。虽然有关网站问题的投诉是传达给技术支持平台，但有关移动应用的投诉是通过应用商店播出，供大家观看的。移动应用必须进行广泛的功能性、适用性以及性能测试，以确保其质量。用户体验开始超越功能性。

2. 适用于移动应用交付的 DevOps 方案

基于移动应用所特有的挑战，以下 DevOps 方案已经针对移动应用做出调整与改进。

持续集成与持续交付

如何实现持续集成与持续交付取决于移动应用开发与交付的性质。

- **资产间端到端可追溯性**：本章前面部分讨论了所有构建物间可追溯性的价值。移动应用开发团队必须确保所有开发资产间的端到端可追溯性，比如代码、配置、脚本、基础设施即代码、测试脚本以及设计文档。此外，追溯性不得局限于移动开发资产，而是必须要扩展到移动应用所集成、连接或访问的企业应用与服务。
- **持续集成**：不仅要对移动应用组件进行持续集成，而且还要定期对正在开发的移用应用所访问的移动后端及服务器端组件与服务进行集成。
- **构建服务器集群**：为了实现移动应用的持续集成，开发团队可能会让服务于所有目标移动平台的移动应用代码共享中央构建与集成服务器。建立服务器集群，实现构建与部署流程自动化，可以确保所有支持的移动平台及目标硬件规格能够同时进行快速可靠的持续集成构建。
- **软件开发工具包(SDK)独立集成流**：开发人员应该为所支持的

每一个客户端的移动操作系统软件开发工具包保留独立的构建与集成区域。移动设备的分化已超越现有四大主要移动操作系统，即 iOS、Android、BlackBerry 及 Windows，并且每个操作系统本身也进行内部分化。包括 Ubuntu、Chrome 还有 Firefox 在内，一些新的移动平台也正在兴起。因此，移动应用开发人员即使只有一个目标平台，也必须编写多个不同的应用版本以支持每个目标平台及其变体。每个移动应用都需要多个版本的软件开发工具包。

为了确保代码的独立及每个目标平台的特定功能，开发人员必须为移动应用每个特定版本的平台保留独立的开发与集成流。

- **自动化部署：**移动应用开发人员习惯于利用集成开发环境(IDE)将应用手动构建并部署到相应的设备上。随着构建复杂程度及数量的不断增加，开发人员应该建立自动化构建与部署。这不仅能够确保构建与部署的较高质量，而且能够确保任何开发或测试人员在任何时间都可以根据需要复制及部署每项构建。

测试与监控

基于硬件规格各异的移动平台与设备，其测试与监控呈现出指数级的复杂性，或许需要提供相应支持。

- **模拟器和物理设备上的自动化测试：**测试自动化是移动应用开发落后于企业应用的一个领域。大多数的移动开发测试都是在模拟器上而非物理设备上广泛进行。更有甚者，模拟器上进行的测试大多是手动操作的。考虑到开发的速度以及移动开发所固有的敏捷特性，自动化功能回归测试是确保质量的唯一有效途径。由于所支持的平台及硬件规格众多，手工完成足够的测试是不可能的。解决办法就是利用自动化测试工具测试所有的应用，先在软件开发工具包所提供的模拟器上进行，然后还要在所支持的真实物理设备上进行。服务供应商可以提供不同设备的测试。

- **测试虚拟化**：这种方法可以虚拟及模拟移动应用测试期间不可用的移动后端服务。移动应用快速的开发流程导致其与后端的企业应用与服务相比，拥有更多的版本。如此快速的开发使移动应用在技术上一直领先于后端应用与服务，这意味着其有些新功能还无法被后端企业应用与服务所支持。如前所述，即使后端服务可用，可能还需要消耗成本或资源进行测试。开发团队可以通过虚拟化(模拟)后端服务来解决这个问题。移动应用需要与之交互的应用、服务以及数据库所组成的整个生态系统都可以作为虚拟情境使用，模拟移动应用需要进行交互的真实功能表现。这种安排使移动应用及其交互作用得以快速进行测试，也会节省运行这些服务与应用的真实情境进行测试所需的硬件资源。
- **应用与系统性能监测**：移动应用开发人员所面临的最大挑战就是应用在测试环境中性能表现良好，在自然条件下却无法运行。不够稳定的网络环境、较低的内存、较低的电量以及数据丢失是导致移动应用性能不佳的一些根本原因。并非所有的这些条件都能在实验室中进行预测及测试，所以开发人员必须能够在应用使用过程中对其性能进行持续监测。这种监测是全栈监测，不仅要监测移动应用，还要对提供移动后端服务的应用及系统同时进行监测。
- **用户情绪反馈**：最惨痛的失败莫过于应用在客户手里使用时功能失效或异常闪退。增加应用的能力，使得应用在问题发生时捕捉包括位置数据及设备特性在内的上下文信息，可以为开发人员提供足够的数据以发现导致问题的根本原因并进行修正。嵌入式崩溃捕捉与分析逻辑是移动应用的一个重要组件。像 IBM 移动测试服务这类的新型服务能够捕捉用户情绪，同时在应用功能及运行表现不符合预期的时候，直接在应用内就可以获取应用崩溃及用户反馈。

移动应用交付

移动应用处于持续交付的最前沿。查看某个移动设备，发现几十个应用每周都有更新，这种情况并不少见。然而，交付移动应用并不像交付 Web 应用或 SaaS 产品那么简单直接。

- **集中管理移动配置文件、证书以及 API 密钥**：无论是否将应用提交到应用商店，或者使用内外部应用所提供的 API，开发人员或组织都是通过供应商发布的配置或文档密钥来识别应用的真实性及所有权。这些密钥是通往商店或者 API 的授权。通常情况下，单个的开发人员会配备单独的用以开发的密钥。但是，应用最终发布时，就需要取消个人密钥，取而代之的是正式的组织密钥。曾经发生过这样的情况，因为官方应用发布时，没有用组织的密钥替换开发人员的个人密钥，或是因为组织的密钥或配置文件被泄露，或者同时因为这两种原因，使组织面临重大危机，甚至不得不下架应用。

组织的密钥及配置文件需要加以保护并且只能用于官方应用发布。针对密钥及配置文件管理的移动管理流程必须明确定义并且严格控制。限制组织的密钥及配置文件的访问尤其关键，因为这既是一个安全问题，也是一个需要严格治理的隐私问题。

- **利用虚拟应用商店来测试终端用户设备应用配置**：移动应用只能通过供应商的应用商店配置到移动设备。通常，在进入应用商店之前，应用要通过一个手动的审批流程。进入商店之后，用户需要下载应用，之后应用才被推送至其设备。开发团队可以使用个人开发应用商店来测试这整个流程。这些虚拟应用商店模拟真正应用商店的表现，使开发人员得以有效测试提交的应用并将其配置到设备的整个流程。
- **获取用户反馈**：通过允许用户提交评级及书面反馈的应用商店，移动应用有一套独特的反馈机制。备受欢迎的应用可能会收到四星或五星评级。不太受欢迎的应用通常会收到一星或两星评级，也有可能伴随着负面反馈。作为一种正式的集中化的机制，用户对移动应用的这种直接反馈不适用于其他的平台。通常情

况下，只有用户打电话给技术支持，在开发人员所监控的论坛上留下意见，开发人员才能发现传统应用的问题。因此，移动开发团队应密切监视应用商店的反馈和评级，并将反馈融入未来的用户故事、功能增强以及软件改进中。充分利用这些宝贵的反馈是持续改进移动应用所必需的。

文化与团队

在大多数组织中，移动应用团队通常都是拥有开发及交付工具与实践的小型团队。

- **移动应用团队与后端团队间的协作：**移动应用开发团队在组织内被拆分成不同的小组，分别作为供应商或者作为雇员，这都是很常见的。对于 DevOps 的实施，当在整个组织中实施 DevOps 时，非常有必要将移动开发团队纳入其中，即使移动团队只不过是组织中很小的一部分或者所使用的是不同的软件开发流程。在 DevOps 生命周期中，直接与企业应用及服务进行交互的移动应用享有特权。企业应用或服务添加新功能时，移动团队可以将其无缝集成到移动应用中。
- **向移动应用团队学习敏捷：**这些移动开发团队非常敏捷，具有快速开发及交付能力。他们的敏捷流程技能实际上也可以为组织的其他部门所用，从而显著提升价值。

4.4.2　方案：大型机的 DevOps

世界级的业务仍然在大型机上运行。在过去的 50 多年间，自从第一代分布式系统开始流行，专家们就一直在预测大型机的消亡。然而，尽管出现了云计算及移动计算，大型机仍在强劲发展。专门为大型机开发的 IBM z 系统，尽管已有些年头，但仍然适用(Sun, 2013)：

- 世界 100 大银行中有 96 家，美国零售 25 强中有 23 家，10 家全球最大保险公司中有 9 家在运行 z 系统。
- 世界财富 500 强企业中有 71%是 z 系统的客户。
- 全球 10 大人寿健康保险公司中有 9 家在 z 系统的大型机上处理

其大批交易。

- 包括最主要的信用卡交易及股票交易、现金交易、制造流程以及 ERP 系统在内，大型机每天大约要处理 300 亿单商业交易。

因此，大型机值得在本书中占用一些篇幅。此外，大型机确实也有一些细微的差别，这就要求 DevOps 方案在大型机上实施时需要进行增强或调整。

在大型机上运行大量工作负载的组织分享了一些常用的相关模式：

- 大型机团队所使用的流程与工具比不上团队向分布式或者云托管系统交付应用时所使用的。
- 多数大型机托管工作已进行很长时间，有些情况下甚至已进行了几十年，却很少有新的进展。尽管这些系统非常稳定且易于理解，但其具有遗留的架构，而且很难转换成现代架构以便更好利用。
- 运行及维护大型机系统的总成本是相当高的。这源于其成本模式，也因为工具与流程无法通过转换变得更加高效。
- 尽管这样说并不政治正确，但事实是大型机的操作人员年事已高，并且面临退休。由于年轻些的工程师对遗留的流程及工具不熟悉，或者不愿意使用过时的用户界面的工具，组织将越来越难以找到替代人员。

最终，大多数大型机系统都是提供核心业务服务的系统。其体现了工业化核心的核心。然而，大型机系统可能不会非常发达。其老旧的工具与流程、非现代化的架构、精益交付流水线的缺失导致这些系统无法快速提交变化，因此成为组织高速创新能力的累赘。边缘创新应用的交付速度有多快，这并不重要。如果其依赖于大型机运行上的后端服务，而且接下来的六周时间不会更新，在那之前，不会发布新的应用。

此外，从投资的角度看，如果最严重的效率低下是源于大型机上应用的交付能力，那么提高这些系统的效率将对资源释放产生巨大影响，无论是资金还是人力资源，释放出来可投资于创新。

为了使创新得以进行并且能够顺利进行，首先需要做的就是优化。

适用于大型机的 DevOps 方案

基于大型机应用交付所特有的挑战，下面的 DevOps 方案已针对大型机进行了调整与增强(Radcliffe，2014)。

- **维护同源存储库：**对于任何多平台的开发工作，使用通用的、跨平台的单源存储库对于实现持续集成至关重要。然而，大型机团队使用遗留代码库的情况也并不少见，遗留源代码库不同于也无法与其他非大型机团队所使用的代码库连接。如此特殊的存储库如果无法跨平台运行，那么 z 系统终将被孤立，无法参与持续集成实践。大型机上进行的任何工作集成都会成为努力过后的瀑布式集成。

  对于多年来可能一直使用相同功能的大型机开发团队而言，向现代源代码库的转换代表着巨大的变化。然而，同源代码管理工具对于管理所有的构建物，帮助打破仓筒以及消除关键瓶颈是至关重要的。
- **确保每个人每天都能坚持把代码提交到主线：**持续集成要求所有组件以及所有开发环境下的每一位开发人员都承诺，每天将代码交付至集成流，以协助确保集成尽可能地保持连续性。对于今天的在 z/OS 系统，在终审之前，很多用户都独立修改代码，直到最终集成时，他们才意识到自己的工作是受到其他开发人员的工作影响的。这可能会导致功能发布的延期，或者最后的变更未经过适当的测试就部署到生产中。代码定期集成有助于确保迅速识别出这些依赖关系，以便开发团队能够及时处理，并且不需要受时间限制。
- **自动化构建：**是自动化构建使持续集成得以持续。由于 z 系统的环境可用性以及访问成本都可能有问题，对于 z 系统构建，自动化就成为一项挑战。在生产环境业务运维期间，可用性必然会是一个问题。
- **测试自动化：**正如构建需要自动化，测试也同样需要自动化。持续集成的目标不仅是集成团队的工作，还要审视所构建的应用或系统在功能与性能方面的表现是否符合预期。如前所述，

持续测试要求自动化要包括按需构建软件、配置测试服务器、提供测试环境、将构建软件部署到测试服务器、设置测试数据以及运行正确的测试脚本这一系列的能力。

所有这些对于 z/OS 系统都是特别的挑战，但必须加以解决。要求配置可以进行构建、部署及随时进行自动化测试的环境有助于提高最终代码的质量。这需要系统资源可用，愿意定期运行大量自动化测试并且开发自动化测试。

- **保持快速构建：**快速构建至关重要，因为运行时间过长的构建是持续集成的最大阻碍。基于只构建更改文件的标准化实践，在 z/OS 系统的构建速度通常很快。然而，这些构建还是需要与其他平台上的构建进行协调，而且，在 z/OS 系统可用时安排适当的时间会是一个问题。
- **类生产环境中的测试：**在无法准确代表生产系统的环境中进行测试会给系统带来很多风险。然而，也不太可能每次都仅仅为了测试而克隆完整的 z 系统或者非 z 系统。创建一个有其他工作负载运行的克隆环境就更加困难。对于在 z 系统上托管的系统，维护这种测试环境的成本可能成为一个主要的问题。通常来说，数量有限的开发与测试逻辑分区必须在团队间共享。此外，为了保障生产环境性能，许多组织会小心控制测试团队每秒百万条指令(MIPS)的使用率。

  有些工具允许 z/OS 系统的非生产环境的实例在分布式系统运行来进行开发与测试，试图突破限制并使最大化 z 系统测试环境可用性的组织可以使用这类工具。IBM 针对 z 系统开发的 Rational Development and Test Environment (RD&T)就是这类解决方案的案例。提供了非生产用途的 z/OS 系统环境，比如在基于英特尔的 Linux 系统上所运行的开发与测试。此环境包括一个 z/OS 系统的硬件模拟器，能够利用必要的中间件运行真正的 z/OS 系统平台。在这种情况下，持续交付流程将应用交付至这些开发-测试环境并最终交付到 z 系统上的生产中。除了可以更加方便地访问多个类生产环境，这类解决方案还可以从大型

机卸载开发-测试环境，也可以释放 z 系统大型机专门用于生产用途。

- **实施自动化部署：**持续集成自然而然地导致了持续交付的概念与实践产生，持续交付是将软件自动化部署到测试、系统测试、类生产以及生产环境的过程。

  自动化部署在 z/OS 系统环境中很常见，因为 SCM 系统通常包括构建与部署。然而，大多数项目并没有足够的 z/OS 系统资源配置给每个团队来随时部署到测试环境中。部署还需要与分布式的基础设施进行协同，由于缺乏通用工具，部署可能会面临挑战。满足这些需求的有效实践就是持续发布与部署。

  对于 z 系统应用，有两种常见的机制或持续交付路径：

  (1) 对于拥有准备就绪的成熟部署工具及实践的组织，或许可以利用遗留的配置管理工具执行目标逻辑分区的部署。虽然这些工具在交付全部持续交付功能方面的能力有限，但可以自动化部署到 z/OS 系统。

  (2) 相反，组织也可以利用已经能够全面支持包括 z/OS 系统在内的所有多平台部署的专门自动化部署工具。像 IBM UrbanCode Deploy 这类的自动化部署工具，有一个 z/OS 系统代理，可以在目标逻辑分区上本地安装以实现持续交付。

- **协作与沟通：**如前所述，这是一个强调文化的方案。就像移动应用的 DevOps 实施一样，当在组织内全面推行 DevOps 实施时，非常有必要将大型机开发团队纳入其中，即使大型机团队或许使用的是不同的工具与流程。在对工具及实践进行现代化改造期间，他们应该加入 DevOps 团队并参与组织变革，以期更加精益、高效。如果组织有很大一部分工作负载及应用是在大型机上运行的，那么这样做会为 DevOps 变革带来巨大的投资回报。

4.4.3　方案：物联网 DevOps

关于科技是如何渗透到人们日常生活的方方面面——我们的家庭住宅、交通方式、工作场所、工厂环境以及我们的身心健康，其中最明

显的变化就是出现了智能或连接设备，统称为物联网(Internet of Things，IOT)。技术在这些领域已经存在了几十年，但物联网的出现使我们所使用的设备能够实时地连接到所有其他设备，并通过互联网连接到后端服务。与服务供应商进行沟通以及提供服务已不再只是智能手机和平板电脑的专利，如温控器、冰箱、汽车与卡车、商业暖通空调系统、安全系统、手表、运动记录设备、电视、甚至街角的摄像头，这些设备也能与服务供应商进行沟通并提供服务。

虽然所有这些设备都有软件，但这些软件在生产时就作为固件嵌入到设备中。更新版本就意味着更换物理器件——芯片、主板或者整个设备，新版本组件中已安装好新版软件(固件)。适当更新，即所谓的远程下载技术(over the air，OTA)也并不能实现，因为设备没有连接，在用户使用时也没有设计更新。固件即使能够更新，也是通过烧写到运行的芯片上来实现，只有利用专门的设备才能完成。

设备的应用交付流程有两个划分清楚的独立周期。在这类设备的开发阶段，硬件与软件都在开发，通常是彼此断开的。物理硬件的设计与制造，以及在此硬件上运行的固件开发都需要在设备发布之前进行同步。当然，设备无法在没有固件的情况下交付，如果硬件设计仍在改进及更新，软件开发就无法完成。当设备硬件的设计基本完成时，如果真正的设备还无法使用，固件团队可以先完成软件并在原型机或模拟器上进行测试。一旦设备可以使用，固件会移交给拥有该设备的技术团队，之后技术团队会对最终设备的硬件及固件进行测试。在发布之前，他们不断与软件开发团队一起处理变更及缺陷。图 4-20 指的就是这些团队，其分别作为软件开发生命周期(Software Development Lifecycle，SDLC)团队及产品生命周期管理(Product Lifecycle Management，PLM)团队。SDLC 及 PLM 都是组织常用的术语。一旦交付，下一个挑战就是新版固件的更新。如前所述，在前期的物联网设备中，唯一的途径就是更换固件中所嵌入的全部或部分硬件组件。

物联网的发展引起这类设备的数量猛增，交付物联网设备的组织开始实施 DevOps，以使整个交付流程更加精益、敏捷。下面是一些实施时需要针对物联网进行调整的关键 DevOps 实践：

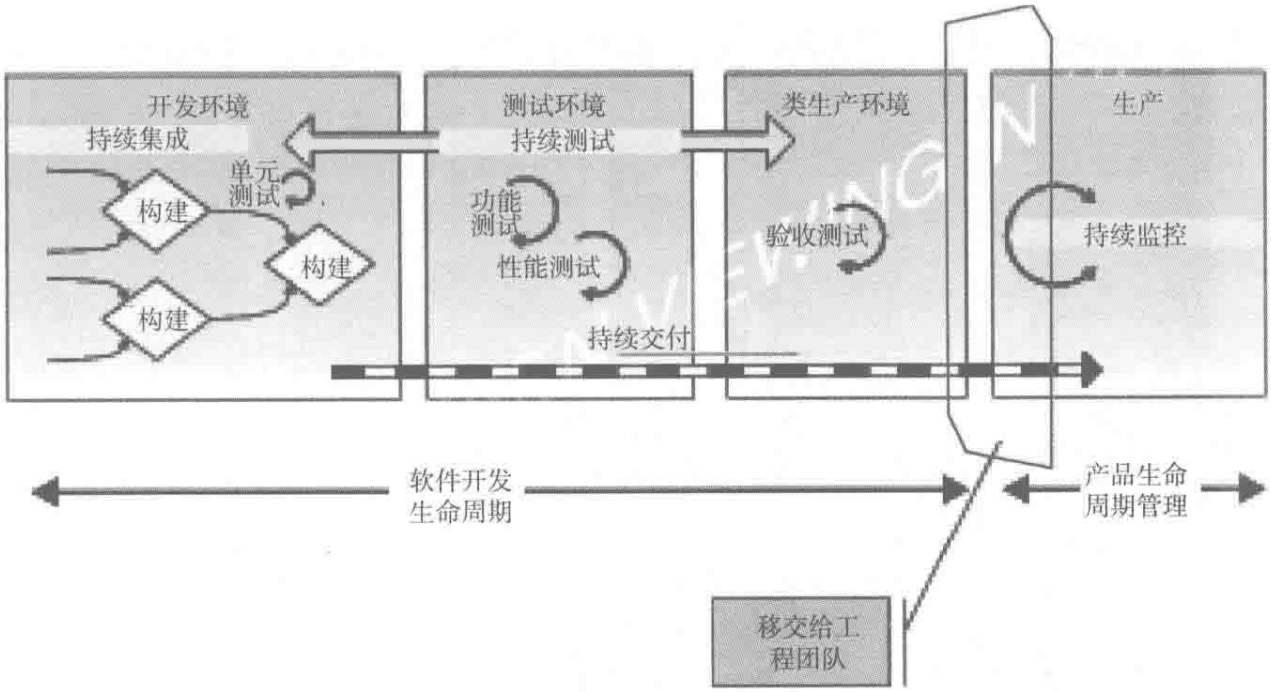


图 4-20　应用交付给工程团队的过程

- **持续集成与测试**。这些流程与传统软件交付一样。但是，尽可能与最终设备保持一致的模拟器或样机(类生产系统)必不可少。固件与硬件的开发周期需要更好地同步。
- **持续交付挑战**。一旦进入生产，即使是已经连接的设备也可能无法在更新可用时进入更新状态。设备可能是没有持续连接到网络，比如，只是在对接时连接，或者只是与附近的其他设备连接。这种做法可能迫使固件的更新只能在设备闲置或不需要维护的状态下(例如，暂停状态)才能进行。此外，设备只能通过“拉动”模式接收更新，而无法使用“推动”模式，“推动”模式需要用户来启动更新流程。
- **持续交付的硬件设计**。通常情况下，设备硬件不会像移动应用那样持续更新其固件。如果设备制造商打算建立一套适当的持续交付机制，以在部署到生产之后对其设备的固件进行持续更新，那么就需要基于这样的更新来设计硬件的架构。应该有一种方法不需要拆除设备就可以启动更新，而且不需要任何的专门设备或流程。理想的设备架构应该是可以扩展的，激活或导入新的功能，只需要更新软件或者打开软件功能，这些软件功

能已经安装，但是只针对某些付费用户或在某些特殊情况下启用。这类设备通常会有两层固件：一个核心操作固件(操作系统)，操作设备且很少更新；还有一个应用，向用户提供特定的功能与服务，更新更为频繁。

- **团队与文化**。在设备开发阶段，硬件团队与固件团队的联合是必不可少的。软件与硬件的架构、需求以及交付周期的设计是相互依赖的，要求两个团队(SDLC 与 PLM)协调一致。毕竟，在物联网 DevOps 中，Ops 指的是设备硬件团队。

4.4.4 方案：DevOps 用于大数据及分析

大数据及分析解决方案是一套完整的应用与服务，正驱动着对更新、更快、更频繁的软件交付方法的需求。以前，诸如交付简单的代码变更这类基本任务也需要四到六周的时间，然而，要等待数周甚至数月的时间才能更新客户端的情况如今已无法接受。鉴于所有的性能与功能的设计、开发、测试以及部署都要集中进行，以前交付全新的或者是改进版的大数据及分析解决方案需要几个月甚至几年的时间。持续交付作为提高业务敏捷性、加快交付速度、把握新的市场机遇以及应对多变形势的一种手段，越来越多的开发组织正转而利用 DevOps 作为其实现持续交付的途径。持续交付的目标是不断地设计、测试、部署持续的改进至生产，以便更加快速地为客户提供价值，并在发现问题时回滚这些变化。

通过实施 DevOps 方案，这些大数据及分析解决方案以更加快速、有效的方式进行交付，流程持续改进，同时确保了软件的变更及改进都是基于真正的客户反馈而进行的。

图 4-21 显示的是大数据及分析的参考架构，强调的是交付这类应用与服务的复杂性。架构描述了一组组件，这些组件促进数据的收集、部署、检索、分析以及报告，以在平衡相关的质量、实用性以及灵活性要求的基础上，实现预期的业务结果。

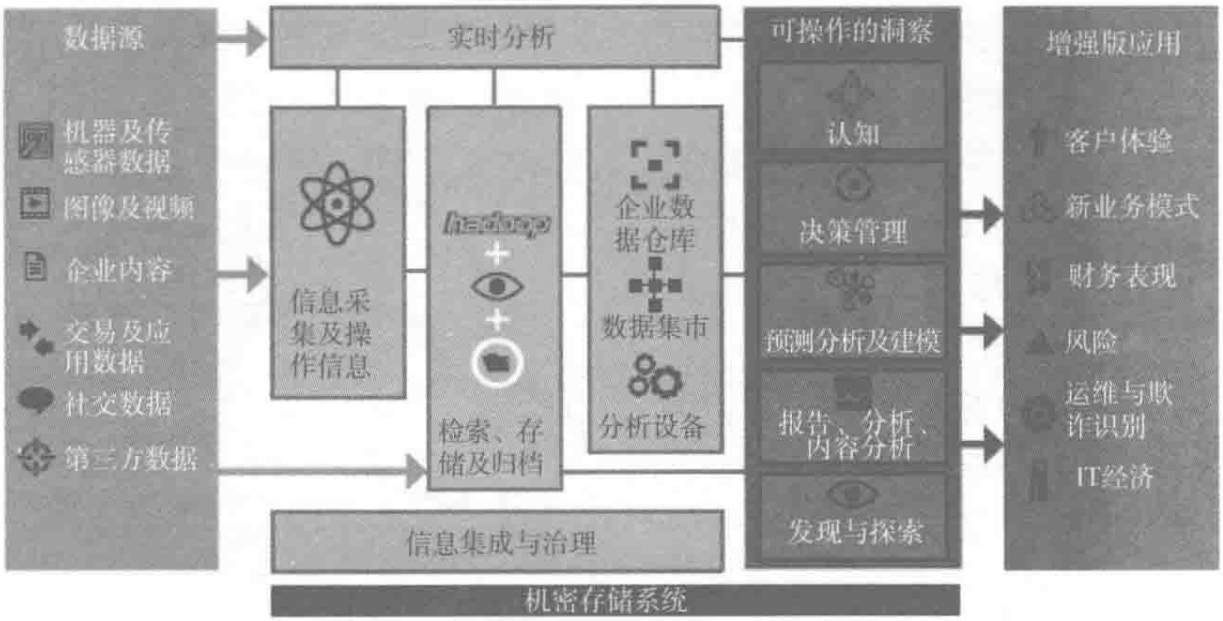


图 4-21　大数据及分析参考架构

分析解决方案统一方法

幸运的是，与大型机开发不同，大数据及分析领域与敏捷实践并不互相排斥。比如，IBM Analytics Solutions Unified Method (ASUM) (ASUM，2016)就是对如何完成大数据及分析方案的整个实施周期进行分步指导。ASUM 将敏捷与传统的实施原则相结合，以实现方案目标并为组织提供最优结果。这些原则都列在这里，很显然是与本章所列的 DevOps 方案相符的，这使得使用 ASUM 的项目更容易实施 DevOps：

- 项目被评定为符合敏捷原则应用。
- 项目的范围是确定的，并已初步收集业务需求。
- 业务人员与 IT 人员都是项目执行团队的重要组成部分。
- 需求是明确的，通过几次原型迭代，对需求进行微调。根据需求、时间表及可用资源的数量与优先级，采用阶段性实施方法来实现目标。
- 然后将原型结果与总需求进行比较，以评估结果并确定进一步的迭代。
- 利用迭代与增量开发完成配置与构建。

- 在整个项目周期内完成充分的测试之后，解决方案的第一阶段就开始了。
- 项目的其余阶段也按照相同的路径进行，将样机迭代与增量开发作为第一阶段。

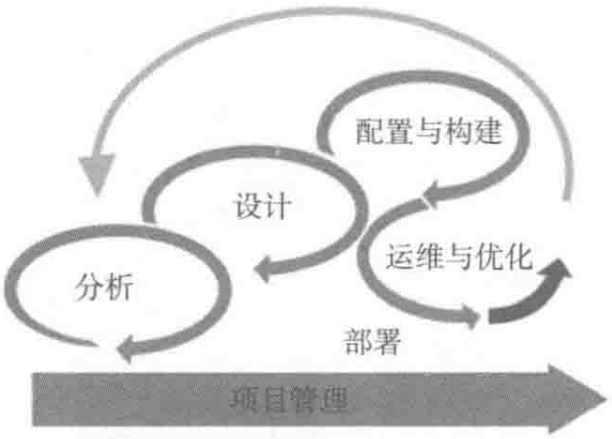


图 4-22 分析解决方案统一方法(ASUM)阶段

如图 4-22 所示，ASUM 利用项目管理的伞面，执行五个完全定义的阶段。

(1) 分析：定义解决方案需要实现的特性及非功能属性(性能、可用性等)。就这些需求，征得各方面同意。

(2) 设计：定义所有解决方案组件及相应附件，识别资源，并安装开发环境。迭代开发设计原型用于澄清需求。

(3) 配置与构建：基于迭代与增量方法配置、构建并集成组件。利用多环境测试与验证计划。

(4) 部署：制定解决方案的实施与运维计划，其中包括支持进度表。迁移到生产环境，必要时进行部署，并将部署传达给业务用户。

(5) 运维与优化：运维包括交付后的维护任务和检查点，以促进解决方案的成功应用及良好运行。还要不断地优化这些运维任务。

(6) 项目管理：包括协助管理流程以及监控项目进度与维护情况的流程。

接下来，以 ASUM 交付大数据及分析应用与服务的流程为例，将这些阶段应用到 DevOps 的实施方案中。

分析与设计

了解数据的来源、生产商以及消费者，这对于任何大数据及分析解决方案都是至关重要的。例如大数据与分析参考架构如图 4-21 所示，其中包括多个数据类型，在多个数据源，利用多种技术。此外，所产生的

数据与分析也可能需要按不同的数据类型通过多种技术交付给多个消费者。企业架构记录 IT 系统(数据存储、应用、基础设施及网络)及定义这些大数据与分析资源、生产商以及消费者，如前面部分所列的参考架构图(图 4-21)所示。

配置与构建

下面介绍大数据和分析应用背景下的相关配置与测试流程。

- **端到端可追溯性：**在软件开发生命周期及数据管理生命周期中，可追溯性必不可少。这些生命周期通常都是并行存在的，但是团队的交互作用有限，构建物的可追溯性也有限。
- **尽早以及更频繁地测试：**基于测试大数据及分析应用与服务所需的数据集的数量与种类，创建真实的大数据测试数据集可以显著增加项目的额外时间。诸如 IBM InfoSphere Optim 这样的测试数据管理工具简化了测试数据集的创建与管理，屏蔽敏感数据，自动对比测试结果，不需要再为维护用于测试的多个数据克隆而耗费资金与精力。
- **测试数据与服务虚拟化：**应用需要大数据仓库以进行充分测试，组织针对大数据仓库开发应用与服务，以下方面可能会遭遇挑战：
  - **测试需要针对还不存在的数据仓库进行。**比如，其中可能涉及对社交媒体应用的测试，但社交活动还未发生。
  - **测试需要针对无法使用的数据仓库进行。**比如，数据可能是还无法使用的外部资源，或者仅用于测试目的，其访问费用太高。
  - **测试需要访问无法以其原始格式进行访问的数据。**这可能包括涉及个人信息的医疗或者财务数据。

这些情况下，为了执行持续测试，需要虚拟或模拟所需的测试数据。这可以通过利用 IBM Rational Test Virtualization Server(之前为绿帽)或者 CA Service Virtualization (之前为 ITKO LISA) 这类的服务虚拟化工具模拟测试数据库来实现。这些工具通过在测试虚拟服务器上运行的虚拟呈现来模拟数据库。然后，测试可以针对这些虚拟的数据库来进行，

不需要利用真实的数据库，甚至不用出现真实的数据库。

部署、运维与优化

鉴于大数据及分析应用与服务的性质，这些应用与服务的部署、运维及优化有所不同。

- **持续交付：**对于大数据及分析应用，推荐使用增量开发方法。增量开发指的是一种阶段性和计划性方案，开发过程中，系统的各个部分依照不同的时间和速度开发，全部完成后集成。通过允许以不同速度开发的组件在准备好部署时随时进行部署与测试，而不是按大爆炸方法将所有组件一起部署，自动化持续交付实践的应用使增量开发得以实现。
- **数据库配置管理：**对于拥有动态数据库及大规模分析需求的组织，数据库配置管理正成为一项挑战。需要配置的数据库由不断衍生的多种数据库类型混合组成——关系数据库、对象存储库、NoSQL 数据库、Hadoop DFS、图形数据库、数据仓库、数据集市等。数据库的配置需要进行管理、版本更新以及控制。这些配置要随着底层数据架构版本的变化还有访问数据应用的变化而更改。

  多个数据库间的数据库配置管理是一项实践，用以管理多个数据库实例间的配置设置清单并防止出现配置漂移。然而，对于某些数据存储技术，这些配置存储在单独的 XML 文件中，对于其他文件，则需要通过 API 进行编程访问与管理。像 IBM UrbanCode Deploy 这样的自动化部署工具有最流行的数据库插件，提供管理及存储多个版本配置的集中化的工具，并且通过管理在哪个数据库实例应用哪个配置设定的版本，来应对这些挑战。
- **团队与文化：**和 DevOps 方案面向多种技术特性的变体一样，团队合作及文化导向也是至关重要的。就大数据及分析应用与服务而言，除了传统意义上的利益相关者以及操作人员，还有一些大数据特定的利益相关者与操作人员：数据库管理员、数

据库开发人员(MapReduce、Spark、R 等)、ETL 专家以及商业智能(BI)专家等。所有这些利益相关者及操作人员都需要参与 DevOps 实施并使流程得以被采纳，其工具也要集成到交付流水线的工具链中。

鉴于云计算及云服务快速发展的特性以及对于创新驱动的影响，关于云的相关方案将在下一章进行介绍，题为边缘创新 DevOps 方案。

4.5　总结

这章的篇幅很长。如果把所有的内容都放入本章所涵盖的主题上，可能这就是一本书了。这并非没有道理。这里所涵盖的实践及能力是 DevOps 的核心。如果一个大型组织想要在整个组织范围内实施 DevOps，就需要以本章内容作为出发点。现在，请立刻提醒自己，本章中的所有方案都不适用于你当前的组织。这本“手册”旨在列明所有的潜在方案。组织需要利用本章以及第 3 章的信息构建自己的手册，选择所需的方案。任何一个大型组织都不是完全统一的或同质的，不同的部门、程序甚至项目，都需要多种不同的方案手册。最终，还是没能完全列举。本书给出的只是专注于不断优化的方案。第 5 章介绍的方案专注于创新。第 6 章介绍的是推广 DevOps 的方案，第 7 章的方案是关于如何引领 DevOps 变革的领导力。所有这些章节都将为组织的手册提供方案。

因此，再来回顾一下本章内容吧。讨论是从一些主题开始的，接下来的章节也将是这样。这些主题提供了组织想要引入的思维及工作方面的核心能力与转变。所介绍的三个主题为：

- 缩短周期时间
- 缩小批次规模
- 建设正确文化理念

这三个主题是相互关联的。组织、组织中的每个项目，甚至项目的每个阶段，目标都是要缩短周期时间(或者像精益实践者那样称之为前置时间)。缩小批次规模与之密切相关，是缩短周期时间的驱动力，

还可以降低风险并促进计划与变更管理更好地进行。最后却也是非常重要的一点是，如果组织文化不改变，所有这一切都是徒劳。第 6 章及第 7 章中会更多地讨论文化。

介绍的所有方案中都有这些主题贯穿其中。这些方案是为了实现这三个主题所承诺的结果需要采取的一系列行动及做出的改变。

- 建立衡量标准及关键绩效指标
- 敏捷实施
- 集成的交付流水线
- 持续集成
- 持续交付
- 前移——测试
- 前移——运维人员参与
- 持续监控与反馈
- 发布管理

对于任何参与 DevOps 实施有一段时间的人而言，这些方案中所包括的条件及实践都不是唯一的。这里所介绍的方案重点是针对大型的、分布式组织，这些组织具有明显的文化惰性，现有流程及管理机制僵化。

本章在最后部分审视了一些特定技术领域，说明了要如何根据这些技术上的差别对各种不同 DevOps 实践及能力进行调整。

- 移动设备 DevOps
- 大型机 DevOps
- 物联网 DevOps
- 大数据及分析 DevOps

下一章，将继续添加专注于创新的主题及方案。

第 5 章

DevOps 驱动创新方案

费德勒的新接发球战术 SABR

原来是可以让因循守旧的人接受新事物的。

今年夏天，网球迷们接受了这位 34 岁网球传奇人物罗杰·费德勒的新战术。这个战术被称作 SABR，是在对手的第二个发球局中，向前移动，试图通过半截击来控制速度并挫败对手。每当有人听到这个词时，首先就会问：什么是 SABR?

这是首字母缩写词:

- Sneak(偷偷)
- Attack(攻击)
- By(由)
- Roger(罗杰)

运用此战术，费德勒在辛辛那提大师赛中取得了统治性胜利，除了第四轮击败约翰·伊斯内尔的比赛外，在其他的美国公开赛中他也一直运用这个战术。这个战术动作有助于他打出破坏性长球后的前场站位，同时让像理查德·加斯奎特这样的对手失去平衡。

—— Chase，2015

板球运动的发展及创新可能超越了其他任何现有的运动项目。虽然通过改变规则或装备、引入 Federer's SABR move(费德勒偷袭)这类的创

新性战术，其他运动项目也在不断发展，但还没有一项运动能创造出全新的比赛形式。板球最初的比赛形式称为测试比赛，每场比赛历时 5 天！双方各打两局，比赛似乎会永远进行下去。然而，随着时间的推移，板球赛事进行了重大创新，以吸引那些没有时间或耐心观看一场历时 5 天的比赛的观众。1971 年，一日的比赛形式应运而生。整场比赛一天结束，每队只打一局。比赛双方各有 50～60 个回合(同一个投手只有 6 个有效球)，由此控制比赛时间。以电视为中心的新一代观众对比赛形式提出了更高的要求，他们期待能有一种更加令人兴奋的形式，2003 年另一种创新的比赛形式：称为 Twenty20 或 T20 的 20 回合赛由此诞生。这些比赛大约三个小时就可以结束，因此一个晚上就可以完成一场比赛。

这些创新，尤其是 T20 形式，导致了全世界板球观众的数量激增。其中一个世界上规模最大的、观众最多、费用最高的体育锦标赛便是印度板球超级联赛(IPL)年度 T20 联赛。2009 年，其为期 8 年的独家转播权卖出了 1.63 亿美元的价格(kalavalapalli，2016)！市场推动了创新的需求，并为此付出了代价。

这些新形式的出现也迫使球队调整架构以适应每一种比赛形式。而今，虽然国家队仍然坚持三种形式的比赛，但职业板球俱乐部几乎全都采用 T20 的比赛形式了。像英国这样的板球运动主体国家已经意识到每一种比赛形式需要配备不同的球队。历时较短的 T20 和单日赛需要年轻的、更具激情的球员；在这种比赛中，强壮及敏捷胜过一切。历时 5 天的测试赛(如今测试赛已很少进行)则需要经验丰富的球员，要具备足够的战略知识与能力才能应对耗时 5 天多的比赛。在这种情况下，战略与防守技巧才是最为重要的。短时赛注重战术，长时赛则更注重战略。无论什么情况下，要赢得比赛，都需要配备不同技能的选手并采用不同的战术。

5.1 优化创新

虽然本章的重点是创新，但优化的必要性也不容忽视。如第 4 章所

述，大多数以创新为重点的应用都是用来为组织提供创新性新技术驱动能力的。其中包括：

- **新业务服务**。这些都是截至目前组织未向其用户交付过的全新服务与功能，例如，银行向其客户提供点对点的转账服务。
- **新商业模式**。这些新商业模式都是要为组织引进迄今为止还未能实现的货币化服务，比如，通过暴露 API 的内部使用方式，投资银行的交易柜台可以为其业务合作伙伴提供希腊计算(金融风险评估)服务。
- **用户参与的新模式或平台**。这是与用户交互的新技术。数字银行是个很好的例子，客户可以通过手机享受几乎所有的银行服务，从账单支付到支票存款再到贷款申请。
- **新市场**。这是为组织开拓全新的市场。还以银行为例，银行引进点对点的转账服务以吸引新客户，比如需要银行服务却不想使用传统银行的千禧一代。

初创公司没有现成的或遗留的系统用来交付业务功能或服务，如果是为了满足这种新公司的创新需求，那么讨论范围仅限于以创新为重点的应用程序即可。然而，大多数组织，从大型企业到小型公司，都已经在利用现有的软件系统为客户交付核心业务服务并进而交付业务价值。这类组织中，创新应用几乎总是依赖于这些预先存在的遗留系统所交付的核心业务服务。银行提供创新的点对点转账应用还需要现有的遗留系统去提供诸如身份管理、欺诈检测、反洗钱服务等。如果这些核心系统已经为银行应用的其余部分提供了服务，为什么银行还要创建新的服务并交付给以创新为中心的新应用呢？这种依赖关系没有商量的余地。因此，这是每一个创新性应用中都具有的架构依赖。

这些后台系统可能会拖累创新性应用的速度。如果创新应用要求后台服务做出改变以满足其需求，那么遗留应用交付变更的周期时间越长，就越会拖慢创新应用的交付时间。为了实现创新，并解决创新性应用的交付速度与遗留应用交付速度不匹配的问题，优化交付流水线及能力，以更加精益、高效的方式向后台系统交付变更，势在必行。优化已然成为创新的前提条件。

后台系统需要优化的另外一个原因是资源。多数大型组织都把主要的 IT 资源用于系统的运行与维护。如果所有可用的人力与财力资源全部都用于运行并维护现有的应用，就无法投资创新了。优化后台系统以释放出资源投入创新势在必行。

5.2 Uber 综合症

有一种新的现象正在驱动着商业世界里的创新需求。那就是 Uber 综合症。组织无论大小，都迫切需要采取行动以应对无孔不入的竞争对手所带来的威胁，正如叫车软件 Uber 在出租车行业中所做的那样。Uber 的创始人特拉维斯·卡兰尼克并非出租车行业出身。在创办 Uber 之前，他曾创立了一个称为 Red Swoosh 的点对点文件共享公司，这间公司 2007 年出售给 Akamai(Roettgers，2008)。

Uber 式的恐惧并不仅限于那些易受攻击的行业。IBM 商业价值研究所的最新调查发现，60%的全球首席执行官预计他们的下一个竞争对手将来自行业外部(IBM 商业价值研究所，2016)，这些竞争对手拥有全新的创新性商业模式，也许只需要配备一个移动应用程序即可，就像 Uber 那样。从金融服务业到零售业再到公共部门，对大多数组织而言，这都是极具说服力的理由。创新的需求基于一个主题：颠覆或被颠覆。

创新的需求实际上推动了针对创新速度而设计的那些新技术平台与流程的应用。DevOps 与基于云的平台即服务显然是两种不同的例子。创新的需求又会反过来推动组织去优化其遗留的应用交付能力。组织需要这样做，以释放人力及财力资源，投资于创新。组织只有这样做，才能确保传统的缓慢交付流程不会阻碍其高速创新的能力。答案仍旧是 DevOps。

5.3 创新与技术的角色

创新大多由初创公司驱动的当今世界，关于技术在驱动创新中所发

挥的作用，存在着普遍误解。技术本身很少有创新。没错，的确有一些技术创新者，比如苹果、IBM、Netflix、Salesforce.com，当然还有特斯拉，这些公司的业务核心就是全新的、创新性、突破性的技术。但对于绝大多数的创新型公司而言，从制药企业再生元(Regeneron)、服装公司安德玛(Under Armour)、电子商务公司阿里巴巴、酒店万豪国际，到政府机构英国税务与海关总署，技术只不过是其向用户交付业务服务的一个平台。IT 不需要再提供颠覆性、创新性的技术，而是要提供精益、高效、可预期、稳定的平台用于交付颠覆性、创新性业务服务。如果业务服务需要开发创新性、颠覆性的技术，那么 IT 才需要提供技术。不应该为了交付技术为最终目标而交付创新技术，交付创新技术的目的应该是促进业务创新。

5.3.1　商业模式创新

你的客户是谁？

他们构建了一个很棒的改变游戏规则的应用程序。全世界的人都有可能成为这款应用的客户。它解决了一个重大问题：客户服务电话的在线等待时间。大概每个成年人都有过等待接通客服电话的经历，等待时间没有几个小时，至少也有几分钟。打电话给客户服务必须通过一系列菜单选项后才能转接到正确的部门，然后等待开始。FastCustomer 却是让客服人员来找你。你只需要在 FastCustomer 应用上选择想要接通的公司及部门，然后继续生活。电话另一端的人工客服准备就绪后，应用就会打电话联系你。(Martin，2014)

问题是，初期的火爆销售过后，应用就停止了销售。每当应用程序通过媒体或博客宣传时，就有销量产生。在其他时候几乎没有销售。他们不明白为什么会发生这样的事情。问题是明确的，市场也是认可的，应用程序以一种简单、有效的方式解决了问题。到底问题出在哪里？

答案是，他们销售给了错误的人。应用程序的最终消费者是用户。他们对消费者的识别有误。消费者本该是愿意付费解决问题的人。个别消费者感受不到足够的痛苦，以至于愿意去寻找解决方案并为此支付费用！真正的消费者是那些有意提升客户体验的公司，他们愿意付

费购买服务以缩短客户等待时间。FastCustomer 改变了商业模式。他们免费提供应用程序给消费者(用户)使用并开始向公司兜售服务。这些公司愿意付费购买应用并集成到自己的呼叫中心系统，他们甚至愿意付费在其网站安装“联系我”的按钮。不再需要等待时间，FastCustomer 已经做到。

5.3.2　商业模式实验

进行创新实验不仅仅是为了确定应用中哪些功能合适，或者哪些网页的点击量更高，也是为了找到正确的商业模式。如前面案例所述，相同的产品、采用不同的创新性商业模式，就可能导致成功与失败两种截然不同的结果。我们来研究一下如何才能发现新的商业模式。

如第 3 章所述，亚历山大·奥斯特瓦德和伊夫·皮尼厄在其创新性著作《商业模式新生代》中列举了一些公司，比如家用咖啡机供应商 Nespresso，和前面 FastCustomer 的例子一样，他们也必须尝试各种不同的商业模式以找到最适合自己的。这种模式必须使其能够将产品推向市场并为客户提供最佳商业价值，从而使公司能够在市场立足并取得成功，他们并没有改变产品，而是尝试各种不同的商业模式，以找到产品与市场的最佳匹配模式。

《商业模式新生代》的编撰模式也是创新性的。作者没有去采访来自世界各地的数十位商业领袖，以开发、提炼并验证他们的想法及观点，而是在社区构建网站 Ning.com 上建立了一个社区，并开始为本书的联合创作者筹集费用。价格从 24 美元一路飙升到 243 美元。此刻，已经有来自 45 个国家的 470 人愿意为这本书贡献力量，审阅并重新撰写内容，贡献研究案例，并帮助设计书中的大量插图。结果这本书畅销了，成为世界各地的创业者及创新人士必读的畅销书，并且开创了全新的图书写作模式。(Wilson，2010)

从 IT 角度而言，当人们想到创新时，想到的总会是这样一款应用，其拥有酷炫的新功能、尖端的技术、精心设计的客户体验，以颠覆行业为目标。然而，更多时候，创新是通过颠覆商业模式而非技术来实现的。例如，虽然 Uber 已经拥有一个非常好且便于使用的移动应用程序，但

并没有因为应用的性能或便于使用而取得成功。Uber 的创新在于商业模式，并且为了实现创新，卡兰尼克和他的团队尝试了各种不同的商业模式(而且正在继续这样做)。他们首先提供一个叫车应用程序，以此找到行程范围内闲置的豪华轿车并以极低的成本租用。现在这款应用仍以 UberBLACK 的形式存在。今天你所知道的 Uber 核心业务是：非专业司机利用私家车提供付费服务。然而，这个核心业务正是通过不断实验发展而来的。

这里，IT 的角色就变成实验商业模式的平台。几种业务模式可以同时根据需要引入并回滚。通过收集并分析快速反馈结果，去评估每一次实验的效果，以决定是前滚动升级还是回滚。通过这些反馈可以了解到哪种背景的用户使用哪种模式，不使用哪种创新不是技术本身的创新。客户正在呼叫出租车，当他站在雨夜的街角等待乘车时，他们并不关心使用了什么技术，也不会在乎是哪个地图 API 正在显示其乘车位置，或是该应用集成了移动后台的某些服务。IT 的目标是提供一个精益、高效的服务平台，商业模式能够在这个平台上进行交付并针对实验需求做出迅速变更。迅速地进行实验变更。

5.3.3　用户参与模式创新

另一种常见的创新途径是探索新的用户参与模式。这些新模式可能是向现有客户交付新的业务功能，也可能是为现有的功能开拓新市场。在发掘新的参与模式方面，实验同样发挥着关键作用，

来看看另一个真实的案例。通过扩展销售商品种类，快餐连锁公司麦当劳正在尝试一种新的方式与顾客互动。他们开发了一种全自动化便利店，直接放在顾客下班回家路上。如图 5-1 所示，自动商店，配有自动售货机销售便利店可以销售的所有商品：从新鲜食品、牛奶、零食到 DVD，一应俱全。这些商店甚至被制作成可移动的，可以轻松地移动并更换实验地点。遗憾的是，实验失败了。显然，并非所有东西都可以通过自动售货机销售，至少牛奶或新鲜食品不可以。

高管们并没有放弃整个概念，而是着眼于其中有价值的部分继续尝试。DVD 的销售与租赁是可行的。人们会毫不犹豫地从自动售货机租

赁 DVD。谁还需要人工来检查 DVD？于是，公司决定撤掉所有其他商品的自动售货机，只保留 DVD 自动售货机。红盒子(Redbox)、DVD、蓝光和电子游戏光盘租赁亭公司就此诞生！



图 5-1 红盒子杂货亭，华盛顿，2002(Imgur，2013)

本例中，实际商业模式的核心并未发生改变，还是提供自助售货亭，向顾客销售产品。改变的是提供交付业务服务的技术平台。它通过实验加强并改变了与顾客互动的方式，并从一个销售各种商品的售货亭变成小型的 DVD 光盘的售货亭。该平台后来进一步完善，还将蓝光及电子游戏光盘也纳入销售范围。平台交付团队要足够敏捷，并能够对平台做出调整(在这种情况下，是显著减少)，以适应业务的方向以及客户参与的模式，并向客户提供业务服务。

IT 团队对平台背后的技术负责，确保平台灵活敏捷并能够适应变化。还要负责确保将数据显示仪表盘嵌入平台中，以提供对业务的持续反馈。因此，企业才得以发现，虽然他们的综合售货亭没有成功，但 DVD 租赁业务却是赚钱的。所以，他们选择了放弃所有除了 DVD 租赁以外的业务，反而取得了更大成功，实验发挥了作用。

正如最初的红盒子售货亭项目，实际上，仪表盘要能够反映出哪个实验是成功的，就像能够测量出哪些商品带来收益，哪些商品没有带来收益一样。在更复杂的情况下，一个应用程序可能同时运行多个实验，这就需要大量仪表盘来收集各种实验的数据，然后对数据进行深入分析以评估每项实验的成果，并提供正确的反馈给业务。通过数据分析来评估每个潜在的增强实验的独立结果，为企业提供正确的反馈。接下来讨论运行实验的必要性时会进行更为详细的说明。

创新与体育客户体验

客户体验，或 CX(Customer eXperience)，是当今的 CMO(首席营销官)们以及所有营销人员面临的最热门的问题之一。提供最佳的客户体验在任何时候都是成功的必要条件。

在职业运动领域，同样适用。以前，球迷的体验完全取决于运动场上的情况。今天，同所有业务一样，以客户为中心并提供最好的客户体验也已经成为各体育运动队的必然要求。

过去大约一年时间，两支美国橄榄球队都在提升他们比赛的客户体验。印第安纳波利斯小马队利用移动应用为球迷创造了身临其境的体验。费城老鹰队与松下合作，在达阵区，为球迷提供高清分辨率的大屏幕，在球场周围也布置了很多高清大屏幕。

—— Olenskie，2015

5.4　核心主题

第 4 章的各种不同方案中都贯穿着一些核心主题，现在我也将介绍一些核心主题，这些主题构成了多种创新型方案的基础。具体如下：

- 实现多级 IT
- 构建正确的事物
- 进行实验
- 实现反脆弱的系统

5.4.1 实现多级 IT

多项目运动员

星期四晚上举行的 2016 美国橄榄球联盟选秀的第一轮比赛中，高中时参加过多个运动项目的球员占了绝大多数。

31 位入选球员中有 28 人此前曾是多个项目的运动员，其中就包括一号新秀 Jared Goff，他在加州格林布瑞的马林天主教高中读书时参加的是棒球和篮球项目。

Goff 是铁人三项运动的 12 强选手之一。与四分卫 Carson Wentz(棒球、篮球)，接球手 Corey Coleman(篮球、田径)还有前锋 Jack Conklin(篮球、田径)的情况类似。

——Spiewak，2016

不需要争论年轻运动员是否应该从事多项运动、发展多样技能，在 IT 世界里，支持周期时间与交付速度各不相同的多条交付流水线的能力，催生了多级 IT。多级 IT 不是一个可选项，而是必须接受的事实。如第 4 章所述，现实世界要求 DevOps 跨交付流水线实施，以支持不同交付团队所采用的不同速度及技术平台。大多数业务系统都需要多个应用程序共同提供服务，因此，流水线间的协作必不可少。要确保每个应用的交付与发布都能与相互依赖的其他应用进行协作，识别并理解不同交付流水线交付的各种不同服务之间的架构依赖至关重要。此外，确保这些交付流水线间的可追溯性也是十分必要的，这样才能有效捕捉并分析每一条交付流水线的状态，并主动识别问题、延迟或者应用或服务交付及发布计划变更所产生的影响。在第 4 章中已经介绍过，如图 5-2 所示，有 5 个特殊的领域，需要对多级交付流水线进行集成，同时各条交付流水线的工具也要实现标准化。各条交付流水线配备标准化或集成的工具，能确保多个多级交付流水线都能正确实施计划、架构设计、可追溯性以及状态报告。我们来看一下这 5 个触点的详细情况。

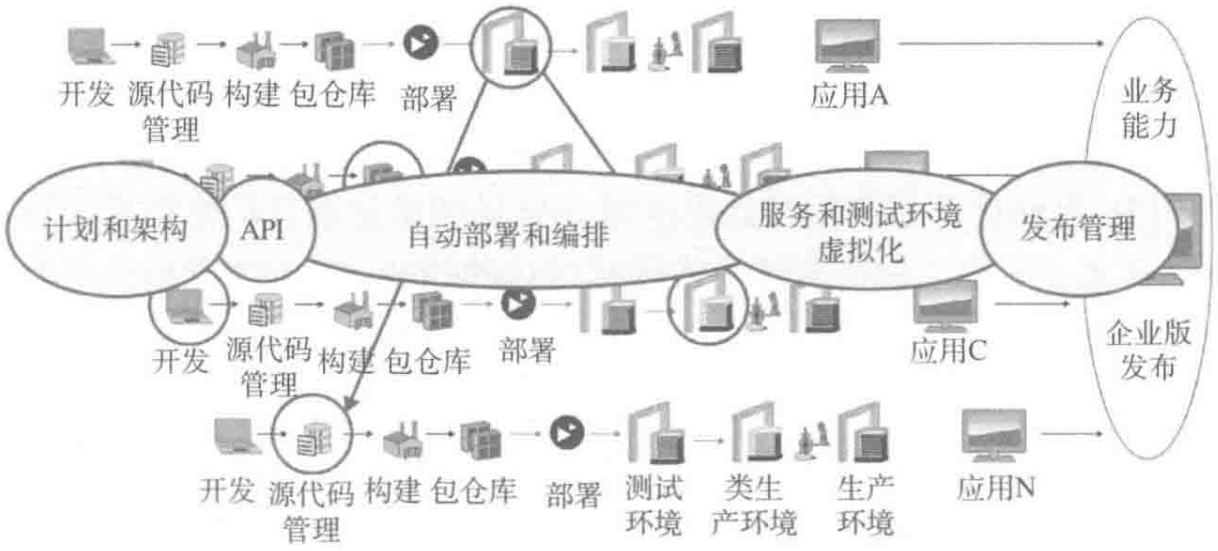


图 5-2　多级 IT 触点

(1) 计划与架构。这些应用程序的交付与发布计划都不是孤立存在的。需要记录并理解各种应用之间的架构依赖关系。只有识别出这些依赖关系，才能正确评估应用或服务彼此之间的依赖情况。一旦了解了这些依赖关系，就可以为每项应用或服务制定与相互依赖的其他应用相匹配的适当开发、交付及发布计划。对于特定的应用而言，将所有可用的资源都利用上，以期在某一特定日期发布，这毫无意义，只是发现其所依赖的服务几周内都无法使用而已。这些资源本来可以在其他地方得到更好利用，而且可以提供更多的时间来发布应用。一旦计划确定，就必须使用标准化的工具集来管理项目任务以及计划工作项。交付流水线上的所有操作人员必须对交付计划、分配给他们团队的工作项以及项目需求列表具有可见性。操作人员需要多大程度的可见性取决于项目之间的依赖性程度。

(2) 应用程序接口(API)。基于应用之间的架构依赖关系，各类应用的开发及发布周期实际上取决于其他的应用。为了使事情更复杂，这些架构依赖甚至可能是版本相关的。也就是说，一个应用的特定版本只能与另一个应用的某些版本进行通信。应用之间的点对点集成随着每个应用版本的更新而变化。通过解耦应用程序，可以减少架构的依赖性。引入明确定义的 API 来实现应用或服务间的通信，可以实现减少架构依赖的目标。这些 API 使应用不需要了解其他应用的实施细节，便可进行开

发，还能够解决每次版本更新时，点对点集成都要随之变化的问题。API 通常不会随着时间推移发生太大变化，从而使应用程序之间的架构接口也保持不变。设计并实施好的 API 是本章后面部分要讨论的主题。

(3) 自动化部署与编排。第 4 章中已详细讨论过自动化部署。这是一个需要在所有交付流水线上实现标准化的领域。根据需要将应用部署到适当环境中的能力是必不可少的。如果不同的交付流水线使用不同的部署工具，那么部署时就需要各条交付流水线进行协作，这就增加了部署复杂性。标准化部署流程及工具，只需要创建单一的部署流程，就可以根据需要部署各个交付流水线及不同环境中的所有组件、应用以及服务。这同样适用于环境的配置与管理。单一的平台能简化配置与编排流程，并能够根据需要配置多个环境。

而今，使用单一的技术或工具实现部署自动化或环境的配置与编排并不总是行得通的。然而，目标就应该是尽可能地少使用工具。每个技术栈只使用一个或最多两个标准化工具是一个很好的奋斗目标。

(4) 服务虚拟化与测试环境。各种应用与服务做好测试准备时，他们所依赖的其他应用与服务可能还没有准备好，因为他们的交付周期及发布计划可能有所不同。很多时候可用性是无法满足的。此外，每项应用或服务的测试环境以及测试数据也无法保证需要时总是可用。虚拟化服务、应用、环境以及数据源解决了这个问题，使每项应用或服务不需要依赖其他的应用或服务即可完成测试。这些方案已在第 4 章中详细讨论过。

(5) 发布管理。由多项应用与服务所组成，并通过多条交付流水线交付的业务系统在发布时需要所有相关的流水线共同努力来协调执行交付及发布计划。传统的组织依赖大量的电子表格，其中包含正在开发与交付的所有组件的数据、组件相互之间的依赖关系以及在各自交付流水线中的状态。这就涉及应用与服务的跟踪及管理，因为在发布到生产环境之前，它们需要通过验证所需的各种质量测试，他们要跟踪并管理所有依赖性应用或服务的所有集成点及测试。在所有交付流水线及技术栈中，使用标准化工具跟踪并管理每项应用程序或服务的发布是非常必要的。如果这个工具可以自动从项目和工作项管理中提取每条交付流水

线的状态信息，并且提取每条交付流水线所使用的自动化工具中的状态信息，则收益更大。

发展支持多级 IT 的能力是推动创新的必要条件。在交付创新的同时也需要速度。要在需要的时候有能力交付产品，并运行多个实验(接下来会讨论)。这就需要摆脱流水线相互依赖的限制，不能让其中速度最慢的交付流水线拖慢节奏。

5.4.2　构建正确的事物

过度训练综合症

运动员们想要提高成绩，就必须刻苦训练，这已不是什么秘密。然而艰苦的训练会使人崩溃，使人变得虚弱。反而是休息会让人更加强壮。身体机能的改善只发生在运动过后的休息时间。这种适应是对心血管和肌肉系统最大负荷的反映，是通过提高心率，增加肌肉中的毛细血管数量，增加肌肉细胞内的糖原存储和线粒体酶系统来实现的。恢复期间，这些系统构建了更高水平来补偿所承受的压力。结果就是你现在表现得更好了。

如果训练过程中没有足够的休息时间，那么改善就不会发生，也不会有稳定的高水平表现。如果训练强度大，又不安排充足的休息时间，那么成绩反而会下降。过度训练可以定义为运动员不断地加强训练，以至于无法恢复的状态。而“过度训练综合症”是指基于历时数周甚至数月的过度训练，心理、行为以及身体所产生的综合症状。运动员和教练也称之为“懈怠”或“无力”。这不同于一般的运动员在日常训练期间和运动过后的疲劳。过度训练甚至会导致恢复期之后的持续疲劳状态。

最为常见的症状是疲劳。这可能会影响到训练和休息。运动员也可能会变得情绪化，容易发怒，睡眠模式改变，情绪低落，或失去对运动及竞争的欲望和热情。有些人会表现得失去胃口，体重减轻。身体的症状包括持续的肌肉酸痛和伤病率的增加。

—— Jenkins, 1998

过度训练可能会导致运动员“情绪化、容易发怒、睡眠模式改变、

情绪低落”，有一件事情，也一定会导致 IT 人员，尤其是开发人员深有同感，那就是满怀激情地投入工作，为项目付出了很多的时间和精力，最后却发现构建的产物是错误的。不论是完整的项目还是仅仅是代码模块，只要解决的问题不对，就无法使用，就是对资源、个人付出的时间与精力的极大浪费。公司倒闭是因为他们将整个未来投资于市场不想要或不接受的产品上。想想那些失败的互联网创业公司，从 Pets.com，到 eToys.com，再到 Webvan.com。大公司的 IT 部门随处可见这样的项目，因为无法吸引客户，这些项目压根没有问世，或者发布没多久就被搁置起来。在大多数情况下，产品功能与客户期望之间的不匹配在完成全部产品开发与交付工作之前是无法发现的，而是只在被用户拒绝的时候才能发现。

1. 精益创业

精益创业，是由埃里克·莱斯(Eric Ries，2011)在其同名的开创性的著作中提出的，代表的是另一种交付产品方法(IT 或其他方法)。在他的方法中，目标不是要构建一个完整的产品，而是通过持续验证用户心目中的产品形象来确保进行正确的产品构建。这种验证是通过开发周期实现的，每次先交付一小部分功能(参考小批次)，迅速从实际用户那里获取所需的反馈(参考最小化周期时间)，确保一直在构建正确的产品。可以看出，这些原则完全符合第 4 章中所介绍的 DevOps 原理。

精益创业的四项核心内容如下：

(1) 消除不确定性。关于产品交付有一个定义明确的方法论。目标不是不计代价将产品发布出去，而是应该以一种有条不紊、合规的方式来完成。这里再一次与本书所介绍的 DevOps 方案保持了一致。

(2) 更聪明地工作，而不是更努力地工作。最重要的问题不是“这个产品能否构建？”而是“这个产品是否应该构建？”在编写代码之前，让用户及潜在客户尽早参与进来，以验证产品的假设，在整个开发生命周期保持这种互动是确保交付产品正确的唯一途径。

(3) 开发最小可行产品(MVP)。最小可行产品(Minimum Viable Product，MVP)，在某些情况下，也称作“最小可行特性”，指的是向用户交付的，能够验证是否满足用户需求并交付业务价值的最小功能。最

小可行产品可以算是投资最少的学习途径。

(4) 验证学习成果。精益创业的进度单位就是验证学习成果。团队花费时间持续为其用户寻找正确构建物，就能确保在发现构建偏离用户需求时可迅速做出调整。精益创业将其称为战略调整(pivot)。

> **最小化可行产品**
>
> 应用的最小化可行产品指的是为了满足客户需求并在市场上测试创意，所需要开发的最小特性和功能集合。用非专业术语来讲，组合出一个 MVP，就像是建立一个球队，最小化可行产品就像是明星球员，一个天才的四分卫(或者足球队里面的 10 号球员)。这样的选手签约后(或者相当于一个杀手级的应用特性完成)，就可以开始考虑剩下的名单/其他应用的额外功能，这样有利于提高成功率。
>
> ——蓝色标签实验室，2016

精益创业已经成为一种运动，一种方法，不仅能为初创企业交付新产品，而且能为企业 IT 部门交付高质量的 IT 系统。无论终端用户是寻找下一个“网络改变世界”应用的千禧一代，还是希望更好地了解雇主所提供健康保障的雇员，又或是为长期、忠诚的客户寻找合适产品的销售代表，精益创业的方法都能确保不会发生特性与功能的过度开发。它确保交付团队构建正确的事物。

2. 设计思维

设计作为一门专业学科在过去 30 年间经历了巨大的发展变化，实践上，已从重点关注审美风格，转变成完全明确的 以“用户”为中心(指的是：使用产品或服务的个人或一群人)，主要关注用户的希望、意愿、挑战及需求。

通过与用户建立同理心，设计师能够更好地满足用户需求。

这种以用户为中心的方法被称为“设计思维”，可以用来帮助设计师和其他人解决更为广泛的复杂的商业及社会问题。

—— IBM Design，2016

设计思维有三大核心原则，这些原则能够确保构建正确的事物：

> **(1) 关注用户成果：**每天，用户都依靠我们的解决方案来完成工作。成功不是以我们所交付的特性和功能来衡量的，而是取决于我们满足用户需求的方式。
>
> **(2) 多样的授权团队：**多样化的团队能比同质团队产生更多的想法，增加突破的机会。赋予他们专业知识和权威，让其把这些想法转化为成果。
>
> **(3) 积极重构：**一切都是一个原型。甚至是任何市场中的解决方案都只是原型。把每件事都看成是另一次迭代时，就有能力用新思维去解决老问题。
>
> —— IBM Design，2016

像 IBM 这样的组织已经将精益创业与设计思维原则融入产品交付管理流程中。通过 DevOps 实践，进一步提升了开发并交付正确的产品的可操作性。结果就是：他们以正确的方式构建了正确的产品。

5.4.3　进行实验

教练的实验

很多教练通过自身的经验以及观察其他教练的做法来提升技能与专业知识。然而，仅靠获取经验并不能保障教练的竞争力。经验与知识相辅相成，才能提高学习能力进而发展专业技能。教练需要知道如何通过经验来更好地学习。反思实践是一种主要的学习工具……。

教练一旦掌握战术，接下来就需要探索每一个战术可能产生的后果，并选择最适当的对策。在某些情况下，这种实验可能是假想的。他们可以向教练同行阐述自己的想法以获得反馈。在一个赛季结束后，教练们正为新赛季做准备的时候,假想的实验则可以付诸实践。实际情况中，教练可以在体育领域的实验中执行假设的方案，并回顾其影响。

值得注意的是，反思实践中的实验不同于试错练习。尝试并犯错

只简单地涉及做某件事情，失败了，再做其他事情直到找到最为有效的。与反思实验相比，这种方法是随机的、不可预测的。反思实验的理念是，从经验中汲取现有知识，并学会根据相关信息做出明智选择。这种方法更具预见性，考虑问题更全面，为运动员提供了更加有效的学习环境。

——法雷斯，2004

教练通过尝试新的战术来准备下一场比赛，以应对特别对手的挑战或者弥补在上一场比赛或练习中所出现的失误。同样，组织推动创新，也需要尝试新创意、新功能、新客户体验、新用户界面、新商业模式以及新技术。正如本章第一个方案中所描述的，有两个推动创新的概念是精益创业运动所推崇的：

- 最小化可行产品
- 快速失败

实验的目的是防止开发出完整功能的产品才能确定产品是否合适这样的情况发生，使用 MVP 进行实验，然后快速失败，就可以最终获得正确的产品。

交付最小可行产品与快速失败的方法都是为了快速实验而设计的。同时进行多项实验，并让注定的失败迅速发生以找到成功实验的速度决定着创新的速度。

A/B 测试

A/B 测试是一种常用的技术，通过同时运行多种不同特性的测试来进行快速的实验。测试的目标是随机向用户同时展示两个网页、两种特性或应用程序的两个版本，看看哪个接受程度高，或者效果更好。这个过程需要在应用程序中安装相应的仪表盘，以确保能够测量出用户与这两个版本互动的各种细节指标并加以比较。这种测试能帮助开发人员与业务团队确定哪些版本的特性可以保留并继续开发，哪些可以删除。

下面是一些利用 A/B 测试实现最大成果并对测试特性执行快速失败的核心实践：

- **定义成功的衡量标准。**哪些指标及衡量标准可以用来确定版本成功与否？出售的商品、注册的数量、点击次数、花费的时间、浏览的项目、页面退出(跳出率)、购物车丢弃率，所有成功或失败的指标都应该预先确定，并利用仪表盘进行测量。
- **同步测试。**要进行 A/B 测试，就需要将两个版本在相同的时间内分别提交给数量相等的不同用户群。
- **应用间版本一致性。**要使这种方法奏效并产生一致的结果，不同的应用就应该显示出同一版本的特性，而不仅仅局限于几个领域。
- **长期监控。**需要收集重要的统计数据集，以确保对两个版本的测试结果都能进行适当的分析。因此，测试应该运行足够长的时间，以获取足够的用户来使用该项功能。长期监控还可以用来掌握趋势并消除异常值的影响。
- **多重 A/B 测试。**其理念是不断进行小的改变(小批次)，并迅速进行 A/B 测试，以确定哪些变更集更加有效。这样就可以将快速失败所产生的浪费降至最低。

运行 A/B 测试，交付平台和生产环境中必须具备特定的关键能力：

- 将应用程序的两个版本部署到两组服务器上，分别将一半的用户路由到对应版本的能力。这需要针对服务器集群进行部署。将应用部署到所有服务器上的非黑即白全量部署模式将无法奏效。
- 将用户随机平均分配到每一组服务器的路由能力。
- 向前滚动部署能力，以利用成功版本替换失败版本的能力。

因此，快速实验需要一个技术平台，即用来进行实验的应用交付流水线。它必须能够快速地向目标服务器集群交付新的功能与特性，以便能够监控应用及用户的行为表现，并能够捕获正确的测试数据进行分析以实现快速失败。

5.4.4 提供反脆弱的系统

培养反脆弱的运动员

我刚从西雅图超音速队 2014 体育科学周末回来，会议主题是培养

反脆弱运动员。首先，超音速队的执行经理 Dave Tenney 讲述了培养反脆弱运动员这个概念的由来(太多比赛失利造成的伤害)，我们能从反脆弱能力最强的行业(航空与核能)里学到哪些经验教训，我们是否可以把一个脆弱的运动员变成抵抗力强的运动员(看起来如此)。

赛场上，超音速队在获取并分析运动及表现数据方面的能力很强，这是公认的事实，这些数据涉及恢复、神经状态准备、受伤和遵守行动计划之间的关系、睡眠、运动员的走动距离以及运动员的经验类型。我突然想到两个指标。

一个指标是用来测量所谓“速度载荷”和“身体载荷”的，这是目前的技术再次让我感到震惊的地方。他们不仅使用 GPS 测量运动员在比赛和训练中所经过的距离，而且还能测量出这些距离的性质。“速度载荷”代表直线运动(例如，前场冲刺)，是伸肌占主导地位的领域，在常规的比赛中比较常见。身体载荷则代表方向改变，倾向于后部肌肉链负荷，经常发生在小范围的比赛中。

另一个有趣的衡量标准是睡眠数据。他们用监视器跟踪运动员每天晚上睡觉的时间、起床的时间以及辗转反侧的时间。戴夫指出，俱乐部中表现最出色的运动员睡眠也最好，不仅睡眠时间长，有规律，而且睡眠质量也很好。最优秀的运动员平均都在什么时间上床睡觉呢？答案是晚上 9:44 和 10:15。

—— Cavin，2014

对运动员来说最大的挑战就是受伤。然而，运动员难免会受伤。可能造成伤害的处境是运动员生活的一部分。我指的不只是美式橄榄球这种需要身体接触的运动，或是体操跳马这样的高危运动，这些本身就是高速冲击的运动，还有一些看似“安全”的运动，如高尔夫或网球也存在受伤的风险，肌肉过度拉伸或用力过猛都可能导致运动员受伤。伤病对于运动员的影响很深远，可能导致退赛几分钟、结束职业生涯，甚至是永久性残疾或死亡。已有大量的研究来帮助运动员改善训练条件，使其能够在这些高压环境中茁壮成长，更具弹性，更快恢复，更加强壮，并使他们的身体能够抵抗脆弱。这也适用于 IT 系统，特别是用于进行

持续变更、持续实验的 IT 系统，这些系统要能够应对混乱，在服务器宕机、新系统永远在线的压力情境下正常运行，这就是反脆弱。

反脆弱(Antifragile)一词是由纳西姆·尼古拉斯·塔勒布创造的，他是一位期权交易商，著有关于随机性、概率及其对于市场及生活的影响这一系列丛书。他在《黑天鹅》(Taleb，2007)一书中首次提出了反脆弱的概念。书中，他论述一些罕见的事件，并非像人们想象的那样随机与罕见，就像股市崩盘。之后他还写了另外一本书，名为《反脆弱》(Taleb，2012)，在这本书中，他进一步扩展了反脆弱的概念，用其来描述既不脆弱也不强大的事物，事实上，混乱中也可以获益。

如果你正在读这本书的印刷版，就会发现这是一本很结实的书。即使扔到地上，也不会损坏。相反，如果你正在通过电子阅读器或平板电脑阅读本书，这些设备本身就很脆弱，扔到坚硬的表面上很有可能会损坏。这些都是大家非常熟悉的系统案例。塔勒布挑战了我们对于反脆弱系统的认知，他认为反脆弱系统是那些既不脆弱也不强大的系统，在有压力的情况下，他们反而会变得更强。所有的生命，大多数的有机系统在本质上都是反脆弱的。塔勒布举出了骨骼的例子。骨折后，如果处理得当，骨骼愈合的部位实际上会产生更大密度，因而比未损伤的骨骼更加坚固。疫苗本身就是利用了动物免疫系统的反脆弱性。人或动物注射少量的细菌就能产生相应的抗体，从而保护其不受细菌的感染。

人类的大脑是一个反脆弱系统的极端例子。如果你在阅读本书时(不论纸质版还是电子版)，碰巧发现书中的某些部分很有趣并给你留下了深刻印象，但你却强迫自己去忘记这部分内容，这种压力反而会迫使神经元加强这段记忆。你不可能有意识地去忘记一些事情，这样只会让你印象更加深刻，问问那些失恋的心碎少年吧。我们不妨来做一个实验：如果我要求你在阅读本书其余部分的时候，不要去想“跳舞的猴子吃香蕉”。由此，我创造了一个压力情境，让你不要去想起需要忘记的事情。在你继续阅读本书时，你的反脆弱的脑神经元会随机展示一个猴子吃着香蕉，跟随神曲手舞足蹈的画面。你绝没有办法阻止自己去想这样的画面。以后你再谢我吧……

总结：

- 易碎的东西在压力下会破裂或损坏
- 强大的东西不受压力的影响，他们不会因为压力而变强或变弱
- 反脆弱系统会因为压力而变强

5.4.5　IT 系统与反脆弱性

运维团队一直致力于使自己的系统更加强大。他们想让系统永远在线。他们希望系统不会受到任何可能遇到的压力的影响。他们希望能够预测系统行为，以便在可能导致系统宕机或只是性能下降的压力发生时，他们可以采取措施来解除压力情境。要建立真正强大的系统，就需要构建可预测的系统，以消除系统中的每一个故障源头。在当今这个充满不确定的世界里，这几乎是不可能实现的。目前的系统都不是静态的。系统中的服务器是实时动态分配与销毁的。他们使用的是来自多家供应商的多种服务，其中一些服务可能还是归属于其他的外部供应商团队所有，包括组织之外的第三方供应商。潜在故障点的数量使得故障预测以及缓解计划制定根本不可能实现。再加上开发团队不断创新，持续交付新版本的应用与服务，失败点数急剧增长。这些都带来额外的压力。加之，要实现真正的 DevOps 架构系统，特别是关注创新的 DevOps 应用就坚决不能采取这个方法，因为注定是会失败的。

另一个方法是使用反脆弱系统。系统假设故障必然发生。服务器会宕机。硬盘会损坏。网络流量会延迟交付。网络交换机会故障。第三方网络连接供应商会掉线。内存会不足。数据集会超出队列的处理能力。数据流会出现故障或者速度太慢。第三方供应商提供的服务会不符合服务等级协议(Service-Level Agreement，SLA)的要求。交付的应用会有缺陷。太多的用户将访问应用的全新热门功能。中间件会配置错误或未能根据应用要求进行调整。黑客会试图攻击服务。木马会用无效流量使应用过载。一个完全独立的应用可能会导致你需要的服务发生故障。另一个系统中的应用崩溃，会在系统内产生多米诺骨牌效应，从而影响到你的应用。人会故意恶搞，也会在服务中插入恶意后门程序。开发人员会在代码中插入善意的复活节彩蛋。人是天生就会犯错误的。

你需要构建能够应对这些混乱的系统。系统建立起来，必须能够处理预先假设的故障情况，而且要通过寻找替代服务来保证系统始终在线。下面是反脆弱系统的一些关键特性：

- **快速失败。**与快速失败的原则一致，构建反脆弱系统也是为了快速处理故障。最流行的方法是当服务器实例故障或不能正常工作的时候，不去修复，而是直接销毁并用一个新实例来替代，这样就不会影响到系统的其他部分。

  我们的目标是让系统永不崩溃，即使在维护或升级时依然可用。Facebook 从来不会公布这样的信息，称其“将在星期一凌晨 2 点到 4 点进行系统维护”；但下次登录时，版本升级已经完成。本章后面部分将讨论蓝绿部署等技术。
- **频繁故障。**如何才能为故障做好充足的准备，并快速处理？答案是让故障频繁发生。

混世魔猴与 NETFLIX 猴子军团

想象一下轮胎爆胎了。即使后备箱有备用轮胎，你可知道它是否充气了？你有换轮胎的工具吗？最重要的是，你还记得怎么换轮胎吗？要确保你能在大雨滂沱的午夜，处理好发生在高速公路上的爆胎事故，一个方法就是在每星期日的午后，将车开到马路上，在车胎上扎个洞，然后进行更换练习。这在现实世界中是最费时间也是成本最高的办法，但在云计算中却(几乎)可以免费和自动化地实现。

这是我们创建混世魔猴(如图 5-3 所示)时所引用的哲学，混世魔猴是一款工具，可以随机关闭生产环境实例以确保我们能够在不影响顾客的情况下处理常见的故障，这个名字来源于在数据中心(或云计算的分区中)放养一只配有武器的野生猴子的想法，在我们持续为客户提供服务的同时，它可以随机打破服务实例并咬断电缆。在一个工作日，我们运行了混世魔猴，整个环境被严密监控，工程师随时待命以解决任何可能出现的问题，我们最终还是发现了系统的一些弱点，并建立了自动恢复机制来处理这些问题。因此，服务实例在星期日的凌晨三点再次出现故障时，我们甚至没有注意到。

受到混世魔猴成功的鼓舞，我们又开始创建新的猴子军团，引入各种故障、探测异常情况，并测试我们服务的生存能力；虚拟猴子军团使我们的云一直保持着可靠、安全及高可用性。

—— Tseitlin，2011

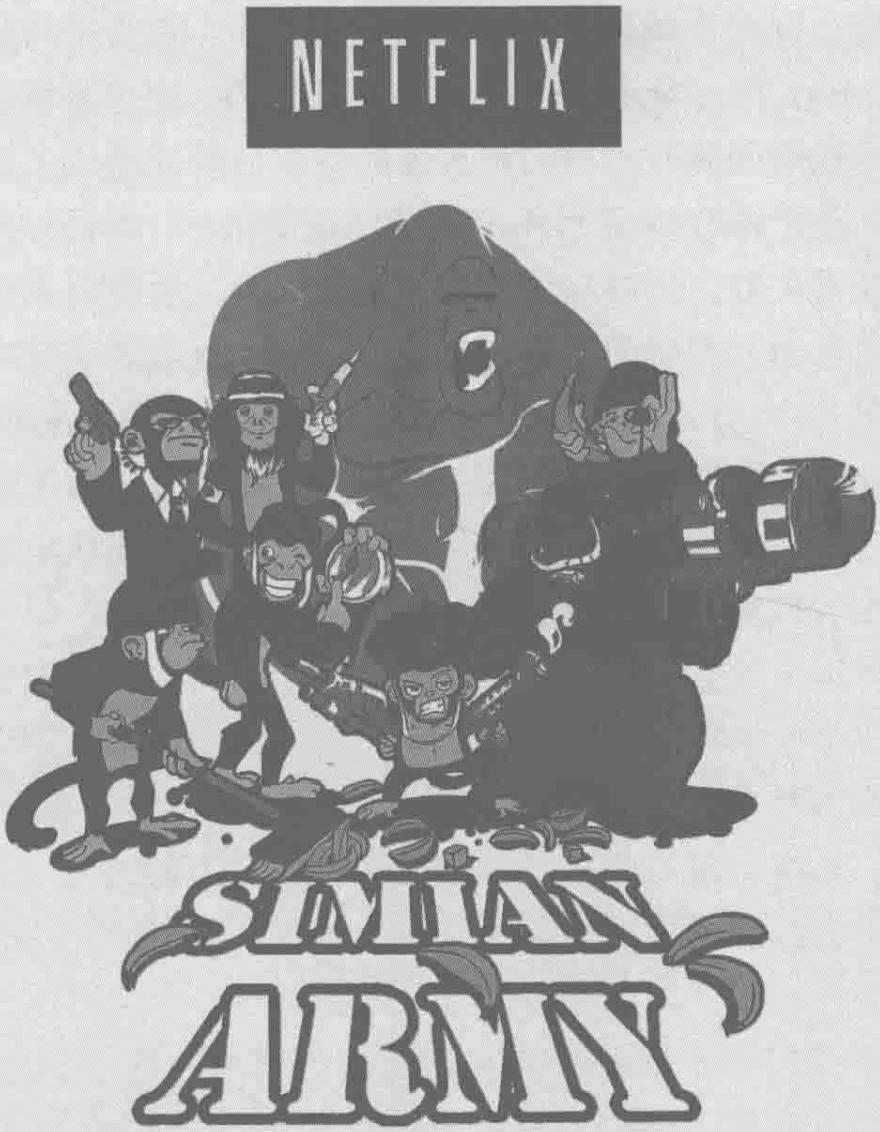


图 5-3　Netflix 猴子军团(图像来源：github.com/Netflix)

反脆弱系统需要能够不断地排解故障。遗憾的是，虽然所有 IT 组织都有处理故障的计划与方案，却很少进行测试。运动队必须在季前赛和赛季中不断练习，以完善其在比赛中想要采用的战术。这可能是他们一直在用的标准战术，也可能是为了赢得关键性比赛，改良过的突击战术。新奥尔良圣徒队在 2010 超级碗比赛中凭借一记出其不意的“赌博

球”击败了印第安纳波利斯小马队。然而，他们取得战术上的成功并非因为突击打得对方措手不及，而是因为他们经过多次练习确定真正有效后才把这个战术加入到自己的战术手册中(Triplett，2014)。

- **平均故障间隔时间到平均故障修复时间。**反脆弱系统要取得成功，就需要采用有别于强大系统的衡量标准，因此需要不同的指标。强大系统有一个传统的衡量指标，称为平均故障间隔时间(Mean Time Between Failures，MTBF)，用来测量其稳定性。平均故障间隔时间指的是相邻的两次故障或事故之间的时间间隔。反脆弱的 IT 系统不应该关注这个指标。其目标是快速失败和经常失败，正好与这个指标背道而驰。反脆弱系统认为故障必然发生，并且无法避免。所以，一个反脆弱系统应当关注的是其架构与运维模式的平均故障修复时间(Mean Time To Resolve，MTTR)。故障要多长时间才能修复，宕机的服务要多长时间才能重新上线？如何才能尽量缩短平均故障修复时间以减少宕机服务对其他服务以及整个系统的影响？如何进入这样一种运维状态，即尽管服务器可以变成“红色”，但服务始终保持“绿色”？
- **牲畜不是宠物。**系统的脆弱性实际上来自于对过分强大的渴望。系统管理员维护单独的服务器，使其始终处于在线状态，当服务器出现问题或是面临压力情况时，管理员会根据需要手动进行维护并提供支持。由于服务器独一无二的属性，管理员对待它们如宠物一般。在物理服务器上运行静态实例是很容易的。然而，在当今这个动态的世界里，这是无法实现的。自动化既需要大规模管理服务器，又要实时监控并解决发生的问题。他们需要像对待牲畜一样来对待服务器。牲畜没有名字，本质上都是相同的。可以使用可扩展的命名规则进行标记(cattletags.com，2016)。牲畜生病了，就会被宰杀。它们是批量饲养的。管理与维护也都是批量进行的。它们都有一个有限的、预先定义好的生命周期，并最终都是以牛排、汉堡包或香

肠的形式结束[①]。同样，服务器实例也需要使用可扩展的命名规则来命名，而不使用个性化的名字。他们需要彼此相同。需要批量进行监控与管理。当他们出问题时，就要被销毁，并由新的实例取而代之。而且，他们需要有一个预设的有限生命周期，以确定其配置及解除配置的方式(McCance，2012，Bias，2012)。抱歉，不再有名叫“黛西”的奶牛住在牧场。(如果有，那也是农场主的宠物。)同样，在你的数据中心，不再需要名为 midnight.rational.com 的服务器。

关于如何利用云计算技术设计并交付反脆弱系统，将在本章后面的内容中进行讨论。

接下来，我将讨论一些针对驱动创新商业意图的 DevOps 实施方案：专注于边缘创新的 DevOps 应用与服务。

5.5　方案：构建 DevOps 平台

美国的学校：1000 多位奥运选手的平台

对于许多全国大学生体育协会(NCAA)的学生运动员来说，奥运奖牌的梦想近在咫尺。

有 1018 名即将入学的、应届的、往届的 NCAA 学生运动员，将参加在巴西举行的 2016 夏季奥运会，他们代表着 107 个国家以及所有三个部门的 223 个 NCAA 成员单位。

在这 1018 名参赛运动员中，有 168 名应届学生运动员将参加 15 个项目的角逐，其中游泳和田径是最有代表性的项目，分别有 64 名及 42 名运动员参加。

加利福尼亚大学将派出 13 名应届学生运动员参加夏季奥运会，是参赛学生选手最多的一所大学。熊溪学校也取得了第二名的成绩，因为

① 我要对素食者以及纯素主义者表示诚挚的歉意。这里真的没有适合的体育方面的类比。我尝试过了。

> 在所有 NCAA 项目中，一共只有 40 名运动员参加夏季奥运会。南加州大学凭借 42 位优秀的选手挤掉了帕克 12 联盟的其他成员，这 42 位选手将代表所有木马人(南加州大学学子的昵称)征战里约，斯坦福大学凭借 39 名运动员跻进前三。
>
> —— Martinez，2016

本文的要点如下：

- 1018 名运动员都是应届及往届的学生。
- 他们代表着 107 个国家(在 2016 里约奥运会上，一共才有 206 个参赛国家)。
- 他们来自 223 所美国院校。

美国的高等院校已经发展成为一个真正的世界级运动项目及运动员的平台。这是一个广阔的平台，足以支持几乎每一项奥林匹克运动(而且，未列入奥运会项目的美式橄榄球还有冬季奥运会的体育项目未纳入统计)。这个平台还通过奖学金以及世界级的训练设施吸引了世界上最优秀的运动员新秀(Farrell，2008)。这是一个真正敏捷、弹性、可扩展且可靠的平台。

如何才能提供这样一个敏捷、弹性、可扩展、又可靠的 DevOps 平台呢？第 4 章中列举了构建应用交付流水线的多种方法。其中既包括兼容性技术的观点，也包括专有技术方法。我们将构建集成交付流水线的这些想法与本章前面所述的反脆弱系统的需求整合起来。因为本章的焦点是 DevOps 创新方案，所以，支持前面所列主题的 DevOps 平台需求是必须优先考虑的。为了便于理解，这里再重申一下这几个主题：

- 实现多级 IT
- 构建正确的事物
- 进行实验
- 实现反脆弱系统

尽管这样的平台也可以在底层基础设施上交付，但利用云技术进行交付则是必然趋势。只有云平台才能提供交付所需的敏捷性、灵活性、弹性、可扩展性以及速度，满足这些主题的要求。

第 4 章列出了进入交付流水线工具堆栈的一系列能力：

- 源代码管理
- 构建
- 持续集成
- 部署自动化
- 中间件配置
- 环境配置
- 环境准备

为了更好地理解所有这些能力是如何组合起来建立交付流水线的，我们来重温一下第 4 章中的应用程序交付流水线，如图 5-4 所示。



图 5-4　集成交付流水线

如图所示，有应用开发、测试及交付的工具，也有应用持续交付的环境(例如，开发、测试、类生产以及生产环境)。当构建 DevOps 平台时，应用的开发、测试及交付工具的安装并不十分重要。可以安装在传统私有云的硬件上，也可以安装在云计算平台上，这个选择不会对平台的性能产生重大影响。肯定会影响到工具的性能，但功能不会受到影响。还有一种情况是，这些工具根本不需要由组织来安装及管理，而是以软件即服务(Software as a Service，SaaS)产品的形式，由工具供应商作为托管工具来提供。多数流行的 DevOps 应用交付工具，如 Git、Jenkins、IBM Rational Team Concert、IBM UrbanCode 部署等，都可以作为 SaaS 产品，以付费订阅模式使用。

DevOps 平台至关重要的决策是选择目标部署环境，包括开发环境、各种测试环境、各种类生产及生产环境。当提及反脆弱环境的交付时，这些环境都需要具备反脆弱性，尤其是生产环境。

5.5.1 应用交付与反脆弱性

反脆弱环境的核心要求是消除维护、应用部署这类标准计划性操作所导致的停机时间。除非发生意外事故，否则环境服务以及运行在环境中的应用也不应该中断。

构建这样的高可用性环境需要以下几项能力：

- 建立冗余的环境服务
- 建立冗余的应用程序架构
- 建立将蓝绿部署作为持续交付流程的组成部分
- 持续监控环境与应用，以实时发现问题并尽可能解决

从 DevOps 的角度而言，要重点关注蓝绿部署的实施以确保应用部署到生产环境时不会发生停机。

利用蓝绿部署实现高可用性

Bluemix Garage Method 坚持每天持续更新，为了确保网站升级期间不发生停机，团队实施了蓝绿部署。一项新的功能推送到生产环境时，会被部署到一个实例中，但不是实际的运行用户请求的实例。在新应用实例验证通过之后，公共 URL 被映射到应用的新实例上。

蓝绿部署包括以下步骤：

- 如果蓝色应用已经存在，要在重新启动之前手动删除蓝色应用。
- 推送一个新版本的蓝色应用。
- 为蓝色应用设置环境变量。
- 为蓝色应用创建和绑定服务。
- 启动蓝色应用。
- 测试蓝色应用。
- 通过将其绑定到公共地址，将流量映射到新版本的蓝色应用。
- 删除用于测试的蓝色应用临时路由设置。
- 将绿色应用重命名为“绿色应用备份”。备份应用程序继续运行，以便使活动会话继续进行。
- 将蓝色应用重命名为绿色应用。

使用内置在DevOps Service运行库中的Cloud Foundry命令行界面，团队完成了蓝绿部署步骤。

—— Joe Loewengruber，2016

5.5.2　环境抽象层

构建正确的 DevOps 平台，特别是反脆弱性的平台的首要目标就是，将环境和基础设施从应用与服务的开发及测试人员那里抽离。对他们而言，环境应该是可以用来进行开发与交付的一组抽象的基础设施或平台服务。这使其可以通过多条交付流水线，实现多种应用的多级交付；可以使用 A/B 测试技术完成商业创意和新特性的快速实验；可以快速失败、频繁失败；并不需要担心实验失败或交付的应用与性能表现不佳会影响环境的稳定性。

软件定义的环境

在基础设施上引入抽象层的方法就是利用软件定义的环境(Software-Defined Environment，SDE)。软件定义的环境使运维团队(可以是内部运维团队，也可以是云供应商)能够交付 IaaS，将基础设施作为一组服务提供给应用交付周期中的所有操作人员，操作人员可以通过定义明确的 API 访问并使用这些基础设施。通过虚拟机管理程序上运行的虚拟层或功能齐全的云计算平台就可以交付软件定义环境。

通过将环境的各种组件都作为软件来定义及管理，软件定义环境可以提供多个不同的抽象层。其中包括：

- 软件定义存储(SDS)
- 软件定义网络(SDN)
- 软件定义计算(SDC)
- 软件定义管理(SDM)
- 编排与工作负载自动化

VMware 与 IBM 这样的公司现在能够为整个数据中心提供软件解决方案，而不仅仅是环境的管理，他们将这种解决方案称为软件定义的

数据中心(SDDC)。

软件定义的环境，通过 API 向环境客户提供抽象层及环境访问权限，同时也通过环境管理软件为环境供应商提供自动化能力。环境供应商可以执行以下操作：

- 通过版本化定义环境的软件来实现环境的版本化管理。
- 创建软件版本库，以便在需要重建环境以评估缺陷时能很容易地访问到之前版本。
- 环境变更管理。运维团队通过创建新的环境版本，将每次变更，无论是打补丁还是变更配置，都变成软件驱动，这就使得环境的变更管理变得更加容易。这些变更都是通过脚本来实现的(不需要登录到不可扩展且难于管理的管理控制台)。这些脚本利用代码及交付的其他应用构建物实现版本化。
- 执行传统的软件工程实践，比如对环境代码的源代码管理。
- 执行软件定义环境的库存管理实践。在环境仓库中，默认使用最新版本的环境构建代码表示软件定义的环境，这样可以使得库存管理变得更加容易。
- 执行环境的配置管理。因为所有的配置管理都是通过版本化的代码来完成，配置管理也变得更加容易。

云托管环境也是由软件定义的。然而，软件定义与管理的级别可能会因云计算供应商及其所使用的云管理软件堆栈不同而有所差别。例如，云中的裸机服务器可能没有完整功能的软件定义网络，因为它们利用物理硬件(裸机)来提供服务器实例。同样，软件定义的存储将决定供应商是否提供对象存储。

5.5.3 云托管的 DevOps 平台

无论是作为基础设施即服务(IaaS)、平台即服务(PaaS)，还是容器服务，云托管环境所交付的抽象层都能实现反脆弱性 DevOps 平台的交付。在本章的后面部分将比较 IaaS 和 PaaS 模式的异同，并将讨论如何在这两种模式上交付 DevOps 平台。

云消费模式

在开始深入研究如何在云平台上构建 DevOps 平台之前首先要确保我们对云计算技术的理解是一致的。云计算已经成为又一个被过度使用及滥用的术语。大多数人对云计算的印象仍然是公有的、多租户的云，提供基础设施即服务，使组织不需要拥有硬件，即可利用供应商的数据中心获得基础设施服务。首先，云不再仅仅是虚拟化的基础设施。事实上，云计算的真正价值在于组织有能力使用基础设施上的云服务；更快速、更大规模地交付创新业务服务不仅仅降低了成本，而且还可以提供在云服务出现前所无法提供的服务。云消费模式已经超越了传统的非私有云的基础设施模式。我们先来研究一下这些消费模式。

娱乐体育节目电视网(ESPN)与体育消费

今天的孩子们不会了解这种感觉，那就是强迫自己保持清醒，只是为了看到 John Tudor 能完胜太空人或 Lee Smith 可以保持并打破原来由 Chris Carpenter 所保持的纪录。他们永远不会冒险在脚上贴上吸引阿肯色民主公报的标签，只是为了回顾红雀队对战利物浦队的决胜局比分。他们永远体会不到这种期待的感受，这很大程度上要归功于 ESPN 娱乐体育节目电视网。

我不会带领读者去了解 ESPN 的诞生及历史，而是会重点关注在针对 7×24 体育节目的看法、消费方式以及投入方面它带给我们的影响。而今，可能有上百个 ESPN 平台活跃在各大媒体上。首先，他们有多个电视频道，例如 ESPN、ESPN2、ESPN 新闻、ESPN 经典。然后他们还有自己的网站，ESPN.com、espn360.com 和 espnradio.com，当然还有广受欢迎的广播电台。每一个平台都为我们提供了关于分数、交易、传闻、观点、洞察力以及分析的不间断报道。除此之外，他们还有多个应用程序，比如评分中心、体育中心和 ESPN 的无线电台应用程序，不胜枚举。随着推特的出现，你甚至可以追随你最喜欢的 ESPN 节目，这样你就可以随时享受运动。现实是我们大多数人都不想没有体育赛事。这是一个

> 简单的供求问题。基于我们的需要，ESPN 和其他体育网络为我们提供了丰富的体育赛事自助餐选择。我不是说这是个坏事，作为一个体育迷，我很是享受，只是惊诧于运动与科技结合所带来的惊人变化，在安装有线电视前，我要看那时而彩色时而黑白的电视，还得不停调整天线，那个年代，只有富人能买得起卫星电视。
>
> —— Lopez，2014

无论云计算是多么全新的技术，你都不能说，“这不是你父亲那个时代的云，”或者像本文的作者那样谈论说“今天的孩子们将永远不会经历”那种在昔日的云上配置服务器实例的感觉。云技术变化更新的速度如此之快，以至于很多组织都无法跟上其变化的脚步。云技术不断发展也导致了云消费模式的重大改变。过去那种只需要简单地选择“私有云”或“公共云”的消费的模式已经一去不复返，而今已有很多的消费模式可供选择，不仅要考虑成本或位置，还需要考虑很多其他的因素。选择适当的模式，在大多数情况下都会选择多种模式联合创建的“混合云”，这将决定 DevOps 平台上应该保留并支持哪些组件。

私有云与公有云 这可能是组织云消费模式的首次重大变化。在云计算刚刚出现时，供应商的数据中心中只有公有云，由供应商管理，而且是多租户的。Salesforce.com(1999 年发布)和亚马逊网络服务(2002 年发布)这样的早期云供应商也就只能提供这一种模式。即使到了今天，人们提到“云计算”这个词时，通常还是指公共云。随着云技术的不断发展，出现了云管理软件，从而使组织能够在其自己的数据中心建立私有云。私有云是自我管理的云，位于组织自己数据中心的防火墙内。私有云在组织自己的硬件上运行，属于组织自有(on-premise，简写为 on-prem)云，当然肯定是单租户的。因此，私有云总被当作组织自有的云看待。

私有和公有成为两种可能的选择。具体做出哪种选择，取决于诸多方面的因素：成本、数据存储位置、单租户还是多租户、支持以及自行管理内部云的能力等。大多数组织都会为关键任务型的应用选择私有云，因为这些应用高度规范，对数据存储及隐私保护有严格的要求，因

此需要将数据存储在自有的数据中心。其他的应用则部署在公有云，成本更低，并且不需要在自己的数据中心安装新的硬件。多数大型组织最终都会采用混合云模式，既有公有云和私有云托管，也有传统的、自有的硬件上的托管应用。

消费模式发展到今天，已经完全突破了公有云共有，私有云自有的观念，使之划分为非专属数据中心和对应自有的专属数据中心的概念。这种演变催生出两种新的消费方式：专有云与本地云。因此，目前有三种模式可供选择：公有云、专有云以及本地云(如图 5-5 所示)。

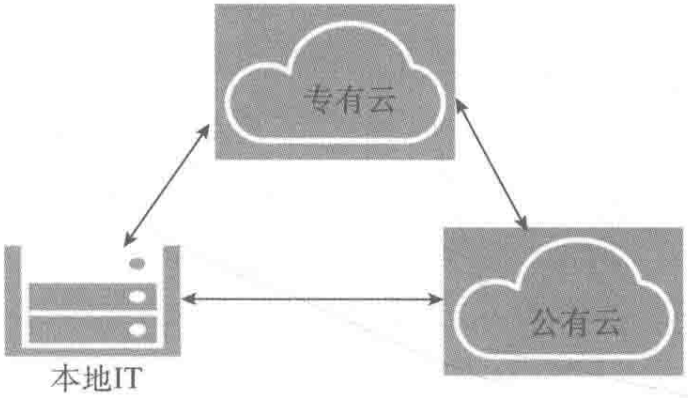


图 5-5　云消费模式

专有云　云技术已发展到全新的水平，云供应商已经能够提供在其数据中心托管的单租户云。IBM 率先交付了专有云模式。所以，现在组织有了“专有”的云，但仍然由供应商管理，留在供应商的数据中心(非自有)，在供应商的硬件上运行，但却是单租户的。所以，这是私有云，但属于非自有性质，且由供应商进行管理。因此，可以这样认为：专有云是私有的、托管的，非自有的云服务。

本地云　组织云消费模式接下来的发展更具颠覆性。组织，特别是那些应用高度规范，对隐私保护有严格要求的组织，通常都想拥有昂贵、便利的自有云。拥有自有云，就不再有客户数据存储的问题；就不再有在其他数据中心运行关键应用的管理合规性问题；也不存在使用“专线”进入另一个网络才能访问应用与基础设施的问题。然而，自有云也给组织带来了相应的挑战，而且一直都是关于自行管理云所需的技能以及可能产生的风险。我指的是那些擅长管理硬件基础设施

的 IT 组织，现在来管理云以及不断发展的技术栈，而这些都是需要管理与维护的。这里说的是基于组织自己的服务等级协议，利用所有相关的云服务高效运行并管理云，这就要求用自有的云来交付公有云的质量承诺。

通过本地云，我们找到了这样一种两全其美的消费模式。这是位于客户自己数据中心(自有的)的云；但却由负责交付的供应商管理；当然，它也是单租户的。在本地云模式下，组织只需要继续管理云交付的硬件基础设施即可。尽管还是由供应商负责管理。这要通过“合作”来实现，要允许供应商进行随机的云部署，持续监控并根据需要升级云软件堆栈。(IBM 仍然是这个模式的先驱，他们将这种“合作”称为“中继”)。客户组织负责协调升级的时间安排，这使其得以控制停机窗口。因此，这是由供应商管理的私有云，但却是自有的。可以这样认为：本地云是私有的、托管的、自有云服务。

选择适当的消费模式并非那么简单。公有云消费被认为是非粘性(相对而言)的模式，容易启动也容易关闭。专有云也是一样，因为其本质上就是分配给组织专门使用的公有云。这类似于租用独幢建筑，而不是多租户共同使用的公寓。本地云则有所不同。理解组织与供应商之间的职责分工至关重要，了解变更管理流程也是十分必要的。与供应商建立起合作伙伴关系更为复杂，却也比使用公有云更有价值。

自行管理与供应商管理 为 DevOps 平台选择最佳模式时，第一个决策点至关重要：你是想找由供应商管理的云环境，还是想自行管理？如果你倾向于自行管理，那么就选择传统的、自有私有云。组织从供应商那里购买云平台技术，培训人员，准备数据中心中的基础设施，并利用已经交付的云构建 DevOps 平台。

如果你更倾向于供应商管理的模式，那么组织就要在自有和共有云平台之间做出选择。首先要决定的是托管还是自行管理云。这不仅影响对云模式的选择，还会影响到 IT 人员的配置。如果基于技能或者员工人数的限制，依靠自己的 IT 人员，组织没有能力来管理云，或是不确定凭借自身的能力和意愿，能否交付用户期望通过云来实现的服务，并满足服务等级协议要求，那么就应该考虑托管云。到目前为止，这仍然

是一个不得已的选择，因为所有托管云模式还是只能在非自有云上运行。随着本地云服务 IaaS(基础设施即服务)和 PaaS(平台即服务)的出现，情况已经发生改变。组织可以获得作为托管服务的自有云，并在此基础上构建 DevOps 平台。

云消费模式一旦确定，接下来就要确定适合的云应用模式了，是选择基础设施即服务、平台即服务还是容器服务。决定好以后，就要开始选择构成 DevOps 平台的服务。组织想要这些服务以何种方式交付？是利用 IaaS 云端运行的 DevOps 工具构建的服务，还是作为实际平台的组成部分而交付的更好？我们来深入研究一下这些选择。

5.5.4　基础设施即服务

> **基础设施即服务**
>
> 向用户交付的能力包括计算、存储、网络以及其他的基础计算资源，用户能够利用这些能力部署并运行任意软件，其中可以包括操作系统和应用程序。用户不会管理或控制底层云基础设施，但可以控制操作系统、存储、部署应用，还可能对网络组件的选择具有一定的控制权(例如，托管的防火墙)。
>
> —— NIST，美国商务部，2011

图 5-6 完美地诠释了美国国家标准与技术研究所(National Institute of Standards and Technology，NIST)所描述的定义。展示了 IaaS 与 PaaS 模式下云供应商管理的组件的对比情况。这里的云提供者可能是云供应商，也可能是组织内部运维团队，这取决于云是由供应商管理的还是组织自行管理的。

图 5-6 直观地描述了 IaaS 和 PaaS 这两种云应用模式的区别，即，有多少技术栈是由用户(或客户)管理的，又有多少是由云平台供应商管理的，而云平台供应商既可以是外部供应商也可以是组织内部的运维团队。在本章后面讨论 PaaS 时，将探讨如何利用 PaaS 平台来交付 DevOps。

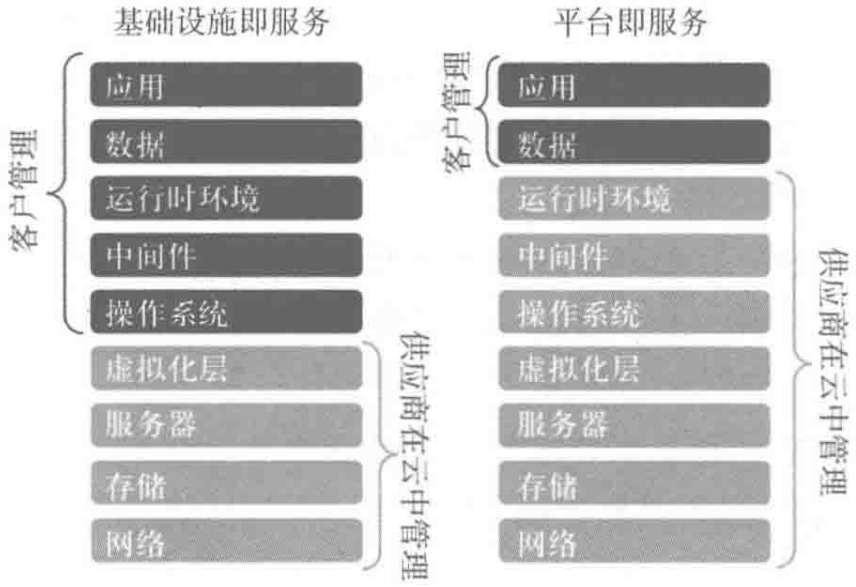


图 5-6　IaaS 与 PaaS

在这些环境中使用 IaaS 云平台，是为了增加软件与基础设施交付人员之间的抽象层，他们看到的只是基础设施的简化视图，然而在表象之后，基础设施可能会遭遇所有的典型挑战，要解决这些问题就需要进行任何硬件设施都需要的同等维护。

要特别注意的是，组织采用混合的方式来应用云平台的情况并不少见。虽然由于成本较低，并且没有数据存储(数据存储位置)方面的需求，组织可以利用公共或专有云作为开发、测试环境，但是，开发、测试的过程并不涉及实际的生产数据，所以组织可能会选择保留自有生产环境。这使得生产中运行的应用更接近数据源，并可以更好地满足与数据存储位置有关的合规性要求。随着专有云这类模式的进一步发展，数据安全方面的顾虑已不再是非本地云实施的障碍，甚至可以将其用于生产环境了。

IaaS 能力栈

创建并管理一个 IaaS 产品，需要首先创建必要的 IaaS 云架构。这就需要具备以下能力，如图 5-7 所示：

- 代理和目录
- 服务编排与集成
- 云编排
- 部署编排

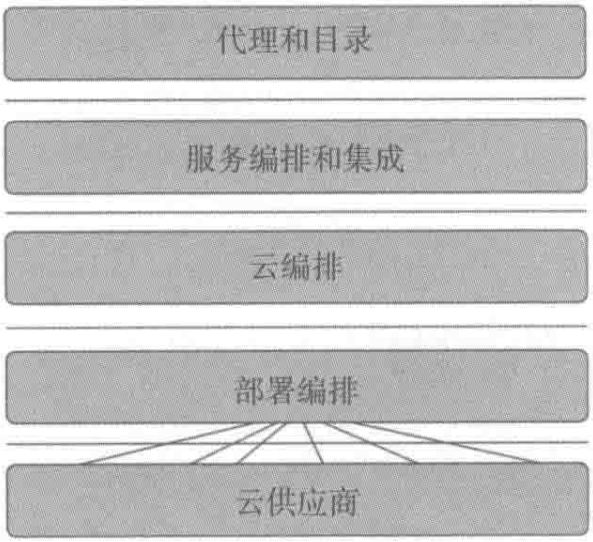


图 5-7　IaaS 能力堆栈

代理和目录　云供应商数量激增，组织对混合云的接受程度越来越高，利用多家供应商以多种云应用模式来提供服务，催生了云堆栈中的一个全新功能领域：云代理。一个橄榄球教练可能同时拥有多个四分卫[②]、跑卫以及接球手，他可以从中选择与所要实施的战术最为匹配的球员，比如说，“第三次进攻”就要配备健壮的四分卫、跑卫以及后卫。同样，IBM cloudMatrix 和 RightScale Cloud Comparison 这样的云代理工具也能帮助组织从可用选项的目录中选择适当的云来部署应用。根据应用对计算、存储、网络以及内存的需求，每种云计算模式的功能与价格，是内部管理还是通过外部的供应商来管理，决策可以实时进行。

服务编排和集成　服务编排或服务集成是整合多个云服务以及在几个云中运行应用程序所需的服务的一组功能。因此，它是云服务的集成层。这一层的核心组件是业务流程管理工具，用于管理正在集成和交付的服务的业务流程，以满足业务需求。服务编排和集成的工具属于业务流程管理工具下面的一个广泛类别，在管理各种服务和互动的过程中，利用工作流来获取编排和业务规则。例如业务流程执行语言(Business Process Execution Language，BPEL)和业务流程建模符号(Business Process Modeling Notation，BPMN)是用来捕捉这些工作流编排

② 多个顶尖后卫在大学橄榄球赛中比在国家橄榄球联盟中更为常见。

和业务规则的标准。

API 和服务管理工具比如监控、安全、度量和计费等，此类的 API 和服务也属于这个领域。

云编排

> **精心策划的大赛前训练**
>
> 运动迷已经察觉到教练们正在为重大比赛进行准备。其中包括情感与精力。这是肯定的，但是情感和精力，就如蜂巢中的两只蜜蜂，只是其中的一部分因素。真实的场景下会涉及更多因素。教练员如何为艰巨的史诗般的对抗赛做准备，使其看起来势均力敌，这是非常重要的。就像乐队指挥要决定如何表演柏辽兹(Berlioz: 法国作曲家)奇幻的幻想交响曲一样，教练也必须要为备战密歇根和诺特丹这样的球队而制定一个全面的基调。请注意：准备工作早在星期一之前就开始了，确切地说这只是猜测……。
>
> 在比赛准备阶段，除了在球场上偶尔出现的“男孩子的冲动”引发的冲突外，对任何干扰的态度都是零容忍的。即使这些小插曲也可以被成员化解，并可以进一步鼓舞士气。如果每个人都为任务做准备，就没有必要担心注意力不够集中。当一个球员迷失方向，造成一些不必要的困扰，参与者(教练)就会崩溃，一个典型的表现就是喋喋不休。教练不希望球员有抵触的情绪，这样做已经足够了。
>
> —— Wolverine，2014

云编排涉及在云环境中自动化管理各种组件和服务的交付过程。编排的目标是管理分配与回收、资源配额、配置、与云技术栈相关所有组件的工作流以及云计算交付的所有基础设施服务。一个定义编排的方法是，将 IaaS 云管理的最佳实践和工作流进行编码化。

云编排的能力包括如下(Peranandam，2012)：

- 集成异构环境和云的基础设施功能，以简化、自动化和优化服务部署。
- 通过自动化降低管理员与物理和虚拟机的比例系数。

- 利用策略性工具自动化实现资源的批量配置与回收，通过自动化回收资源来管理虚拟机的无序扩展。
- 完成能力整合的工作流程和跨越技术孤岛审批链，促进协作和减少延误。
- 对物理的和虚拟的云资源进行实时监控，消费和计费扣款功能跟踪以及优化系统使用。
- 封装自动化模板和工作流程中最常见的资源类型来缩短实施最佳实践和减少转换时间。

IBM Cloud Orchestrator 和 VMware vRealize 是在云编排领域市场领导者的典范。

> **云计算模式**
>
> 基于特定的一组虚拟机镜像、中间件、应用程序和运行时配置形成一套可重用的解决方案。部署模式的结果是配置、调整和优化的应用程序环境。
>
> —— Chiara Brandle，2014

编排和架构定义可以被云计算模式获取。通用的标准模式为 OpenStack Heat 模式，称为 Heat 编排模板(Heat Orchestration Templates，HOTs)。

对云计算的定义和代码化的模板已经在过去几年显著成熟。IBM 是这个领域的先驱者，其专有的云模式在 2009 年的 Cloudburst Appliance 第一次亮相，在 2011 年进化为 IBM Workload Deployer(IWD)。这些模式被称为虚拟系统模式(Virtual System Patterns，vSys)，已经进化并且成为目前 IBM PureApplication 系统(PureApplication Systems，PureAS)主要支持的产品。vSys 模式在 IBM 正在逐步淘汰，OpenStack Heat 将取而代之。在 IBM 之外，其他模式也在进化。其中，亚马逊 Web 服务(Amazon Web Services，AWS)CloudFormation，发布于 2011 年，鉴于亚马逊扩张的足迹，CloudFormation 是最受欢迎的一种云计算模式。

从标准的角度来看，OpenStack HOT 模板的社区标准受到 AWS CloudFormation 的启发，该模板支持 OpenStack Heat 引擎。今天大多数

云供应商直接支持 OpenStack 的 Heat，或者提供 OpenStack API 的支持，允许在这些非 OpenStack 云上提供 Heat 环境。

下面是一些云组件的例子，它们被定义并包含在模式中：

- 预安装的操作系统
- 跨组件预集成
- 预配置和优化中间件
- 预配置的监控
- 预配置的安全
- 生命周期管理

因此，模式至少由以下部分组成：

- 操作系统基础镜像
- 应用程序的二进制文件
- 自动化脚本(例如，Chef、Puppet、SaltStack 或 Ansible)
- 编排模板(例如，OpenStack Heat 或亚马逊 CloudFormation)

正如你所了解的那样，模式本身在未来将受到不断演进的容器的挑战。尽管容器并非传统意义上的基础设施即服务，其轻量化和可移植性却使其可以作为传统 IaaS 的一种替代方案，甚至可以作为 PaaS 的替代方案。虽然一些容器技术正在兴起，但容器领域中的领袖仍然是 Docker。我将在本章后面更详细地讨论容器。

部署编排 在第 4 章中详细讨论了部署编排。值得注意的是所提供的云环境是否独立于在此环境之上部署的应用程序，或者是否把正在部署的应用和环境作为一个全栈部署过程。每个案例中的模式将使用不同的方式定义，例如作为环境模式和应用程序模式，或者作为一个完整的堆栈模式。就像第 4 章中所描述的那样，IBM UrbanCode 部署工具使用 OpenStack Heat 模式有能力设计和提供完整的环境堆栈，此堆栈可以定义为应用设计图。

5.5.5 OpenStack Heat 作为抽象层

利用 OpenStack Heat 的好处是，它可以作为一个抽象层，允许环境具有云兼容性，使用 Heat 模板记录一个独立的环境，具有在多个云中

实例化的能力。事实上，由多个模板组成的复杂的环境可以从几个供应商的多个云中提供，如图 5-8 所示。这也让云具有可移植性，也是所谓的圣杯云(Holy Grail)。今天在这个云里提供的一个环境，如果有更好的选择，明天可以由另一个云提供该环境(参见“代理和目录”)。

IBM UrbanCode 部署工具是支持这种模式的工具之一。使用 UrbanCode 部署工具的设计功能，使得 UrbanCode 部署工具在内的 Heat 引擎能够创建 Heat 文档(HOT)，用来在多个云中提供环境(Frederick，2016)。

我的朋友 Sudhakar Frederick 在博客中提到，多个云的移植性还没有完全实现，甚至是在使用 OpenStack 技术的云上也很难实现。在一些云上的某些功能在其他云中没有直接对应的类似功能，所以对于所使用的特定云服务来说，特殊性越强，HOT 文件的可移植性就越弱。

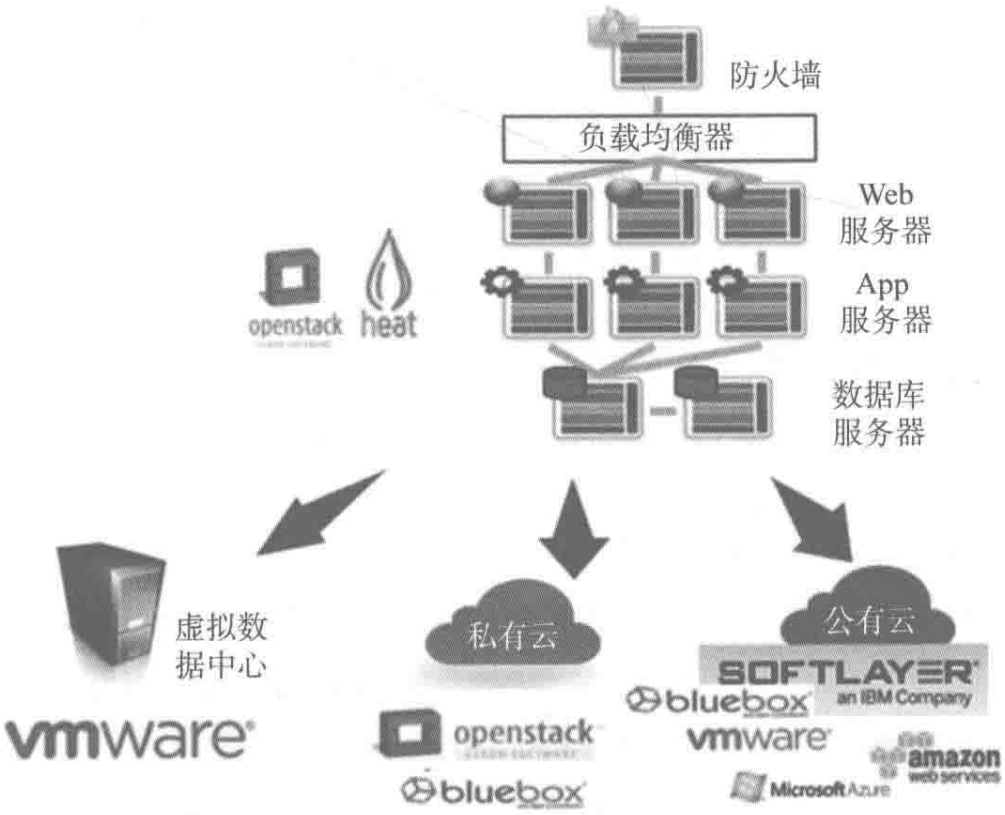


图 5-8　OpenStack Heat 支持的多云场景

5.5.6　平台即服务

平台即服务

向用户提供将应用程序部署到云上的基础设施的能力，这些应用程序

是由客户创建或通过供应商支持的编程语言或工具创建的。消费者不需要管理或控制底层的云基础设施，包括网络、服务器、操作系统或存储，但对已部署的应用程序和可能的应用程序宿主环境的配置具有控制能力。

——NIST，美国商务部，2011

同大多数的 IT 术语(或者更概括地说，在语言这种人类沟通媒介中)一样，PaaS 也是被滥用、过度使用并误读的。快速的网络搜索或者访问维基百科上的 PaaS 页面就能证明这一点。

定义 PaaS

NIST 对 PaaS 的定义中最为关键的部分是，PaaS 服务与“基础设施即服务”的消费模式的差异，具体内容是：

……消费者不管理或控制底层的云基础设施，包括网络、服务器、操作系统或存储，但对已部署的应用程序和可能的应用程序宿主环境配置具有控制权。

图 5-6 很好地说明了 IaaS 和 PaaS 之间的差异。在图中可以看出两种云计算模式 IaaS 和 PaaS 的差异，以及有多少技术栈由用户(或客户)管理，以及多少是由云平台供应商(可能是一个内部团队或外部云供应商) 托管的。强调以下两件事是很重要的：

- 技术栈中的每个功能都可作为托管服务；利用从底层实现抽象出来的，可以作为提供给客户的一项共享的、多租户的服务。
- 客户/用户只关心他们自己管理的应用程序、数据和用户访问，利用平台提供的服务管理其他的部分。

平台即服务可以是公共的(IBM Bluemix、Pivotal、Salesforce 的 Heroku、谷歌应用程序引擎等)，也可以是私有的(自运维的 Cloud Foundry；自建和自运维的平台)。利用多种技术和自运维的私有云，组织也可以构建自己的运维和管理环境。对于在其自有的数据中心中托管的一个私有平台实例，组织将负责管理在平台上托管的所有服务。通过 Pivotal 的

Cloud Foundry 建立一个自我管理的 PaaS 会是一个很好的例子。

基于 PaaS 的 DevOps 服务

当你考虑基于 PaaS 实现 DevOps 时，你需要了解托管在平台上的一些服务，以便在平台上实现一个 DevOps 应用程序流水线。如果研究 DevOps 应用程序流水线和流水线的核心组件，每个组件需要作为服务以便组成完整的基于 PaaS 平台的 DevOps 解决方案。

下面是组成流水线的能力列表，都需要以 PaaS 服务方式提供：

- 源代码管理
- 构建
- 持续集成
- 部署自动化
- 中间件配置
- 自动化测试

这些都是与 DevOps 有关的服务，如图 5-9 所示。如果留意的话，你会发现，这是 IaaS 章节所给出列表的一个子集。这是因为 PaaS 包含了环境提供和配置管理有关的能力，而这也是 PaaS 的核心能力。PaaS 的终极目标是将环境维护工作从平台消费者那里抽离出来。

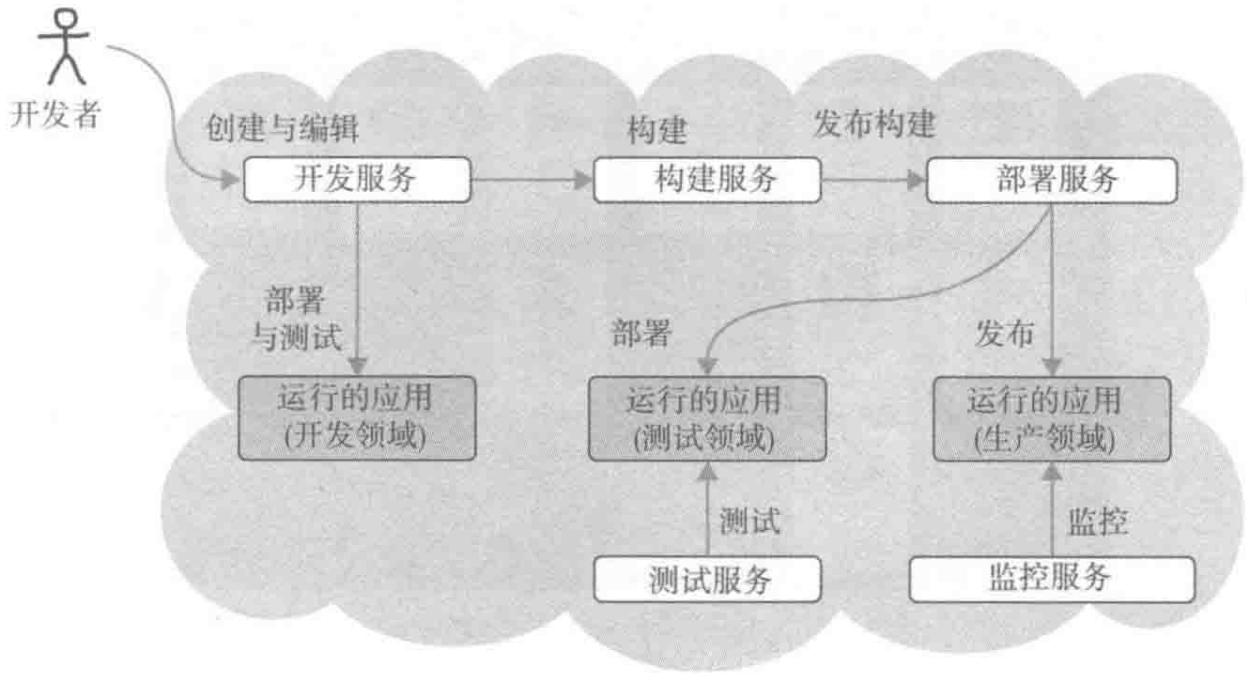


图 5-9　DevOps 服务

像 IBM Bluemix 这类全功能的、托管的 PaaS 产品已经包含了平台原生的 DevOps 服务。下面是 Bluemix 的一些 DevOps 服务：

- Git 服务，和 GitHub 企业版服务
- 基于 Web 的 IDE(Integrated Development Environment，集成开发环境)
- 敏捷计划和跟踪、团队协作服务
- 流水线服务
- 全球化服务
- 部署自动化服务
- 自动扩容服务
- 性能监控服务
- 告警通知服务

这些能力相互协同工作，在 IBM Bluemix PaaS 上提供了持续交付流水线，如图 5-10 所示。

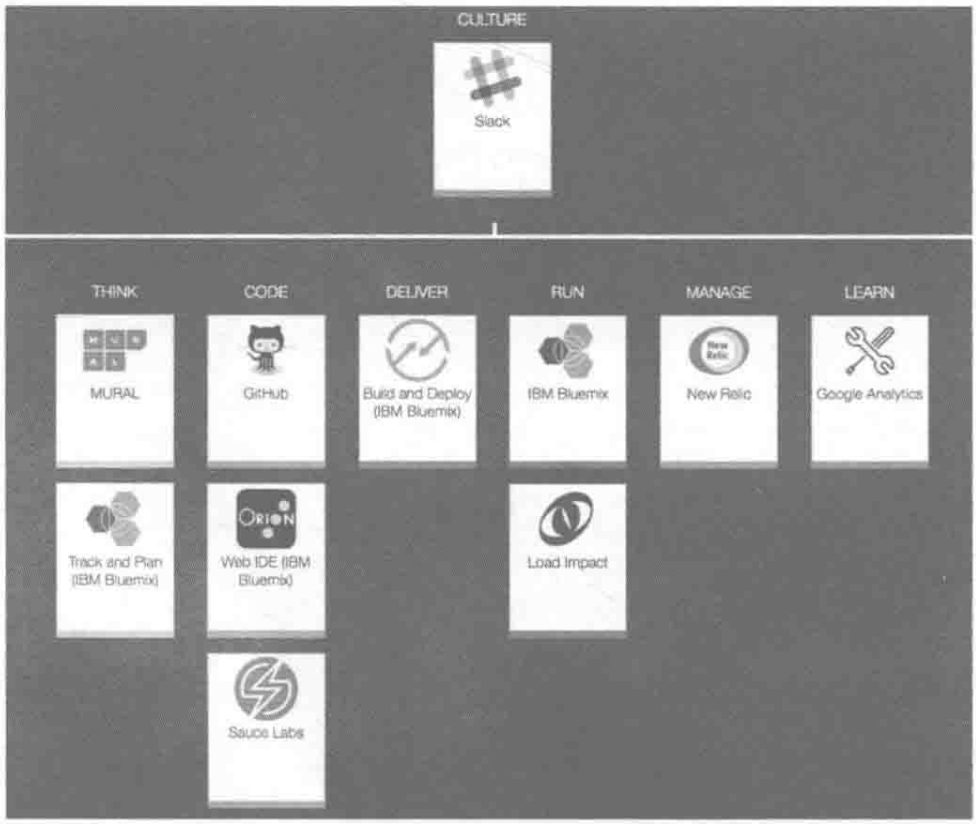


图 5-10　Bluemix DevOps 流水线(IBM)

DevOps 服务

采用 PaaS 平台的价值是不言而喻的。对于任何想要实施 DevOps 的企业而言，使用 PaaS 提供的 DevOps 服务是一个成本非常低的启动方式。不需要手工制造交付流水线并实现整个持续交付工具链。集成、托管、运维这些问题将不再困扰你。按使用量付费，允许弹性伸缩。

PaaS 与 Cloud Foundry

说起 PaaS 的话题就不能不涉及 Cloud Foundry，毫无疑问它已经成为交付 PaaS 标准。Cloud Foundry 是一个开源的 PaaS，现在由 Cloud Foundry 基金管理。基金的董事会主要来自云计算用户和供应商，如 IBM、Pivotal、HP、EMC、SAP 和 VMware，他们支持以及提供基于 Cloud Foundry 的 PaaS 平台。IBM Bluemix 是共有的 PaaS 平台，基于 Cloud Foundry 的 PaaS 实例是本书写作的一个较大部分的内容。

云计算的真正价值不在于开源本身，而是为了让开发人员和 PaaS 供应商有更多样化的选择：

- **开发框架和语言。**Cloud Foundry 在本质上是真正支持多语言的，无论是内置的或通过社区添加的构建包，支持所有主要的语言。
- **应用服务。**除了核心的应用服务之外(如 MySQL、MongoDB、PostgreSQL、Redis 和 RabbitMQ)，供应商如 IBM 还增加了几个服务，其中包括 WebSphere 应用服务器、DB2、区块链、MQ Light 等。
- **多云兼容。**组织可以选择在任何云上部署 Cloud Foundry。像 Pivotal 这样的供应商提供了在多个云上部署的能力，包括 AWS 和微软 Azure。IBM 只提供托管的 PaaS，在 SoftLayer 上部署 Bluemix、VMware 和 OpenStack。

5.7.7　容器

迪昂·桑德斯：全垒打和达阵

绰号“Prime Time”和“Neon Deion”的桑德斯(Sanders)打了九年

的棒球，而且在资格达到的时候很有可能成为名人堂橄榄球球员[③]。他是一个伟大的橄榄球运动员，曾八次入选职业碗，并两次赢得超级碗，同时，他还是一位优秀的棒球运动员，如果专职的话，很可能会表现得更好。

桑德斯是唯一一位既参加过超级碗又角逐过世界职业棒球大赛的运动员，也是同一周内打出全垒打和达阵的独一人。

—— Timmons，2008

像桑德斯这类的运动员是独一无二的。他们在任何体育竞技场都表现得出类拔萃。无论什么类型的比赛，他们都能自如发挥。教给他们规则和基本打法，并进行战术指导，他们就能很好地完成比赛。

这些独特的运动员，他们的运动能力似乎是“可移植”的，同这些运动员一样，容器也具有可移植性，可以在不同环境运行，并可跨环境移植。容器的概念本身并不是最近才出现的。早在 2008 年 Linux 上就出现了容器。Linux 容器允许在 CPU、内存、块 I/O 和网络资源级别上进行隔离，同时共享操作系统(OS)内核。每个实例都有一个完整的本地操作系统，这允许进程独立运行，同时消除了虚拟机(VM)的瓶颈。这使得容器比 VM 轻得多，从而得以在服务器更大的规模运行。Docker 开始作为一个开源项目使 Linux 容器更具移植性，它们可以在任何 Linux 实例上、任何物理服务器上，或在云上进行迁移。自从 2013 年的发行版以来，Docker 已经成为一个最成功的开源项目，Docker 引擎的下载量达到了一亿(Martin N，2015)。

简而言之，容器的主要优点是，提供了打包应用程序、配置及其所有依赖项的标准方法，使其在环境中具有可移植性。容器中的特殊应用程序代码与容器外的环境隔离。这几乎是 DevOps 的涅槃重生。开发人员不需要关心容器外面的环境。他们专注于将应用程序打包到标准化的容器镜像中，这对应用最为合适。同样，运维团队不需要担心容器中代码变更后的应用程序、配置和依赖关系。他们只需要专注于在环境中运行标准化的、经过认可的容器。

---

③ 引文撰写完成 3 年后的 2011 年，迪昂·桑德斯成功入选了橄榄球名人堂。

基于容器的 DevOps 平台

如前所述，组织把构成应用交付流水线的工具部署到容器中，使其成为可用的服务，从而通过容器方式提供交付流水线的能力。或者，当进入到应用与服务的开发阶段时，应用交付流水线的工具可以按照传统的方式部署到容器以外的任何环境中，仅在需要部署到测试、预生产以及生产环境中时才打包到容器中。在这两种情况下，容器的最大优势在于能够提供一组标准化的产品开发、测试、生产以及其他的环境，使应用部署在可移植的容器中，并且当从一个环境升级到另一个环境时，不会产生任何兼容性问题或配置管理任务。

当涉及应用程序交付流水线时，需要做的唯一改变是构建过程，它现在包含将应用程序打包到容器中的步骤。像 Jenkins 这样的现代的构建工具，和 IBM UrbanCode 部署工具，支持以下功能：

- 通过 Dockerfile 构建 Docker 镜像
- 将 Docker 镜像发布到注册表

使用 Docker 工具，或部署自动化工具的能力，在完成构建后，Docker 容器可以部署到所需的环境。如果需要更高管理水平的工具，那么可以使用 IBM UrbanCode Deploy 部署自动化工具。

需要注意的是，在市场上有几种容器技术，就像 CoreOS Rocket 和 VMware Photon 容器。然而 Docker 才是市场的领头羊，相比同类产品具有明显的市场占有率优势。2016 年 4 月，开放容器倡议(Open Container Initiative)由几十家厂商推出，规范容器格式和运行时环境。

容器编排

容器的透明的可伸缩特性使得在多个容器中部署多个实例得以实现。在其生命周期中，管理这些容器，涉及多种类型的任务和能力，其中包括：

- 部署
- 更新容器
- 提供实例
- 容器的发现

- 监控
- 调度
- 集群(Clustering)和弹性伸缩
- 故障转移
- 政策管理
- 约束管理

基于提供这些能力所需面临的挑战，一些管理容器编排的技术应运而生：

- **Docker Swarm**。利用 Docker Swarm，Docker 开始为大规模的 Docker 容器集群提供支持，通过核心 API 管理 Docker 引擎资源池中的多个容器，而不是进行单独管理。
- **Kubernetes**。谷歌推动了 Kubernetes 编排技术在容器领域中的发展。谷歌声称，它每天使用 Kubernetes 管理着超过二十亿的容器，Kubernetes 非常可靠。Kubernetes 架构是主控服务(master)管理多个代理程序。主控服务通过管理和编排流程来调度所有的代理程序。这些代理程序依次调用名为 Pod 的多个容器集群。
- Mesos：Mesos 是一个独立于 Docker 存在的开源项目，它可以管理服务器上的共享资源池并运行复杂的任务。它已经增加了对 Docker 的支持，而且支持容器管理。与 Kubernetes 类似，都具有主从模式。主控服务运行高级任务并将任务委托给管理容器的代理程序。ZooKeeper 软件提供更高级别的管控、伸缩性和高可用性，它可以协调主节点集合。ZooKeeper 通知所有的主控服务和代理程序，哪个是主导地位的主控服务，用以实现协同目标。

容器即服务

作为一种替代 PaaS 的方案，容器即服务(Container as a Service，CaaS)产品已成为主流。在这里，PaaS 环境作为一种服务被提供，并且由内部运维团队或供应商管理。开发人员只需要在 PaaS 服务上运行它们构建的容器，而不必为设置环境、管理或配置的任务所困扰。不同于 PaaS，

CaaS 本身不提供任何应用程序或中间件服务。应用程序如果需要任何 CaaS 之外的服务，就需要在 CaaS 上部署一个运行该服务的容器或者直接利用所需的外部运行的服务。

流行的 CaaS 服务包括 Docker Cloud(以前是 Tutum)、IBM Bluemix 容器、亚马逊 ECS、CoreOS Tectonic、Rancher Labs 的 Rancher 以及谷歌容器引擎。

5.6　方案：交付微服务架构

主力团队与特勤组

作为特勤组的核心成员，前 NFL 中卫和线卫 Coy Wire 在联盟参加了 9 个赛季的比赛。他承认在 NFL 并不是所有人都想做踢球手，但谈到适应新角色对于一个专业球员来说非常重要。

“NFL 特勤组中的大多数球员，在其整个职业生涯中从未踢出达阵。所以，你必须像变色龙一样适应新的环境，” Wire 说，“每个特勤组的球员都希望成为 NFL 的全职球员。”但是为了真正发挥他们在特勤组中的优势，必须把自我放在一边，接受你为你的团队所扮演的新角色。

—— Bowen, 2015

在第 4 章讨论过，从 DevOps 的视角看小批量交付的核心主题。小批量的价值主张是不言而喻的：频繁交付较小的变化，减少反馈的周期时间，以及最小化变更的影响。当你更频繁地验证较小变更时，测试和安全验证就变得更容易了。当部署较小的更改集时，部署变得更加容易。同样，变更管理也变得不那么复杂了。DevOps 涅槃重生了。

然而，从架构的角度来看，交付小批量并不总是可行的。大多数企业应用程序都是单块架构的。它们有几个大的组件，每个组件都作为一个可部署资产交付：

- 用户界面(UI)，一个典型的 UI 组件通常包括 Web 页面、移动应用程序、供第三方应用调用的 API 等。

- 数据库，通常分布在多个数据库服务器上运行。
- 服务器端组件，这往往是多个可执行的组件，通常是为数不多的几个大型组件。

所以，从部署的角度来看，至少有三个较大的组件需要部署。所有组件由多个团队开发，使用 sprint 迭代方式，每次迭代中交付少量的代码更改。然而，所有这些变化都需要集成到较大的可部署组件中。这就与小批量交付的目标背道而驰了。虽然每个交付周期都向可部署组件提供了少量的、增量的更改，但组件本身必须作为单块组件，使用全量部署方式来交付。

从部署编排视角看，下一个挑战是部署组件的多个实例。如果要对任何组件进行弹性伸缩，只能通过在多台服务器上部署多个实例来对其进行水平伸缩。即使只需要扩展其功能的一部分，例如电子商务网站的购物车功能，也需要部署多个单块架构的组件实例。单块组件是可部署资产的最小单位。

5.6.1 微服务架构

微服务架构模式

微服务架构模式是将一个独立的应用作为一组小的服务进行开发的方法，每一个服务运行在自己的进程中并且使用轻量级的通信机制相互沟通，通常是 HTTP 方式的 API。

这些服务建立在业务能力的基础上，并由完全自动化部署机制独立部署。只对这些服务做最低限度的集中管理，这些服务可以用不同的编程语言编写，并使用不同的数据存储技术。

—— Martin Fowler，2014

微服务的方法高效地实现了既提供小批量又具有可扩展性的解决方案(见图 5-11)。马丁·福勒定义将微服务的关键特征归纳如下：

- **通过服务实现组件化。**利用明确定义的 API，微服务可以实现组件化服务的开发与交付，这些组件化服务组合起来可以提供复杂服务。

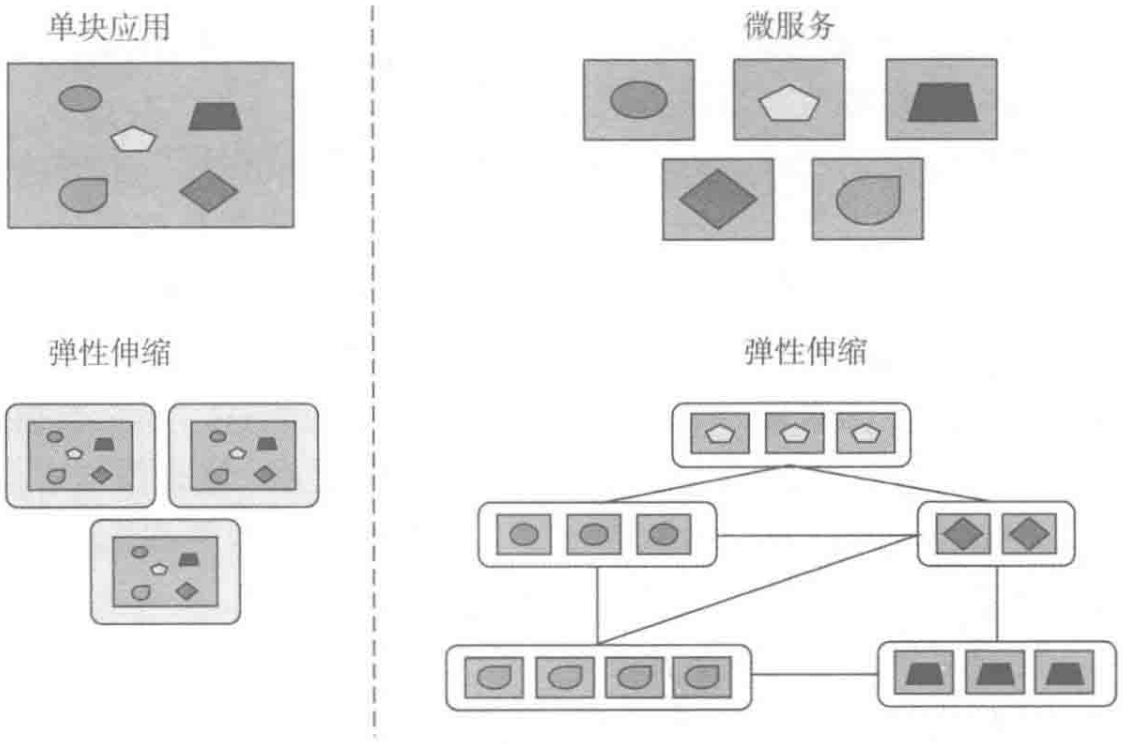


图 5-11　微服务的弹性伸缩

- **围绕业务能力进行组织。**微服务的目标是根据业务功能，把单块架构分解成更小的服务。在第 4 章介绍过，按照康威定律，团队倾向于把业务逻辑映射到他们负责的架构组件上，即使在架构角度并不适合的地方也是如此。每个业务的能力，需要在架构层面上独立为服务，可以进行独立开发和交付。

  以业务能力为中心开发微服务同样可以解决扩展性的挑战。对弹性伸缩的需求通常来自某些业务能力，就像前面提到的购物车，或者说，在月末的时候银行核查余额能力。拥有典型特征的微服务或为独立业务能力而开发的一组微服务，使得那些微服务在需求增加时可以水平伸缩，却不会影响其他微服务的部署。
- **基于产品而不是项目。**产品不是暂时性的，而项目是暂时性的。同样的，团队应该通过微服务的生命周期拥有它们，而不是在一系列项目中由不同的团队管理一套组件，而在组件的生命周期中，项目团队并不拥有对组件的所有权。
- **智能服务和轻量级通信。**微服务是由简单的 REST API 或轻量级消息总线作为架构接口构建的。这允许微服务与其他的微服

务以及外部的服务和应用进行通信，使用轻量级的方式通信，可以很容易地对微服务进行编排和管理。

- **去中心化的治理。**这使得微服务的开发及交付团队能够将最好的平台和工具用于开发和交付微服务。微服务是相互独立的，通过良好定义的 API 进行通信。它们不需要使用标准化的开发语言或技术栈来实现微服务。
- **分布式数据管理。**如果应用层被分为小的组件，那么数据模型的持久化层也需要匹配这个架构模型的要求。这是符合逻辑的，不再需要使用单一数据库支持单块架构方式的数据访问。
- **基础设施自动化。**编排大量微服务持续交付，要求相应的工具支持其弹性与交付，并满足生命周期管理的需求。使用 PaaS 或容器来管理微服务的部署势在必行。
- **针对失败而设计。**微服务通过小巧的功能，独立运行的服务促进了反脆弱的系统设计，从而实现从零开始实现反脆弱性。
- **演进的设计。**与典型单块应用相反，利用微服务架构设计一个应用程序，避免了从一开始就将开发人员锁定在应用架构之中。在快速反馈周期中，通过小批量交付方式，微服务的架构和设计可以进行演进，通过变更、替换、移除来增强架构。

5.6.2 应用的 12 要素

微服务方法，聚焦于现代化应用程序设计与开发的架构模式，同时也催生出大量关于如何开发与交付此类应用的方法论。其中，最受欢迎的是Heroku的员工所开发的“应用的12要素”。

Heroku目前是Salesforce.com旗下PaaS服务的供应商。12要素的方法论是由Heroku发展出来的，其目的是为了帮助组织开发和交付云原生应用，这些应用是专为云计算而设计的。然而，微服务架构和12要素的方法论是高度吻合的，因此12要素成为了开发与交付微服务的最常用方法。

来快速浏览一下12要素应用的概念及最佳实践，其中显然包括了以下十二要素(Koffel，2014)：

(1) 代码库。将所有代码都放在一个源代码管理(SCM)系统中。当然，它可以是一个类似 Git 的联合的以及分布式的 SCM 系统。

(2) 依赖。从应用程序源代码到基础设施代码，确保所有依赖项都声明并彼此隔离。

(3) 配置。将变量存储在环境中而不是存储在应用程序中。即把配置变量从一个环境变更到另外一个环境，或随时间的改变从应用程序中独立出来。

(4) 后端服务。从后端服务中抽象应用程序模型，通过简单节点配置解决切换后端服务的问题。应用程序利用外部的后端服务支持其运行，后端服务类型从数据库到缓存再到消息队列系统。

(5) 构建、发布、运行。使构建-发布-运行在应用交付流水线中作为独立的阶段运行，并用自动化将应用程序从一个阶段推送到另一个阶段。

(6) 进程。将应用程序中的服务构建成为无状态进程运行。应用程序的所有有状态信息都应该存储在服务外部的数据存储中。这允许应用程序具有反脆弱性。如果一个服务实例宕机，另一个实例可以在不丢失任何与状态有关信息的情况下替换它。

(7) 端口绑定。让应用和服务使用 URL 访问其他服务，实现所有应用程序服务都可访问。在某种程度上，这扩展了第四个要素。

(8) 并发。在引入微服务时，这一要素所面临的挑战与可扩展性完全一致。确保应用程序中的服务可以同时运行多个实例。这样就能在不影响整个应用程序的情况下，在单一服务粒度上进行弹性伸缩。

(9) 可丢弃。通过设计将“快速启动和优雅关闭”特性最大化，用以实现反脆弱性。简单来说，使服务独立于其他服务，让服务的启动和停止更加快速。这就要求应用程序中的服务所消耗和需要的所有服务都存在，以便实现快速启动、高可用、快速优化。

(10) 开发与生产环境的一致性。很简单，在应用程序交付流水线的所有阶段提供与生产类似的环境。

(11) 日志。将日志视为可以实时监控并分析的事件流，以便向需要数据和信息的所有利益相关者提供快速反馈。

(12) 管理流程。确保管理进程的运行被监控以及应用程序的信息被收集，例如管理 A/B 测试结果、数据清理和运行分析等操作，把它们都作为生产环境中运行的进程，以确保收集数据的准确性。

这 12 要素，看起来似乎就是本书中所列的几个 DevOps 方案的另一个版本。尽管这个方法论是专注于云原生应用的，却能够达到 DevOps 相同的最终效果。利用微服务架构开发应用时，非常适合采用 12 要素应用的方法。

5.6.3 云原生应用

在继续讨论之前，必须恰当地定义在本节中介绍的云原生应用这个术语。云原生应用是将以下概念完美地结合在一起：

- DevOps
- 微服务
- 容器

云原生概念汇集了以上独立的含义，即为云计算交付应用程序，需要一个新的开发、交付和运维的模式，运行在云计算的规模下，需要具有管理编排的能力，以及在这种规模提供反脆弱环境的管理能力。

> **云原生系统属性**
>
> **a. 容器封装。**在软件容器中运行应用程序和进程，作为应用程序部署的一个独立单元，并作为实现高水平资源隔离的机制。全面提升开发体验，促进代码和组件重用，并简化云原生应用程序的运维操作。
>
> **b. 动态管理。**由集中化的编排过程积极计划以及主动管理应用程序。从根本上提高服务器效率和资源利用率，同时降低维护和运维成本。
>
> **c. 面向微服务。**对松耦合的依赖关系进行明确描述(例如，通过服务接口)。显著提高应用程序的整体灵活性和可维护性。本基金会将促进该技术的发展，提高应用管理的技术水平，并通过稳定的接口使该技术无处不在，且容易获得。
>
> ——云原生计算基金会(Cloud Native Computing Foundation，CNCF)，2015

就定义而言，云原生应用是在 PaaS 或 CaaS 上运行的，利用容器部署或运行在云计算模式中的应用。它们是使用微服务架构的设计，并且使用 12 要素应用方法论开发和交付的。小批量、快速交付、可扩展性和反脆弱性是它们的核心主题。

开发及交付云原生应用，要求开发和运维团队要能够充分地理解这些应用与传统应用在几个核心假设方面的差异(Brown，2016)：

- 在云原生应用中，应用和服务本身提供了非功能性需求(Non-Functional Requirements，NFR)，而在传统的应用中，NFR 是由基础设施所提供的。比如负载均衡、高可用性以及应用监控。
- 云原生应用的基础设施是不断变化的(其本质上是灵活的)，而传统应用是运行在具有固定拓扑结构的静态基础设施上的。
- 云原生应用的组件可能跨越多个云环境，分布在全球范围内，作为服务来运行，而传统应用的组件通常位于同一环境中，由本地的服务器组成。
- 云原生应用由 DevOps 团队成员来控制生产环境服务器，而传统应用则是由运维团队来负责运行。对于云原生应用，运维团队运行的是平台或容器服务。这使得 DevOps 团队得以在平台上运行自己的生产环境实例，无论是单独的服务器实例还是容器。
- 云原生应用中，如果发生事故，DevOps 团队会负责确保应用程序保持运行状态。而在传统应用中，从开发到生产环境有一个正式的交接时间，在此之后，运维团队将尽可能减少开发投入。同上所述，对于云原生应用，运维团队运行的是平台或容器服务。只要核心的平台服务是可用的，他们就算很好地完成了使命。如果在该平台上运行的应用程序崩溃了，那么拥有这个应用的 DevOps 团队来负责修复。

开发和交付云原生应用不仅仅是一种架构或方法论的转变。为了有效、高效地开发并交付这些应用程序，整个团队的构成、团队成员的角色以及需要的技能，都需要改变。

5.6.4　微服务和容器

有足够的证据表明微服务和容器是相互关联、相互促进的。尽管二者之间不存在依赖关系，容器不需要为微服务打包而存在，微服务不必通过容器方式部署，但使用容器方式进行构建、部署和运行微服务的价值是极大的。主要表现为以下三个方面：

(1) 环境抽象。根据定义，容器使得在其内部运行的应用从容器运行的环境中抽象出来。主要体现的是容器的可移植性。而微服务总是以同样的方式利用环境中的服务，不论容器在什么样的环境中部署。

(2) 细粒度执行。由于容器比完整虚拟机轻得多，因此它们能以更低的成本进行弹性伸缩。因此，正在运行的微服务可以更容易地水平扩展，不会降低底层的基础设施的效率。

(3) 隔离性。微服务在容器中运行的方式是可以作为相互隔离、彼此独立的程序运行，不受正在使用或与之交互的其他服务的运行位置及运行方式的影响。多个微服务可以运行在同一个服务器的实例内，也可以分布在多个服务器实例中，而不必担心多个微服务之间相互干扰。由此，可以更好地规划并管理负载均衡，以最大限度地提高服务器利用率，而且，通过运行跨多个服务器实例的同一微服务的多个实例，能够更好地实现微服务冗余。

5.6.5　微服务化改造

简化橄榄球战术口令

厄尔哈德-珀金斯体系的精髓是战术，特别是传球战术，不是通过路线树或者单个接球手的路线来组织的，而是通过教练称之为“组合概念”(concepts)来命名的。每个战术都有一个名字，而且这个名字会让人同时联想到四分卫和其他的进攻球员。而且，最重要的是，这个概念几乎可以在所有的阵型中运用。无论哪个角色做出什么改变，理论和战术驱动始终是保持不变的。珀金斯声称：“本质上而言，使用的都是同一个战术，只不过是给他们进行一些装饰，让其看起来有所不同而已。”

基于概念体系最大的优势在于它是从进攻中最关键的四分卫球员

的角度出发的。在其他体系中，即使基本原则是完全相同的，战术内容及名称也可能会完全不同。不需要接收理解所有的实时信息，厄尔哈德-珀金斯体系下的四分卫只需要明白接球手的路线即可。爱国者队前进攻教练查理·维斯在 2000 年的时候曾说过："你可以简化战术口令，基于阵型和球员阵容采取不同的方式，这样便于球员学习，更便于四分卫掌握"，"你可以在不影响其理解的情况下改变战术口令的表达方式，而不需要无穷无尽的战术数字。"

—— Brown C., 2013

为了总结交付微服务架构的方案，让我们回到方案的核心主题：开发和交付微服务架构。这里有两种场景：

- 开发和交付新的云原生应用
- 将现有的单块应用迁移到微服务

第一种情形可以利用微服务、12 要素以及前面提供的云原生应用的指导原则来实现。这是一个全新的应用程序，尽管可能会用到现有的应用程序和服务，但它是从头开始构建的，不会局限于现有架构，也不会受现有代码及数据的约束。

第二种情形，就如同开发一种新的橄榄球战术口令系统。目标是简化现有的架构，将代码和数据放入结构化的组件中，这些组件是可以迁移到能够分别映射到微服务的那些组件的。然而，单块应用的架构拆分和存储在数据库中的现有数据成为主要挑战的来源。如图 5-12 所示，应用和数据需要进行重构。

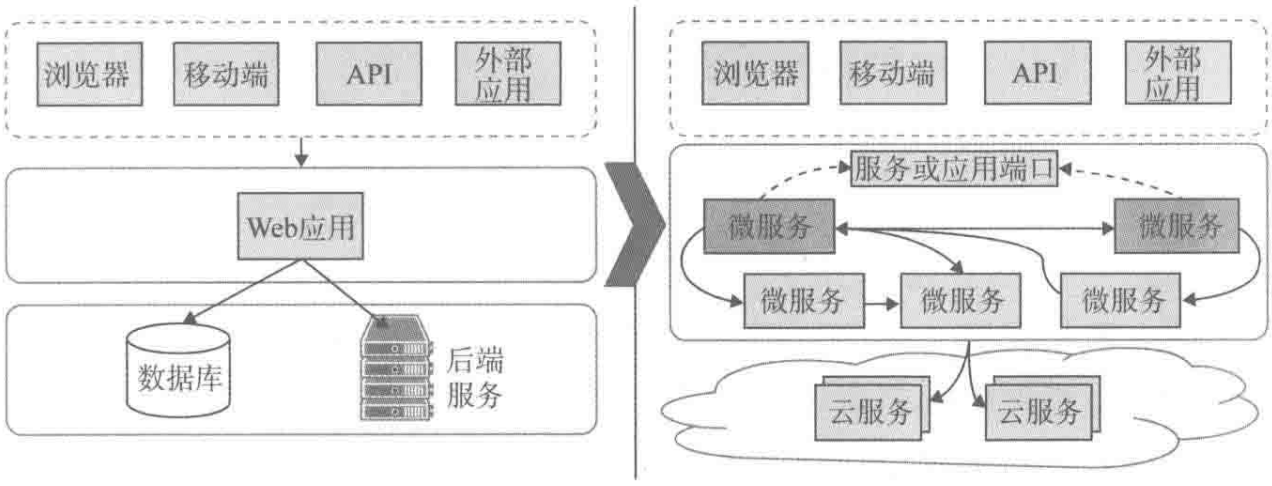


图 5-12　重构到微服务架构

此外，康威定律利用现有的团队架构才能发挥作用，为了匹配重构的应用，则可能需要打破现有的团队架构。

以下是将传统的单块应用重构为微服务的核心步骤(Brown，2016)：

- **重新打包应用。**让我们以一个 Java 应用程序为例。大多数 Java 应用程序会作为一个大的 EAR(Enterprise Application Archive，企业应用程序归档)文件部署到 IBM WebSphere 应用服务器、Oracle WebLogic 或 Apache Tomcat。这些 EAR 文件通常由多个 WAR(Web Application Archive，Web 应用程序归档)文件组成，这些文件是由不同开发团队开发并集成到 EAR 文件中的。如图 5-13 所示，将 EAR 文件分解成单独的可部署 WAR 文件，把每个文件作为一个微服务，这是良好的开端。虽然通过解耦基层组件，并不能解决架构的复杂性问题，(换句话说，它并不能基于康威定律解决架构依赖增加的问题)，但确实实现了让每个组件能够独立于其他组件进行构建、部署和管理，这是微服务的关键要求。

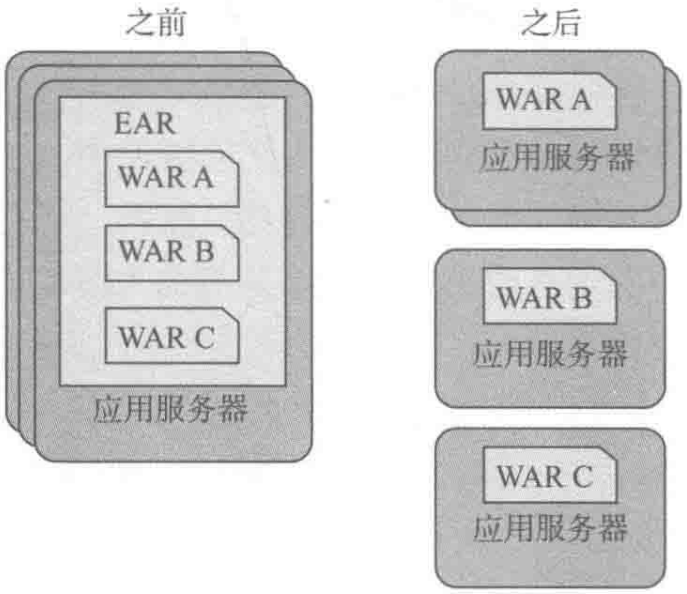


图 5-13 将 Java 应用重新打包为微服务

- **重构代码。**要提取那些没有意义的架构依赖项，并将它们分解为细粒状的微服务，这项工作需要查看代码，以确定嵌入在现有组件中的细粒度的服务化行为的位置，然后将代码重构到组

件中。为了创建完整的微服务，如果接口不存在，该组件还需要配备一个明确的 RESTful 接口。这种重构工作可能会相当复杂，根据所使用的设计模式不同，应该得到相应的处理。

- **重构数据。**迁移现有的应用程序到微服务最艰难的挑战根本不是代码，而是数据。多年来，现有的应用程序在数据库和应用程序组件之间存在大量且通常是多个数据源和复杂的数据流，这使得理解存储的内容和逻辑成为复杂的任务。然而，当涉及将应用重构为微服务时，重构数据结构模型成为其中的关键。使微服务的行为和操作独立于其他的微服务，它们需要使用正确的数据结构进行交互，将数据保存在正确的数据存储中。为了实现这一目标，所有的数据模型和数据存储决策都需要重新考虑：
  - 正在使用正确的数据结构吗？
  - 使用了正确的数据存储吗？
  - 有使用正确的方法进行数据查询吗？

在过去，关系数据库是唯一的选择。这导致所有的数据格式，从二进制数据到结构化 Java 对象再到图形数据模型，都被存储在关系数据库中。此外，为了节省空间，这些关系数据库被重度标准化。所有这些都导致了过于复杂的关系数据结构，二进制文件被存储为不能有效查询的大型二进制文件，或者极其复杂的查询图表。今天多语言数据存储，从对象存储到 JSON 文件存储再到图形数据库，其中存储的数据最佳匹配了格式和查询需求。同样，这种重构并不简单。事实上，由于可能涉及的数据结构类型、数据模型和数据存储的复杂性，它通常比重构代码要复杂得多。然而，要获得真正的微服务架构，数据重构是必须完成的任务。

5.7　方案：API 经济

笨蛋！问题就出在经济上

世界足球俱乐部由于无法再从传统业务中增加收入，已开始实施战略转型，将自己转型为现代体育传媒公司。在现任总裁弗洛伦蒂

> 诺·佩雷斯(Florentino Pérez)的领导下，西班牙皇家马德里足球俱乐部成为一个让人大开眼界的成功案例。这种模式的基本支柱之一就是要设计并实施新的市场营销战略，以提升俱乐部的品牌价值。通过营销，这种模式带来了收入的大幅增加。无可否认，在这一领域，皇家马德里已经成为世界足坛的领袖。
>
> —— fordacell，2006

就像足球俱乐部寻找创新的方法来开发新的盈利模式，以提升其品牌价值一样，API 也可以帮助组织寻找超越其核心竞争力和能力的创新商业模式。

API 允许应用程序和服务将其功能和能力暴露给团队外部(不仅是组织内部而且是组织外部)的应用和服务。API 有时也被称为内部 API，是在组织内部使用的 API，而外部 API 是为外部提供的。当然，安全性、管理和计量要求根据 API 的目标受众而有所不同。

API 应该是应用程序和服务相互通信的默认机制。如果你有了基于微服务的架构，这就是默认模式。在遗留应用中，可能存在应用程序之间点对点定制化的集成场景。这些场景应该使用友好定义的 API 替换。

API 并不是最近才产生的。这个概念已经存在了很长一段时间，在面向服务的体系架构(Service Oriented Architecture，SOA)开始流行后更是发展成了主流。事实上，经常有关于 SOA 和 API 之间的差异的争论。它们以最纯粹的形式提供了一种机制，将应用程序与服务解耦，这些服务具有定义良好的架构接口，用于与其他服务进行通信和连接。根据定义，这个结构化接口是一个 API。目前针对 API 和传统 SOA 接口的主要区别是 API 与 REST(Representational State Transfer，表示状态迁移)/JSON(JavaScript Object Notation，JavaScript 对象标记)接口相关联，而 SOA 则与 XML(Extensible Markup Language，可扩展标记语言)和 SOAP(Simple Object Access Protocol，简单对象访问协议)相关联。相比于 SOA，API 显得非常轻量级、灵活，易于管理和使用。

API 允许开发人员创建新的应用程序，利用内部和外部的 API 为应

用添加更多的服务和创新驱动机制。通过以下两种方式来实现：

- **引入生态系统。**组织可以通过使用来自外部第三方服务的 API 开发新的创新型应用程序来实现目标。如果有第三方公共 API 提供服务，为什么要构建自己的地图服务？为什么构建一个用户身份管理服务，如果组织中的另一个团队已经建立了一个用户身份管理服务并通过 API 公开该服务？图 5-14 显示体育网 ESPN 的公共 API，它允许开发者在他们的应用程序中加入运动数据。

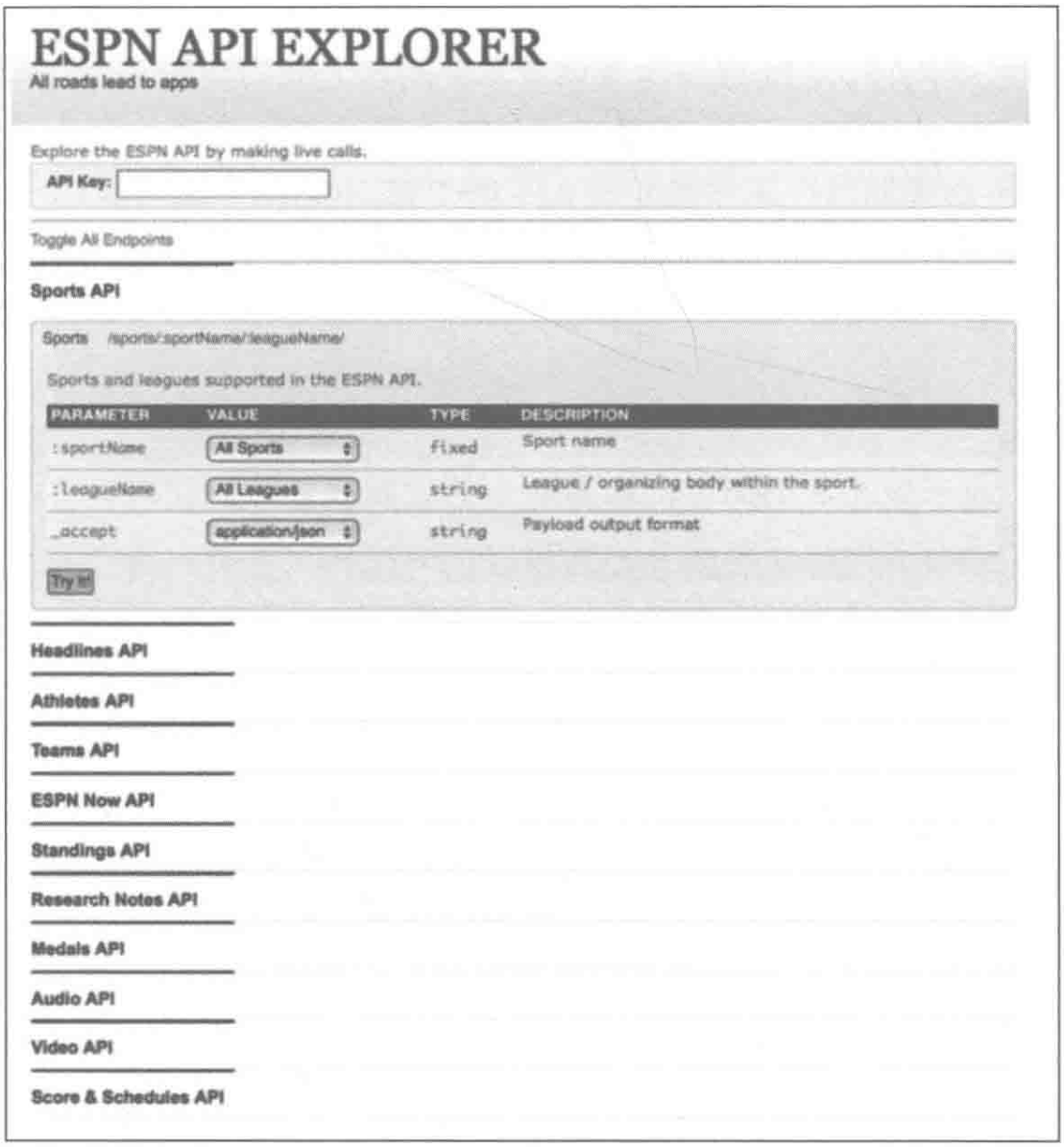


图 5-14　ESPN 公共 API(ESPN 开发者中心，2015)

- **通过生态系统弹性伸缩。**除了利用自身现有商业模式盈利外，

企业现在还可以通过 API 将业务能力暴露在更广泛的生态系统中，从而实现自身业务能力的商业化。如果银行已经开发了一种服务，能够高效、准确地计算出“金融风险”或金融衍生产品交易的风险，为什么不将它商业化，允许外部合作伙伴付费使用该服务 API 呢？

5.7.1 部署自动化和 API

从 DevOps 的角度而言，API 与服务具有相同的功能。无论是否商业化，组织都需要将 API 视为一种产品，对于“外部”API 更应如此。即使在组织内部，API 应作为提供者和使用者之间的契约，而且理应如此。为了应用交付的目的，需要交付 API 并提供 API 公开的服务。然而，服务的生命周期现在还包括处理 API 的步骤。例如:

- 只是测试应用程序或服务的功能、性能和安全性是不够的。还需要对 API 进行测试，通过用户如何使用 API 的用例来进行测试。
- 应用程序和服务的配置管理通常也包括管理相关的中间件的配置。现在，API 和任何 API 管理软件的配置管理问题也应该得到解决。
- 对于发布管理而言，业务能力的发布就是针对交付业务能力的所有应用程序和服务的发布。在包含 API 之后，发布版本中还需要包含所有应用程序和服务需要的第三方 API。管理这些 API 的 SLA 成为发布管理团队备受关注的任务。

在本章有很长一部分是关于微服务架构持续交付的内容，其中讨论了 API 的核心价值主张。API 允许应用程序和服务解耦。因此，它们可以独立地构建、部署和运行，而不必关心所消费的其他服务的部署位置。关键在于 API 的可用性。应用程序和服务也可以通过部署更多需要的实例进行扩容，而不会影响到其没有扩容需求的其他服务。因此，API 才是真正的持续交付，以快速的反馈周期来支持小批量交付变更。

5.7.2 DevOps 平台和 API

API 的作用也是一个重要组成部分，这是因为 DevOps 平台本身也可通过 API 进行访问，无论是通过 IaaS、PaaS 或 CaaS 实现的。让我们

看看 API 对这些变体的作用。

- 如果使用 IaaS 云计算来实现 DevOps 平台，那么所有的 IaaS 服务都是以 API 形式呈现的。交付至平台上并在平台上运行的应用都是通过 API 方式来使用这些服务及其所提供资源的。

我曾在之前的“云编排”内容中提到过，使用可移植的 Heat 模式在多个云中进行全栈配置的案例，通过 OpenStack API 方式可以在多个云上配置 HEAT 模式。图 5-15 展示由 OpenStack 提供的网络 API 实例。

POST　/v2.0/networks

Create network

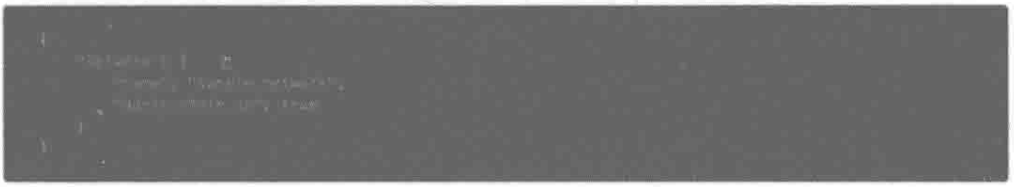
Creates a network.

A request body is optional. An administrative user can specify another tenant UUID, which is the tenant who owns the network, in the request body.

Error response codes: 201,401,400

Request

Request Example



Response Parameters

Name	In	Type	Description
status	body	string	The network status.
router:external (Optional)	body	boolean	Indicates whether this network is externally accessible.
availability_zone_hints	body	array	The availability zone candidate for the network.
availability_zones	body	array	The availability zone for the network.
name	body	string	Human-readable name of the resource.
admin_state_up	body	boolean	The administrative state of the resource, which is up (`true`) or down (`false`).
tenant_id	body	string	The ID of the tenant who owns the resource.
mtu	body	integer	The MTU of a network resource.
qos_policy_id (Optional)	body	string	The UUID of the QoS policy.
subnets	body	array	The associated subnets.
shared (Optional)	body	boolean	Admin-only. Indicates whether this network is shared across all tenants.
id	body	string	The UUID of the network.
network	body	object	A `network` object.

图 5-15　OpenStack 网络 API(OpenStack.org,2016)

- 如果 DevOps 平台是由 PaaS 或 CaaS 实现的，那么处理过程就是通过访问 API 来实现。在这种情况下的 DevOps 工具是在 PaaS 或容器中运行的服务。这些服务被连接和编排，以便通过 API 方式实现应用程序交付流水线。

5.8 方案：组织创新

发展前景

多米尼加共和国和尼加拉瓜人口大致相当，对棒球有着同样深厚的热情。但多米尼加共和国已派出超过 100 名球员到棒球大联盟(MLB)的球队，而尼加拉瓜仅有三个一流球员。

尼加拉瓜正在寻求变化缩小差距。而第一步可能是今年初的尼加拉瓜棒球学院开幕。丹尼斯·马丁内斯在 1976 年成为推动主流(加入棒球联盟)的尼加拉瓜人，他帮助培养在棒球方面有志的青年，他有着吸引年轻人注意力的能力。

不仅成功促进新一代尼加拉瓜球员增加，而且为该国经济发展引入了一些投资。而多米尼加共和国就是另一个很好的例子。

在多米尼加共和国，投手、击球手和外野手是主要的援外球员，为国家赚取了大笔外汇，同时为运动员家庭带来大量汇款。多米尼加的超级巨星，像大卫·奥蒂兹、罗宾森·坎诺和萨米·索萨繁荣了美国职棒大联盟体育场，也促进了旅游产业在他的家乡的发展。

同样重要的是，棒球已经成为数千名多米尼加青年摆脱贫困的途径，是一个健康的取代帮派和犯罪的职业。

尼加拉瓜的马丁内斯，曾于 1991 年成为唯一的拉丁美洲裔打出完美比赛的投手，可以断定尼加拉瓜青年，像他们的有才华和抱负的多米尼加同仁，会在美国职棒大联盟取得成功。挑战只在于开发合适的商业模式。

—— Feinberg，2011

如何鉴别和培养职业球员？只有掌握了球员大量的跟踪统计记录可以作为决策依据时，电影《点球成金》中的模式才会奏效。你如何在成千上万的正在努力变得出名的年轻球员中确定他们的前景，并投资于他们？

创新也是一样的。对于每一个创业公司，都想成为 Uber 和 Airbnb，

以及 Facebook 和 PayPal，甚至在它们之前的微软和苹果，但有成千上万的创业失败。有些人从来没有度过创始人团队的组建阶段，而另一些人甚至在通过 IPO 后离开或被收购而退出。当然，你不会听到那些没有成功的人的故事。

在大型组织中，情况更糟。很少有像谷歌这样的大公司，从未失去创新文化，并有机制来鼓励员工的新想法(He，2013)。大多数大型组织不是为创新而设计的。他们没有创新的文化，事实上，大多数组织恰恰相反。员工被严格的管控和绩效管理措施所扼杀，鼓励他们做规定的工作，保持在“封闭状态”，尽最大努力工作，不鼓励新思想。当然，这对于当今竞争激烈、瞬息万变的世界来说，并不是有价值的商业模式。如果不进行创新开发和实验，那么拥有最佳的 DevOps 平台和高效的为创新而设计的技术平台，是没有任何价值可言的。

创新的故事从未改变。从来都是只有一小部分人拥有新的创意，却通常不被周围的人和他们的管理者所理解。

——埃里克·施密特，谷歌董事长

从 DevOps 实施的角度来看，文化是至关重要的，并且创造文化胜过所有可以通过 IT 的团队实施的技术以及流程的改进。本章中讨论的所有主题和方案都是为了让埃里克·施密特团队进行讨论的。这些团队提出了创新的想法，并努力进行实验，从中寻求商业机会以及创新的想法是否具有可行性。如果组织中的文化甚至不允许团队以他们所需要的自由的方式和资源来处理这些想法，如果他们周围的人和埃里克提到的管理人员不能发展出能够让思想进化、繁荣和非问责的文化，那么所有这些都是毫无意义的。这种文化在一个有着巨大文化惰性的大组织中是不容易形成的，但却是必不可少的。大型组织通常能够在监管以外的小团队中做到这一点，但这只对孤立的小项目有效。为了实现全组织的创新文化，需要在企业范围内对组织的能力进行创新。

发展大型企业的创新文化

> 许多运动，不只是橄榄球，都具有一种男子气概的固执心态，在他们看来创新几乎是不能接受的。
>
> ——劳伦斯·杰克逊，前美式橄榄球运动员

麦肯锡公司已经就如何建立和扩大这种创新文化提出了四种行之有效的方法。他们在《数字化颠覆》中描述如下(Edelman，2015)

(1) 组织枢纽。组织让创新的领导者管理整个组织，而不仅仅是推动创新。这使得组织能够发展创新优先的文化，这也加速了遗留系统的优化。

(2) 反向接管。这是一个更激进的做法，创新领导者接管遗留应用和系统，并将其流程、技术和团队转变为创新团队的模式，从而在整个组织范围内实现流程、技术以及文化的创新。

(3) 分拆。这是一个切割的方法，是将注重创新的团队组合成单独的、独立分工的团队，让他们成长、繁荣和发展正确的流程、IT 平台和自己的组织文化，并且排除其他组织文化惰性的干扰。在一些极端的情况下，这些部门也被分拆为独立的公司或独立的子公司。

(4) 背驮式。这是一种合作的方法，组织与另一家拥有必要的创新型技能、专有技能以及创新文化的公司联合起来。然后，合作伙伴开始提供创新型系统来完成组织原有的遗留系统，而组织本身不需要进行变革。

在接下来的一章，将讨论同时为优化和创新而采用 DevOps 文化的特定团队模式。

5.9 总结

本章重点讨论了 DevOps 方案，是在具有创新性目标的项目和应用程序中实施 DevOps 成功的模式。值得注意的是，在创业公司之外，大

多数组织很少有单独运作的创新项目。因此，每个创新产品都依赖于现有的应用程序和服务，这些应用程序和服务在本质上并不是创新的，而是提供核心业务服务。因此，大型组织的真正的 DevOps 实施将总是同时包括优化和创新的方案。

在本章中介绍的四个方案是：

- 构建 DevOps 平台
- 交付微服务架构
- 发展 API 经济
- 组织创新

这些方案有助于实现本章开头所强调的四个主题：

- 实现多级 IT
- 构建正确的事物
- 进行实验
- 实现反脆弱性系统

最终，实现这些主题才是成功的关键。这些都是我们正在努力实现的目标。

多级速度是现实情况。在大多数大型组织中，广泛存在着技术栈、实践、团队成熟度、管控和合规性要求，以及业务驱动应用程序在交付应用程序和服务时产生多级速度。

一个非常重要的项目失败了，原因在于构建了错误的东西，尽管构建本身是正确的。遗憾的是，大多数组织都在改进开发和交付过程，而不是在设计实践，更不用说采用精益创业运动中的技术。不仅利用实验来发现正确的东西，而且用正确的方法来构建它。

最后，作为一个可以快速实验的构建系统，需要处理大量的使用和负载，并实时弹性伸缩，以及提供反脆弱的系统，使系统在混乱中茁壮成长。

第 6 章

DevOps 的企业级推广

创建成功文化

从一个积分垫底的球队到一个顶级球队，这样的期望让人望而生畏。屡屡失败的时候，很难想象自己还可以赢。因此，即使直面球员们的弱点，我也总是试图建立一种成功文化。这不是一蹴而就的事情，只能一步一个脚印，就如同达阵要一码一码地移动一样。

我的哲学是：想要赢得比赛，就要相信团队有获胜的能力。也就是说，信心是与生俱来的。这可能听起来自相矛盾，但重要的是要记住，即使很小的成功也需要有强大的力量来帮助人们建立自信。

因此，在训练营中，我们并没有把注意力放在赢得超级碗的终极目标上。我们设立了一套触手可及的明确目标：我们要成为智慧的团队；我们要成为健康的团队；我们要成为努力的团队；我们要成为自豪的团队；我们要成为集体获胜的团队；我们要成为不会互相指责的团队。

当开始执行这些目标时，我确定每一个人都是清楚目标的。我在每一个可能的机会中强调积极的方面，同时强调需要完成的下一个目标。如果我们表现特别好，我会把团队召集起来，对他们说：“我们今天做了一些事情，我们执行得非常好。我对你们的工作很满意。但是明天我要做的是：我想看到一个完美无缺的专业团队。如果你们做到了，那么我们就为周日的比赛做好了准备。”

当设定的是小而且看得见的目标，人们实现目标的时候，就会开始

> 意识到自己是可以成功的。他们打破失败的习惯并开始培养成功的习惯。看到球队在对自身的看法上发生这种转变，是非常令人满意的。
>
> —— Parcells，2000

本章所列方案探讨的是如何在整个组织范围内推广 DevOps。大型组织在本质上并非是一个纯粹的整体。组织内部有很多小的部门或业务单元，每个部门或业务单元都有自己的平台、流程、成熟度、政策以及文化。这些组织可能通过收购及兼并来发展壮大。这些收购或兼并的组织作为几乎独立运营的子公司在母公司中继续存在的情况也并不少见。有些时候，母公司是有意把收购的公司仅仅作为一个全资投股子公司来运营，而无意将其整合到母公司。大多数情况下，付出努力的结果是整合失败。失败的原因或许是由于计划或执行不力，更多的则是因为兼并组织的领导层对整合投入不足，真正的整合是文化的切实改变。整合或标准化技术相对容易。规范文化却并非如此。

此外，正如第 3 章与第 4 章所讨论的，大型组织通常有抑制速度与创新的严格而复杂的管理实践。这些实践是用来管理及控制有很多不确定因素的庞大组织，而非用来实现敏捷与创造力。因此，这类组织通过在小型团队所服务的小型项目上实施 DevOps，见证了成功。这些项目可以是“在监管以外的小团队”，或是受到负责项目的管理层保护，使其得以在传统的管理体制与流程之外运行。他们可以灵活与创新。然而，不改变管理模式或者组织层面的文化，这种成功无法推广。本章的方案旨在实现这种推广。

6.1 核心主题

和前两章一样，本章也来先看几个核心主题。本章所列方案旨在在大型的、潜在的分布式组织中推广 DevOps 实施，这些主题贯穿始终。

6.1.1　组织文化

> 文化导致积极结果。这并非成功路上的事后聪明。冠军在成为冠军之前就表现得像个冠军；他们获胜之前就按获胜的标准行事。
>
> ——比尔·沃尔什，美国橄榄球教练

本书中曾多次重申：DevOps 的核心是文化运动。DevOps 的首要目标是要在应用交付流水线中的不同利益相关者之间建立起信任、沟通与协作。所有其他目标都是次要的，或者是为实现首要目标服务的。如果组织拥有完全自动化的、集成的交付流水线、按键式自助服务环境以及高度优化的流程，但是开发、测试人员与运维人员从不协作，开发、测试人员从不将要部署并将影响其开发-测试-生产环境的变化对运维人员给予指导，开发、测试人员从来收不到来自运维团队的反馈，那么其所交付的应用将无法充分实现其业务目标。

这成为大型组织的一项主要挑战。正如第 2 章所述，这类组织有根深蒂固的文化惰性。他们利用复杂的、官僚的组织架构以及管理流程进行管理。长久以来，这些架构用以管理、维护和监督大型组织的运作。组织按照代表业务线的部门进行分解，或者是按照狭隘的组织路线进行任意划分，比如那些代表收购或兼并业务的组织。这些部门可能按照功能仓筒进一步细分。可能还有其他一些操作仓筒，代表着共享服务，即多条业务线所共用的服务供应商。

创造一种打破所有这些界限及仓筒的文化不是一件简单的事情。需要组织的最高层级管理人员的领导与支持。只有最高层级才能影响并促进组织的所有部门与仓筒的改变。只有最高层领导才能为实施 DevOps 流程与实践的团队提供必要的“空中掩护”，使其脱离组织传统的管理机制控制以及固有文化的影响，从而克服文化惰性。

这里要记住的最重要的一点是，文化转型不是一蹴而就的事情。这也不是一个一劳永逸的项目。文化惰性也会在新的文化环境中发展，带来新的挑战，流程僵化，人们按照自己的方式行事，导致即使在最好的 DevOps 文化下也会效率低下。真正的文化变革需要伴随着文化的持续

改进，即今天要好于昨天。

6.1.2　工具与实践标准化

> 流程标准化之前是不可能进行改进的。如果流程不断变化，那么任何改进终将只是偶尔使用，更多则是被忽略的另外一个变化而已。在进行持续改进之前，必须通过标准化保证流程的稳定。
>
> ——今井正明，日本组织理论家及改善咨询集团创始人

持续改进要求在实施之前先做好标准化准备。正如今井正明所言，如果流程在每一次实施时都有变化，那么改进工作是无法进行的。为了最大限度提升流程效率而产生的改进需求也将无法确定。还有就是，对流程进行了改进，但是流程的实施方式与改进前不一致，那么就将无法衡量这些改进是否对流程产生了影响以及影响程度如何。现在把团队数量以及是否每个团队都依照自己的差异及喜好来实施流程这些情况综合起来考虑，就会发现改进需求几乎无法确定。这里需要声明的是，不是说一定要严格遵循流程，因为这本身就与敏捷原则相悖，但是，变化需要有适当理由并且在合理范围内，为团队或项目提出的特定需求定制流程就是这样的实践。

总之，需要将流程标准化为最小集合，团队可以从中选择满足其需求的最佳标准化流程。组织需要根据不同项目的风险值分布来确定所要准备的标准化流程集合数量。这里所阐述的对于流程的思考同样适用于用以实施以及自动化这些流程的工具，这些工具同样需要自动化。

此外，人员需要是可以互换的。[①]“人员互换”这样的表达似乎很冷酷，并且在政治上是不正确的，但对于敏捷组织而言却是十分必要的，这些组织拥有多个项目及交付成果，人员需求不断变化。设想一个拥有特定技能的运维人员。如果一个团队不再需要这些技能，但是另一个团队需要，要付出哪种程度的努力才能把这个人从第一个团队转到下一个

① 互换：是在契约平等条件下，一个部分或数量可能被另一个相等的部分或数量所取代的这样一种性质。

团队呢？是否需要进行流程培训？是否需要进行工具培训？运维人员需要多久才能正式加入新团队？流程与工具的标准化使这种转换得以无缝进行，从而实现运维人员在团队与项目间的互换。

6.1.3　有组织的实施

> 冠军的辉煌不是在赢得比赛的时刻才铸就的，成就冠军的是准备比赛时每小时、每周、每个月以及每一年日积月累的付出。获胜的表现本身不过是其冠军品质的一种彰显。
>
> ——T·艾伦·阿姆斯特朗，美国教练

如前所述，DevOps 实施并非一劳永逸的项目，需要持续不断的努力。在大型组织中实施时，必须有组织地进行，以确保流程与工具的适当标准化以及生产率损失的最小化。正如前面章节所讨论的，引入任何变更都会导致生产率的下降，即使变更是积极正向的，因为运维人员需要做出调整以适应变化以及新的工作方式。以有组织的方法来引入变更，辅以适当的强化与训练，才能确保生产率降幅最小。

要以有组织的方式进行，还有另外一个原因，那就是驱动变更只需投入最少的资源。只有那么多的领导者可以提供指导，也只有那么多的流程与工具专家能够帮助推进流程与工具的实施。组织承受变化及相应的生产率流失的能力也就只有那么大。因此，变更应在企业内滚动实施，每次只针对一组项目或团队。每次实施的规模根据可用的强化与培训资源而定。

由于组织的各个团队、部门以及不同地域的运维人员还有流程成熟度可能都存在着显著差异，因此还必须要制定相应计划来处理这些差异。可能需要在不同的部门与地区运行一系列试点项目，证明要实施的新流程与工具可以在差异环境下运行，如果无法运行，则需要在哪些方面进行改变与增强才能使其适应这些差异。例如，相比于运维人员经验丰富的小型团队，运维人员更多为初级水平的大型离岸团队需要更多的训练与强化。基于项目需要而在旧工具集上工作的团队可能需要先升级到中级工具，然后才能到最高级别的自动化。比如，要将“绿屏”

(green screen)上工作的 COBOL 程序员迁移到基于 Web 的 IDE(Integrated Development Environment，集成开发环境)，首先要将其先迁移到基于 Eclipse 的桌面 IDE。

6.1.4 打破组织仓筒

要打破大型组织中的组织仓筒是极其困难的。组织中存在着阻碍变革的政治力量。一些行政领导与高管人员的权力结构、影响力以及“领地”会在探索组织变革时受到威胁。极少数有政治领导力的组织才会自上而下地重组整套汇报结构。在应用交付流水线中向其汇报的所有功能团队中创建新的高层管理角色是极不寻常的，能成功执行就更加不寻常了。

打破组织仓筒势在必行，但是在大型组织中要不破坏现有的汇报结构才行。矩阵团队中，运维人员保持原有的汇报线，但在功能团队中工作并向矩阵团队负责人汇报，利用这种矩阵结构可以替代性地解决这个问题。运维人员只要知道自己的项目团队并能够自由地合作，同时也能获得团队领导的指导，向哪位副总裁汇报就真的无所谓了。如果沟通必须通过“适当的管理链”，那么矩阵模式就行不通了。运维人员需要与团队自由沟通及协作。自由地交流，所有团队成员的工作具有可见性，才能建立起彼此的信任。

6.2 方案：DevOps 能力中心

印度羽毛球职业能力中心

如果说有一个人和一个学院帮助印度培养出世界级的羽毛球运动员(shuttlers[①])并使之成为羽毛球的温床，那么一定是戈比昌德和他的羽毛球学院(位于海德拉巴)。悉尼奥运会惨痛失利 16 年后，戈比昌德几乎实现了他的奥运梦想，尽管角色不同。

① shuttlers 指的是羽毛球运动员。

辛杜在里约奥运会上的惊人表现让其著名教练和他的学院成为焦点。辛杜是 Pullela Gopichand 羽毛球学院培训出来的一名运动员，在女子单打比赛中摘得银牌，创下了历史纪录。

辛杜是继塞娜·内维尔之后第二位叱咤羽坛的女子选手，并为前全英公开赛冠军所设立的学院带来了荣誉。

分析人士称，印度在羽坛势不可挡，要归功于 42 岁的戈比昌德，他培养出了世界级的人才。

戈比昌德一直以来的梦想就是培养奥运会奖牌得主。随着辛杜在 2012 伦敦奥运会上摘得铜牌，他的付出开始有了回报。

辛杜成为印度第一位实现此壮举的女子羽毛球运动员。四年后，辛杜闯进决赛并且只输给了世界排名第一的卡罗琳娜·马林，戈比昌德再一次梦想成真。

—— 体育校园，2016

驱动大型组织实施 DevOps 文化，也许要做的最重要投资就是设立 DevOps 能力中心。

印度是一个在运动方面没有什么遗留财产的国家，却培养出两位女子羽毛球的奥运会奖牌得主，这都要归功于教练戈比昌德以及他的羽毛球学院，一个羽毛球职业能力中心。教练戈比昌德本身也是一个世界级的羽毛球选手，在自己的职业生涯中错失一枚奥运奖牌，却创造了专业知识与能力的源泉，训练羽毛球新选手遵循行之有效的方法论，以发展成为世界级的奥运奖牌得主。同样，DevOps 能力中心的服务宗旨也是成为专业知识与能力的源泉，为希望实施 DevOps 的组织进行团队培训。

首先要定义何为能力中心。

能力中心

企业用以协调 IT 技能的组织架构。能力中心提供项目专业知识或程序支持，既是知识存储库，也是多个业务领域的资源库。信息服务组织中最常见的是技术型能力中心，用于应用开发、软件语言技能、数据管理、互联网开发以及网络设计。在企业内部，应用于物流、财务以及

人力资源的能力中心(或共享服务)越来越普遍。基于知识库的能力完全是作为信息来源。

—— Gartner，2016

能力中心(Center of Competency，CoC)不是一个管理机构，或者一个工具/能力群，而是供 DevOps 实施者向专家学习、彼此学习并分享所学知识与经验教训的地方。随着组织实施并不断推广 DevOps 实践，能力中心也成为 DevOps 知识、能力、工具指引甚至推动力的来源，并且成为帮助团队和项目实施 DevOps 的教练。能力中心也掌握着组织的 DevOps 架构与方法论(他们认同的 DevOps 的风格)。

6.2.1 DevOps 能力中心的功能与目标

能力中心需要有明确的章程以及组织目标。还要成为多元化组织，拥有自己的功能以及实现这些功能的一系列目标。如果利用本书，能力中心还要负责利用所列出的方案开发组织的指导手册，理解并掌握方案的实施，其中包括：

- 为 DevOps 实施提供思想指引。
- DevOps 变革期间提供指导。
- 帮助项目团队遵循 DevOps 实施路线图开展相应工作，这个实施路线图是利用价值流图这类的技术专门为项目团队开发的。
- 通过管理，促进组织内部、团队之间以及与管理层的沟通。
- 在组织内设立 DevOps 实施小组。实施小组应该设立一个虚拟门户，通过虚拟门户，最佳实践、支持以及其他资产得以共享，运维人员也可以参加论坛。还应该设立定期面对面的本地会面。
- 驱动变革并实现持续改进。
- 随着 DevOps 实施在组织内的推广，就实施进展情况及实施成果提供可见性。
- 获取并提供监测成功的衡量标准与指标。
- 记录并沟通最佳实践。
- 促进或提供开发平台通用工具。

- 使跨功能仓筒的利益相关者都参与进来以驱动 DevOps 实施。
- 在组织内传播并庆祝成功以赢得怀疑论者以及落后分子的认同与支持。
- 作为永久性的组织存在，以确保 DevOps 实践被视为一种持续的、不断改进的努力，而非一次性的项目。
- 通过 DevOps 教练培训项目团队。
- 为驱动长期变革的高层管理人员提供培训，确保拟定适当的规划并进行适度的投资以实现成功。

6.2.2　能力中心的核心角色

DevOps 能力中心需要有一些明确定义的角色来实现其成功定位：

- **项目经理。**随着越来越多的项目开始利用能力中心来协助驱动 DevOps 的实施，这就需要项目经理帮助管理跨项目部署的资源，以及各类项目的不同需求。
- **执行经理。**执行经理负责推动评估、工具解决方案、项目实施 DevOps 情况评价这些工作的执行。
- **基础架构经理。**基础架构经理负责建立并维护工具，以向项目团队交付 DevOps 平台。DevOps 平台可能隶属于能力中心，由能力中心进行管理，也有可能是由单独的工具或环境团队进行管理。
- **DevOps 教练。**接下来的章节会对这个角色进行更详细的说明。
- **技术布道者。**顾名思义，技术布道者负责驱动创新、沟通，并分享 DevOps 学习及成功的故事。

6.2.3　DevOps 教练

教练能够成就或者摧毁一个奥林匹克运动员

超级运动员与那些参与奥林匹克角逐却空手而归的精英选手究竟有什么分别？新的研究表明，这可以归结为教练与运动员的关系。根据 11 月在伦敦举办的世界级体育会议上的调查结果，超级运动员觉得他们的教练除了提供高超的技术支持外，还作为朋友、导师以及坚定不移的拥护者，完全满足了其情感方面的需求。未摘得奖牌的精英运动员却没

有这种感受。威尔士班戈大学的博士后研究员马修·巴洛(Matthew Barlow)引领了运动心理学的研究，并指出："要取得最佳成绩，就必须简化最好的技术与战术教练与最好的运动员的匹配过程，这是由来已久的观点。"

政府组织英国体育局委托马修和他的同事们找到赢得多枚奖牌所需的必要条件，以促进国家精英运动以及运动员的发展。研究人员初步确定了 43 个变量，这些变量能准确地预测一个人成为超级运动员的概率。其中一个因素就是教练与运动员之间的关系，所以英国体育局资助进行了专门针对这方面的第二次深度分析。马修和他的同事们招募了 16 位男女超级运动员，他们都曾在重要的锦标赛(比如奥运会)上获得过奖牌。他们还招募了 16 位参加过这样的锦标赛却从未获得过奖牌的运动员。这两个群体在运动项目、年龄以及性别方面都是一致的。之后，科学家们对运动员本人还有他们的父母及教练进行了深入访谈。分析结果后发现，所有的运动员都声称教练给予了其技术上的支持，但是只有超级运动员报告说他们享受到了情感上的支持。马修称，超级运动员的情感支持与尊重的需求得到了满足，而普通和精英运动员却没有。

超级运动员的教练几乎就是他们的代理家长，赞扬运动员的努力、坚定不移地拥护他们、提供积极反馈，并对个人生活感兴趣。马修说："一个骑自行车的人进来，教练可能会说：'嘿，你看起来不太好，我们来喝杯咖啡，聊聊你或许在家里遇到什么困难了吧。'"他们之间有一种超越电子表格、能量输出及图表的联接纽带。"另一方面，有一些精英运动员在他们最需要支持的关键时刻却感觉看不到教练的身影，或觉得导师似乎期待的就是失败。

—— Nuwer, 2016

尽管 DevOps 教练的角色在业界相当新潮，但这并非全新的概念。敏捷教练已是敏捷实施领域由来已久的角色，DevOps 教练就是以敏捷教练为原型的。DevOps 教练将其敏捷知识、DevOps 实践以及结果导向的行为引入项目。DevOps 教练负责一个或多个项目，这取决于项目团队的规模，而目标就是要将其知识与经验传递给项目负责人。通过培训

出自力更生的团队，使其能够交付有效利用 DevOps 的实践，并且进入持续改进的流程，从而让自己从工作中脱离出来是 DevOps 教练的目标。当然，即使在委任的教练卸任后，团队还是可以通过 DevOps 能力中心接受进一步的指导。团队也应该通过分享经验、成功故事以及学到的教训为能力中心提供相应的反馈。

DevOps 教练的角色说明通常包括以下内容：

- DevOps 教练与团队及团队成员紧密合作，开发并执行发展精益及 DevOps 能力的活动，以实现团队及成员技能的持续提升。
- DevOps 教练与团队分享知识与经验，以确保能力中心实施的最佳实践可以用作组织的 DevOps 方法论。
- DevOps 教练的目标是帮助团队发展自己的 DevOps 能力，以快速变得独立，不再需要教练的长期指导。
- DevOps 教练帮助团队选择适当的项目小组工作模式(本章后面部分将详细介绍)，以开发跨职能团队来处理所有的技术需求。
- DevOps 教练促进沟通、协作以及团体动力，并致力于增强团队成员之间的信任。
- DevOps 教练需要提出问题并推动变革，以消除实施障碍。
- DevOps 教练需要成为组织内多个项目间的联系纽带，并利用能力中心，确保组织内 DevOps 实施的一致性及标准化。
- DevOps 教练需要在所指导的团队内进行实验，以了解 DevOps 方式、方法的哪种变体最适合团队，如有增强，又是哪方面的增强。
- DevOps 教练有责任收集成功故事、经验教训以及改进建议并反馈给能力中心，以分享给组织的其他团队。
- DevOps 教练需要具备持续提升其自身能力与技能的态度。

6.2.4　建立能力中心

建立能力中心离不开高级管理层的支持。这是一项重大投资。为了获取支持，能力中心应该先从创业模式开始运作，把自己作为最小可行产品(MVP)来开发，并由来自不同项目的志愿者承担工作任务。这个最

小可行产品需要证实一个假设前提，那就是能力中心通过帮助项目实施 DevOps 实践，能够影响项目的业务成果。需要通过在多个试点项目运行，并且显示出项目的改进以及与每个试点项目交互过程中能力中心自身的改进来证实这一假设。一旦这个假设得以证实，并且证明这是可以重复进行的，而且每次重复的结果都相差不大时，能力中心就可以作为一个商业案例获取支持及资金投入。

一个大型跨国金融服务公司的首次能力中心会议就是周五下午的一场便当午餐会，而我是受邀作为外部演讲嘉宾参加的。三年后的今天，能力中心已经在其全球的虚拟社区拥有超过 3000 名成员。

6.3 方案：发展规模创新文化

> DevOps 最重要的就是其驱动创新的能力。业务的快速发展要求软件交付也要更加快速。目标应该是在几周内引入创新，尽管有些业务每天都需要创新，而不是用几个月的时间去开发并发布一项新的功能。企业面对持续创新已别无选择。他们需要重塑自己以免被淘汰，特别是在软件的交付方式上，软件交付方式已日益成为市场分化及创新的命脉。
>
> —— 杰夫·史密斯，IBM CIO(Smith，2015)

正如 IBM 的首席信息官杰夫·史密斯所说的，最终目标是在整个组织范围内推广创新文化。虽然 DevOps 能够实现这点，但改变文化需要整个组织协调一致的变革。这场变革影响着所有的业务线、部门、程序、项目、团队以及个人处理日常任务的方式，还有实现业务价值的方式。

变革需要改变团队的架构(团队模式将作为一个专门的方案在本章后面部分进行讨论)。变革还要求团队改变方式，将大型、复杂的应用开发与交付工作分解成更小的批次，在更短的时间内进行交付并从业务及用户处获得反馈。甚至在进入开发与交付阶段之前，变革就要求包括

业务线、开发、运维在内的所有团队都确保其交付正确的东西。核心创新要求不断地进行实验，以验证业务正努力向客户交付价值的假设。

重要的是，如何在整个大型分布式组织范围内建立这种文化。要做到这点，所有的利益相关者都需要持续不断地问这样几个问题：

- 如何理解终端用户是谁，以及组织想要满足其什么样的需求？
- 用户需求满足的时候，如何确定组织想要的结果？
- 用户通过所交付的应用或服务与业务交互时，希望他们产生什么样的用户体验？
- 需要运行什么实验才能确保组织正确理解以下问题：
  - 用户是谁？
  - 要满足哪些需求？
  - 期待什么样的结果？
  - 预期的用户体验是怎样的？
- 如何将这些实验分解为最小投资即可交付的最小可行产品或性能，以获得快速反馈？
- 需要用什么指标去衡量实验的成功或失败？
- 随着实验的进行，需要继续交付的下一组目标是什么？
- 如何将工作分解成组件并分配给小型团队，同时还能确保团队遵循更为宏大的交付路线图？
- 团队如何能持续验证其进展？
- 单个团队运行细粒度实验的时候，谁来负责交付物的总体把控？
- 如何衡量进展并创建环境，以及利用一系列的失败实验来测量进展？
- 随着团队的壮大与分散，如何保持开发对用户的关注？
- 如何进行团队与利益相关者的权责分配？
- 如何在以实验为中心的开发模式下确定工作优先级？
- 如何在视觉上传达某个用户体验的最终目标，而这些目标是需要交付至所有组件的？
- 如何创建一种文化，使任何人或团队都能对所开发的产品以及开发的方式提出质疑？

- 如何确保正确提问？
- 基于小批次交付并在短时间内获得反馈的目标，要如何获取反馈并与所有的利益相关者进行有效沟通？
- 如何利用收到的反馈并对已经开发且交付的产品进行调整？

为了实现创新，有两种方法可以引入这个层面的思维与能力，通过实验的视角来审视每一个项目、每一项任务以及每一种流程：

- 精益创业
- 设计思维

这两种方法在第 5 章中都有过介绍。这里再重申一下精益创业的核心主题，更详细的描述请见第 5 章的“精益创业”部分：

- 消除不确定性
- 要更聪明地工作，而不是更努力地工作
- 开发最小可行产品
- 验证学习

设计思维的核心主题这里再重申一下，更详细的描述还是请见第 5 章的“设计思维”部分：

- 关注用户成果
- 不同授权的团队
- 积极重构

这些方法之间的协同与目标一致性是清晰可见的。精益创业为按照创业模式运营提供了所需的核心方法，创业模式指的是组织为了满足定义还不甚明确的业务需求而设计，但业务会就如何解决这个问题设立“未经验证”的假设。因此，创业项目必须进行多次实验去验证其无数的未知：业务问题、建议解决方案、现有市场的盈利能力以及团队交付解决方案的能力。

创业公司是一家致力于解决没有明显解决方案的问题，且无法保证成功的公司。

——内尔·布卢门撒尔，Warby Parker 的创始人兼联席首席执行官

另一方面，设计思维提供了一种方法论以构建、交付并利用这些实验去引领组织开发出满足用户需求与期望的可行方案。

[设计师]在确定真正的问题之前，不会试图寻找解决方案，即使确定了问题，他们也不是先急于解决问题，而是会停下来考虑各种可能的解决方案。只有到那个时候，他们才会最终确定建议方案。这个过程被称作“设计思维”。

——唐纳德·诺曼，《设计心理学》的作者

设计思维旨在理解用户需求并持续交付成果。设计思维通过由以下几点组成的连续循环来实现这一目标：

- 观察——了解用户并理解其需求。
- 思考——产生一种观点，据此制定计划以满足用户需求。
- 行动——创建原型以赋予思想具体的形式并探索交付真正成果的可能性。

设计思维还提供了一种机制，使观察-思考-行动这个循环能够利用大量的团队解决复杂的问题。这个机制由三个因素构成：

- **体验山峰(Hills)。**这是需要实现的三个最重要的用户成果列表。Hills 将用户需求纳入项目的目的及目标。
- **回放。**这包括把团队召集在一起，回顾并反思目前为止所交付的用户成果，并获取“安全”的批评与反馈。
- **天使用户。**这些是作为组织与真实世界接触点的实际用户。他们通过观察-思考-行动循环来提交用户的观点。

产品管理团队

像 IBM 这样的组织已经同时采用精益创业与设计思维在整个组织内发展创新文化。通过在内部创建名为“产品管理”的新角色，IBM 已做到了这点。

IBM 产品管理是 IBM 在市场、用户、产品及服务方面的观点。产品经理决定 IBM 将涉足哪些市场，以及如何通过独特的功能、卓越的用户体验、数字化参与以及生态系统合作在这些市场中脱颖而出。

> 产品经理有权代表企业家利用新的用户体验去开拓新的用户市场。他们负责领导“完整”产品的联合创造，“完整”的产品围绕所有六个普遍经验进行价值交付。
>
> ——IBM 设计思维领域导师，2016

引用中提到的产品经理需要关注的六个普遍经验，能够帮助他们从用户体验的角度审视所有的决策。这六个普遍经验是：

- 发现、尝试及购买——如何获得产品？
- 启用——如何获取价值？
- 每天使用——如何完成工作？
- 管理与升级——如何维持运行？
- 利用与扩展——如何在产品上增加功能？
- 支持——如何脱离困境？

重要的是，要注意针对这些经验的问题是如何措辞的。不找到用户的感觉，是无法读懂的。不站在用户的立场，连正确陈述都不可能。这使得提问成为一种强有力的训练。

像 IBM 那样建立一个产品管理团队，使组织可以提供产品及服务在整个产品生命周期中的全盘所有权，而不是暂时性的。随着时间的推移及市场的变化，所提供的产品及服务也会发生变化。然而，如果组织业务在这些变化中幸存下来，则要通过以客户为基础不断进行产品改进来实现这些变化。改变产品，使其包括新的产品及服务，以满足市场需求。像苹果公司，其产品已从最初的台式机发展到音乐播放器、笔记本电脑，再到如今的移动设备。产品管理团队引领公司根据市场的变化以及用户所期望体验的改变来改进其产品。就苹果公司而言，他们开发了用户自己都想不到的全新市场(想想 iPad 设备)。

可以想象一下，体育用品特许经营商就提供了解决方案来满足如何让球迷与球队互动的业务需求。最终，他们的顾客是球迷们，不是运动员，不是电视网络，当然也不是赞助商。过去，他们只能通过销售门票及提供优惠实现赛场上的互动。现在则演变为销售球衣、帽子以及其他

纪念品。之后出现了广播与电视，通过广播与电视，他们甚至可以与相距很远的球迷联系。这种情况进一步扩展到网页以及移动应用。如今，几乎全部是通过社交媒体来吸引球迷。团队中的产品管理团队掌管着这些演变以及未来的进一步发展。他们负责提供新的产品，使粉丝通过新的用户体验与团队互动。他们负责保持与提高市场份额，并提升团队的业务价值。他们交付的单个产品本质上都是暂时性的。他们的产品，即允许球迷与球队互动这种情形却是永久的，至少是只要球队还存在就可以。

总结来说，设计思维与技术无关。事实上，它与情境无关。可以利用设计思维的原理与方法规划假期或者设计家装。球队可以利用设计思维为下一场比赛发展新的选手。可以利用设计思维来创作一本书(我就是如此)。设计思维提供了一系列的思考工具，比如，关于持续创新以及持续改进的工具。设计思维为组织提供了实施 DevOps 文化所需的核心思想，不论组织的规模大小。

6.4　方案：发展持续改进文化

改进大学橄榄球赛事

如果你问曼尼·迪亚兹这样一位长期防守的教练，体育运动将走向何方，他定会提到人们记忆中激动人心的赛事里那个标志性的时刻：奥本队在 2013 年的钢铁碗(Iron Bowl)比赛中战胜了强大的阿拉巴马队。

打击迪亚兹的不是注定老虎队胜利的“六连踢”触地得分，也不是萌发出的萨班与对手间的竞争。而是奥本队四分卫尼克·马歇尔的一场比赛，上场时比分为零，却比克里斯·戴维斯 109 码比赛冠军提早 31 秒结束比赛。

这位任职 17 年的资深教练说道：“这是大学橄榄球比赛最为重要的事情。”上个赛季最重要的战术就是把比分定格在 28 分的触地得分。“作为标准赛区阅读选项比赛，所说的比赛由此开启，而且奥本队打了整个

赛季。就是在这个时候，Gus Malzahn 的进攻战术将橄榄球带向了更广阔的未来。

比赛的方式都是标准化的，从始至终保持着熟悉的程度，甚至连美式橄榄球联盟的球迷们都是一样。

进攻线开始封锁内区比赛。他们要一直阻挡另一名球员特雷·梅森的奔跑，直到峰线队员知道为止。后卫杰伊·普罗什在未被封锁的防守端周围绕行，以为马歇尔提供一处边线领先位置，免得四分卫被持续阅读。

在边线为四分卫额外安排一个阻卫给大学防御带来足够多的麻烦，但这是阿拉巴马队早已经知道会出现的问题。马歇尔看到防守端还在里面，给了他一个信号，让他控制住球并跟随普什(Prosch)的边线防守。

这就是事情变得有趣的地方。马歇尔又做了一次阅读，以确定他的 X 接球手，萨米·科茨，是否被盯防。由于游卫和角卫都跑过来阻止四分卫，他艰难地拉起并猛地传给科茨。

触地得分。钢铁碗比赛结束。大学橄榄球由此改变。

—— Boyd，2014

美式橄榄球的三重选择是一种战术，四分卫在比赛中可以进行选择。有“教练”负责指挥，但最终还是由四分卫做出选择：是自己带球、把球传给后卫，还是投给已经移位进来的缝卫(Davie，2015)。当然，还可以像上面描述的 2014 钢铁碗(Iron Bowl)比赛中那样，即兴发挥把球传给自由接球手，然后一起跑阵。授权利益相关者对其可能在应用交付流水线中识别出的任何改进机会采取行动才能发展出持续改进的文化。要授权利益相关者进行持续改进，以使其不会盲目遵循方案要求，而是有权做出改变，至少是有权建议改变，以实现方案的改进。这就是在日本称为“Kaizen”(持续改进)的丰田生产系统。

实现持续改进(Kaizen)

Kaizen 的理念是丰田的核心价值观之一。意味着“持续改进”。任何流程都不可能一直宣称完美，但总是可以改进的。

实践中的持续改进意味着组织各个部分的所有团队成员都要持续寻找改进操作的方法，而且公司中的各级人员都支持流程的改进。

持续改进还要求设立明确的目标及指标。关注应该做什么，而不是能够做什么，这才是非常积极的态度。

——丰田生产系统

6.4.1　开发实施路线图

本书前面的章节中已经讨论过实施路线图的概念。附件 A 中也附上了实施路线图的样例。在大型组织中推广 DevOps 实施也同样需要一份这样的路线图。如果是单独的一个小型团队实施 DevOps，团队可以自主决定实施的实践内容和实施顺序，并且让所有的利益相关者都参与进来。然而，当多个团队实施 DevOps，而且是以并行却不一定完全同步的方式进行时，让每个团队都遵循主路线图来开展工作就显得尤为重要，这样才能保证实施按照既定的架构有组织地进行，避免团队在实施实践时与采用相反方式实施不同实践的其他团队相互影响。

实施路线图通常是在企业、业务线或部门层级，也有极少数情况下是在项目层级进行开发的。这些层级可能在名称、规模上各不相同，这取决于所讨论的组织，以及组织内部职责划分的界线：是整个组织共享一个 IT 团队，还是每个单元或部门拥有自己的 IT 团队？因此，路线图旨在提供一份蓝图，供单位或部门内的所有项目使用，以实现推广实施的目的。目标则是要避免不同团队按各自不同的喜好来实施 DevOps，确保每个团队在实施 DevOps 时都能遵循主路线图，而不是去匹配及增强自身的具体需求。

此外，实施路线图应该是一个动态文档，本着真正的持续改进精神，基于团队实施过程中的反馈，根据市场、业务、技术以及团队的变化进行持续改进。市场情况可能变化，这就要求业务目标以及组织的关注点也要随之改变，如下所示：

- 故障可能导致关注的焦点从速度转移到质量。
- 竞争对手开发出真正创新性的产品，给组织造成威胁，可能就需要变更正在开发的产品。

技术的改变能够引发路线图实施方式的升级需求，尤其是在如何实现自动化方面，如下所示：

- 从前没有或是不够成熟的新技术现在已经可以利用。近年来，容器的日渐成熟就属于这种情况
- 旧的技术可能被淘汰。比如，软件供应商对正在使用的软件版本停止支持或开发，从而导致软件失效
- 组织可能会决定转移平台或技术供应商，从而导致 DevOps 平台发生显著变化

最后，团队的变化也可能导致实施路线图的变更需求，如下所示：

- 团队模式变成新的矩阵结构
- 退休、流失或裁员(resource action)导致团队成员离开
- 新聘人员加入团队
- 更换外包服务供应商

因此，实施路线图需要有专人管理。负责管理的专人(们)需要遵循明确定义的变更管理流程对路线图进行更新。路线图的更新来源有两类：

- 来自实施路线图的团队的反馈
- 重新运行价值流图实践以获取变化

6.4.2 持续开发与价值流图

谈及利用价值流图来驱动持续改进的文化，定期重新绘制价值流图实践意义重大，不论是正式、非正式的工作坊还是专门的训练。这种实践能够不断识别出交付流水线中现存的瓶颈问题与浪费。然而，无论是从时间还是从资源投入方面而言，运行正式的实践都需要很高的成本。即使是非正式会议，也要求执行的人具有相应的专业知识。专业知识的开发是一项高附加值的投资。DevOps 能力中心是一个好地方，可以帮助人们掌握正式工作坊以及专门会议的价值流图技巧。

这就是说，对所有运维人员进行价值流图基本概念的教育，这本身就能带来长期的收益。人们在任何流程中遇到浪费源时都能识别出来，这种价值是无法衡量的。最终的目标就是持续改进。如前所述，真正实现持续改进的方法就是授权每一个运维人员去识别并采取行动，解决其

自身工作流程中的瓶颈问题与浪费。这里的关键是能够识别出浪费产生的根本原因。了解价值流图的运行方法对此很有帮助。

让我们再从第 2 章重温一下如何在交付流水线的价值流图中查找浪费源。要识别出浪费源，需要审视以下方面：

- 构建物
- 利益相关者
- 环境
- 流程

这四方面中的任何一个方面都可能存在低效及浪费。我们再来重温一下第 2 章中所列的这些方面存在的浪费源：

- 流程低效
- 流程手动
- 构建物形式错误
- 利益相关者之间的交接低效
- 利益相关者无法在流程中执行任务
- 利益相关者无法根据需要访问构建物
- 利益相关者在不必要的任务上耗费时间
- 利益相关者创建不必要的构建物
- 流程被过度监管

所以，要如何来发展持续改进的文化呢？通过在每一个运维人员心中建立起一种心态，使其能够随时检查所使用的构建物、所交互的利益相关者、所处的工作环境以及所执行的流程，还要检视自己，看是否存在浪费，才能发展出持续改进的文化。以下是需要提出的一些问题：

- 这个构建物是必需的吗？它能为开发交付流程或者终端用户带来任何附加值吗？
- 我所接收的构建物，其格式是否不需要再进行不必要的转换或工作便可以直接使用？对于那些从我这里接收构建物的利益相关者而言，也是这样吗？
- 我正在执行的操作能给构建物带来增值吗？一旦我改变了其状态，谁还会使用？我给那个利益相关者带去附加价值了吗？

- 我是否采用了最有效的方法来改变构建物的状态？
- 对于正在处理的构建物，我是否对整条交付流水线具有可见性？
- 我与其他利益相关者的交互是否必要？他们为我的工作带来增值了吗？我能给他们的工作带去附加价值吗？
- 我们交互的时候交换了什么构建物，原因是什么？
- 我们执行了什么流程，我们是否都需要执行这样的流程？
- 我与利益相关者的交互是否高效，还是我们其中一方不得不等待对方或第三方利益相关者做出行动？
- 我是在合适的环境中工作吗？
- 环境是根据我的需要配置的吗，还是我需要再花时间和精力进行重新配置？
- 我能否在需要的时候、在所需环境中有效执行所需流程？
- 我是否需要等待长得不合理的时间才能获得我想要的配置好的环境？
- 在环境内外取得构建物与工具的流程是否有效？
- 我是否正在执行适当的开发、交付流程以实现流程增值并为终端用户交付价值？
- 我是否正以最有效的方式执行流程？
- 如果我需要与其他利益相关者协作来执行全部流程，我对他们正在从事的工作是否具有可见性？

如果应用交付流水线中的每一位利益相关者都能在日常工作中不断地提出这些问题，并且有权采取行动去减少识别出的浪费，那就是正在朝着实现持续改进的方向前进了。当然，要在整个组织范围内实现真正的持续改进，需要将这种文化渗透到整个组织中，从运维人员到高层管理者的各个层级。

6.5 方案：DevOps 团队模型

如何为 DevOps 开发稳定的团队模型呢？组织需要一个由跨职能利

益相关者(如图 6-1 所示)所组成的团队，这些跨职能利益相关者具有以下特征：

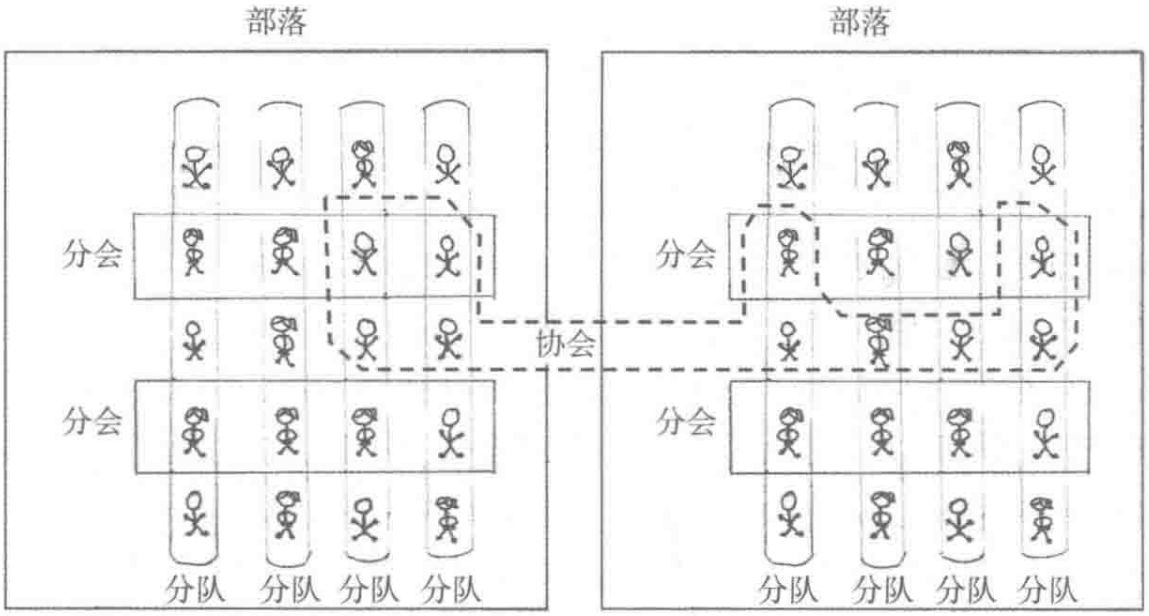


图 6-1　分队、部落、分会、协会(Shreya Sharma 绘制)

- 通过专门的团队成员或者资源共享，工作所需的跨职能技能都可以为团队所用。
- 没有组织仓筒来破坏其合作能力，他们能够进行沟通与协作。
- 他们对彼此的工作具有可见性。
- 他们能够在集成工具链间来回传递构建物。
- 他们的规模足够小，能够实现操作所需的敏捷与独立。
- 通过大量复制小团队，他们可以扩展为大型分布式团队。
- 这些团队彼此之间能够交流与协作，因此需要集成工具及标准化流程。
- 有综合治理模型来管理团队规模，同时也允许各个团队自主敏捷运行。
- 团队中的利益相关者加入矩阵式团队，仍可以保留现有的汇报链。
- 掌握专业技能的利益相关者可以将时间分配给多个团队。比如，企业架构师、安全专家以及网络专家就是具有此类技能的利益相关者。

- 掌握相似技能的利益相关者可以进行合作，并在团队间相互分享经验教训及最佳实践，有效覆盖自己的团体。想象一下：Heat 模式设计师团体。
- 应用与系统架构支持各个独立团队的组件开发与交付，这些组件还将进行持续集成与测试。
- 这种团队模式可以复制以组建更大型的团队，也使地域分布式的团队得以存在。

音乐流媒体公司声田(Spotify)就创建了这样一个完全相同的团队模型，如图 6-1 所示。Henrik Kniberg 和 Anders Ivarsson(Ivarsson，2012)在其文章中所介绍的这款模型，如今已被 IBM 这样的大型分布式组织修订后应用于其开发团队。虽然这个模型最初被当作一种方法，用以为敏捷开发流程的实施开发可扩展团队，而今，声田及其他公司将其用于 DevOps 的实施。该模型具有以下核心组成部分：

- **分队(squad)**。分队是团队中的最小单位。分队的运作很像迷你型创业公司，他们与利益相关者直接进行沟通，致力于整个应用中某一个功能单元的开发与交付。分队成员长期负责这项功能，通常，一个分队有一个“用户故事”。“用户故事”是一个敏捷方法论的术语，用来从如何使用的角度描述功能需求。

  用美式橄榄球做个比喻，用户故事就是团队为了将球向前移动而采取的一种特殊战术。分队就是战术制定者带到场上来执行战术的队员。比如，球队如果打算进攻，而不是弃踢的话，可能会在第四次进攻时召集专门执行特殊战术的分队上场。
- **部落(tribe)**。部落是工作在应用相关功能领域的一组分队的集合。部落可能负责整个应用，或“史诗”或“体验山峰”。“史诗”是一个敏捷方法论的术语，用来描述共同获取更强大功能的一组用户故事。通常，最高级别的史诗会映射到设计思维的体验山峰。

  部落内部的分队与其他分队会相互依赖。分队之间还可能有跨部落的依赖关系。部落不应该制造障碍或者仓筒，这会阻止多个部落之间开放、自由的沟通与协作。

还是用美式橄榄球做比喻，史诗就是球队努力把球传到达阵区的一组战术。他们向前传球时可能还不知道要采用哪一种战术，但是他们有全面的计划、现有比分、可以上场的球员、已经掌握的战术手册，还有时钟上剩下的时间。进攻组就是团队中的一个部落，专业的团队都是作为第三次进攻的部落，防守组也是一样。

- **分会(chapter)**。分会是同一个部落内，在不同分队的相同能力领域工作的运维人员所组成的团队。分会成员间通过分享各自分队的最佳实践及经验教训，彼此进行沟通与协作。所以，部落里所有的基础设施分会定期会晤，一起讨论基础设施即服务(IaaS)的需求、问题以及部落交付的应用所要面对的挑战。

  仍然以美式橄榄球为例，每个球队都有跑卫、接球手以及线卫分会。进攻线锋与防守线锋也都是分会。他们都有专门的教练，彼此之间也会互相帮助以提升各自的专业技能。

- **协会(guild)**。协会本质上也是一个分会，只是协会是跨越部落界限的。协会也包括对其能力领域感兴趣的任何一个利益相关者。比如，测试协会、安全协会、Docker 协会等。

  如果美式橄榄球中也存在这种协会，那应该是国家橄榄球联盟中所有球队的跑卫、接球手以及线卫跨越团队边界所组成的一个特殊利益群体。弃踢回攻专家应该很受接球手协会的欢迎。进攻线锋与防守线锋应该都欢迎彼此加入自己的协会。四分卫协会则必然会吸引更多的人。

6.6　方案：工具与流程标准化

体操评分标准化

80 多年来，体操比赛一直使用的是 1～10 分的评分标准。满分 10 分就是个神话，也是每一个运动员的终极目标。然而，自科马内奇在 1976 年的奥运会上缔造 10 分神话以后，评委们评分时开始变得慷慨起来，

而“10 分”也就失去了其重要意义。

比如，在 1984 年的洛杉矶奥运会上，就出现了 44 次完美的 10 分。分数膨胀导致更加难以区分动作是良好(表现不错，难度系数高)还是优秀(表现完美，难度系数前所未有的高)。

20 世纪 90 年代，国际体操联合会(International Federation of Gymnastics，FIG)认为评出了太多的 10 分，并决定修正整套体系。基于难度水平以及成套动作完成情况赋予动作起评分。出现任何失误，都会从动作起评分中扣减分数，由此，取得 10 分成绩几乎是不可能的。

整个 90 年代都是用的这种评分体系。然而，在 2004 年的雅典奥运会上出现了争议。美国选手保罗·哈姆(Paul Hamm)以 0.049 分的微弱差距在男子个人全能比赛项目中赢得金牌。之后，来自韩国的铜牌得主梁泰荣(Yang Tae Young)提出抗议，声明称其双杠比赛的最终得分不准确，原因是起评分计算错误。如果动作起评分准确计算，那么金牌得主就应该是梁泰荣，而不是哈姆。

巨大的争议随之而来。三名裁判被解职，哈姆先是被要求放弃奖牌，之后又要求共享，经过长期的法庭审理流程，最终又正式被认定为获胜者。

在这场争议的推动下，国际体操联合会(FIG)于 2005 年修订了评分标准，以全新的方式区分运动员动作的差别。全新的更加复杂的评分程序根据起始难度以及动作完成情况来分析运动员的表现，由此完美 10 分成为了历史。在目前的体系中，好的成绩通常在 16 点几分，和完美 10 分不完全相同，但可能对运动员而言更加公平与精确。

—— iSport.com

如前所述，人与流程需要能够在团队与项目之间、分队与部落之间互换。也讨论了如何实现跨团队及项目的可视性与端到端可追溯性。为了能够准确地测量正确指标，项目或团队间的衡量指标也要进行标准化。就如同对奥运会上两名优秀体操运动员进行评分一样，如果衡量标准不一致，是无法比较两个团队的业绩改进程度的。

要实现人员可互换、可视性以及可追溯性所有这些目标，就需要一套标准化的集成工具。将所有应用交付线完全标准化成一个工具集是不

合理的。如前所述，组织并不是一成不变的。不同的业务单元、部门，不同的项目需要使用不同的技术栈，不管是因为他们需要在这些堆栈上进行并启动标准化，还是因为其应用需要这些堆栈。不可能将所有的应用都转换成 Node.js 或全部部署到大型机。[①] 然而，谈及实现此处所描述的这些目标时，工具的随意扩展则会引起混乱。这就需要一个中间平台，允许团队保留技术堆栈，同时又能实现标准化、可视化以及可追溯性的目标。中间平台的作用就是将工具限定为最小集成工具链，每个技术堆栈只提供 1-2(最多 2 个)工具链。实现集成是核心的目标。也就是说，供应商或工具提供者需要交付开箱即用的预集成的工具。最后，组织还需要准备的是维护工具之间自主构建点集成所需的费用。

DevOps 集成平台的标准化

有些工具是以商品化的形式为运维人员所用的，其对于其他从业人员的影响微乎其微，因此，不需要进行标准化或者集成。以集成开发环境(IDE)为例。只要分队中的所有开发人员都使用相同的集成开发环境，那么其所交互的其他分队是否使用另外的集成开发环境关系不大。另一方面，随着代码库成为开发人员在分队及部落内或者跨分队/部落进行同业间代码共享时的协作工具，代码库的标准化势在必行。开发者协会应该确定代码库的标准，只要这个标准能为所有的集成开发环境所用，就可以实现其目的。

因此，组织需要确定哪些工具及平台需要标准化，哪些不需要。对于那些需要标准化的工具与平台，通常会为每个技术堆栈确定 1-2 项标准(总是会有例外的，结构复杂的大型组织也会有 2 项以上的标准)。确定不需要标准化的工具应该集成到交付流水线的其余部分。构建物的跨交付流水线流动，工作的可见性以及端到端的可追溯性是任何工具得以进行标准化的前提条件。

通过为其 Bluemix PaaS(Brealy, 2016)开发出开放工具链，IBM 同时满足了标准化与集成的需要。IBM 的开放工具链如图 6-2 所示。为了让

① 我那些来自大型机世界的朋友们会强烈反对这种观点，特别是关于 Zlinux 与 OpenStack 在大型机上的可用性方面。

第三方供应商能够将其工具添加进工具链，IBM 还开发了工具链软件开发工具包(Software Development Kit，SDK)。软件开发工具包允许任何工具代理作为工具目录中的一项服务在 Bluemix PaaS 中进行构建与使用，工具目录是可以添加到应用开发团队构建的工具链当中的。在 IBM Bluemix 上构建工具链的架构如图 6-3 所示。

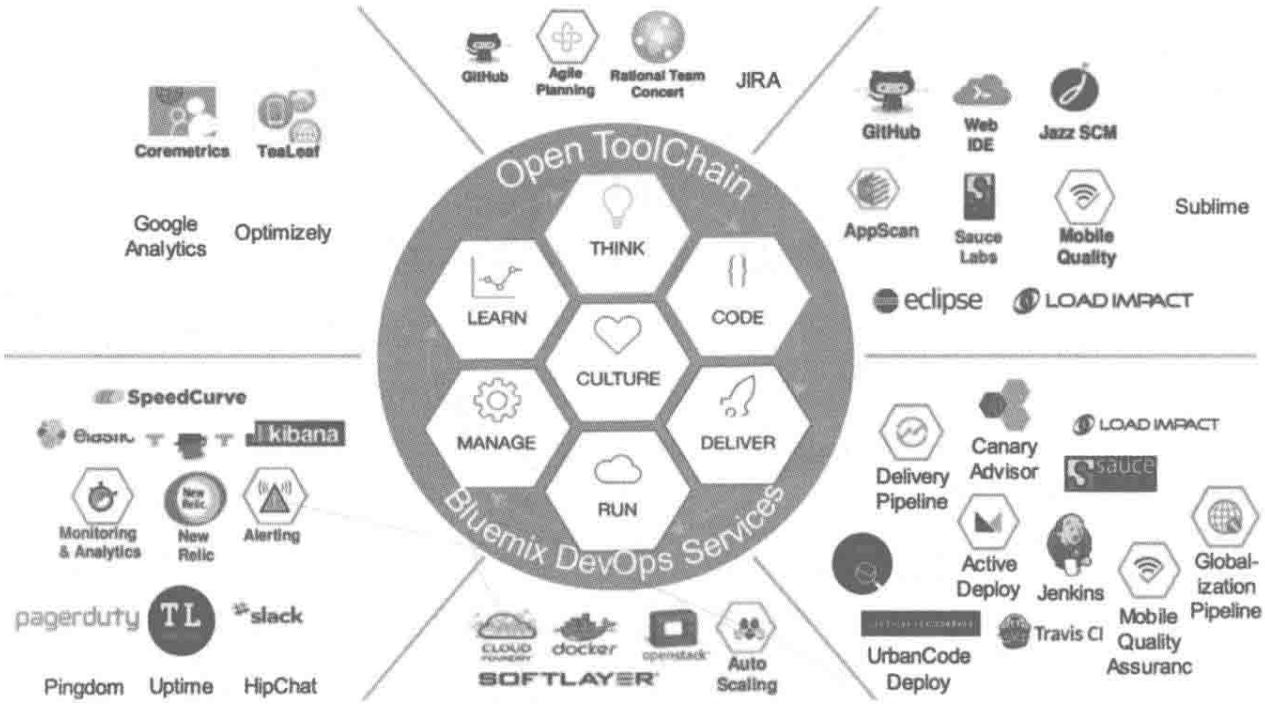


图 6-2　IBM 开放工具链

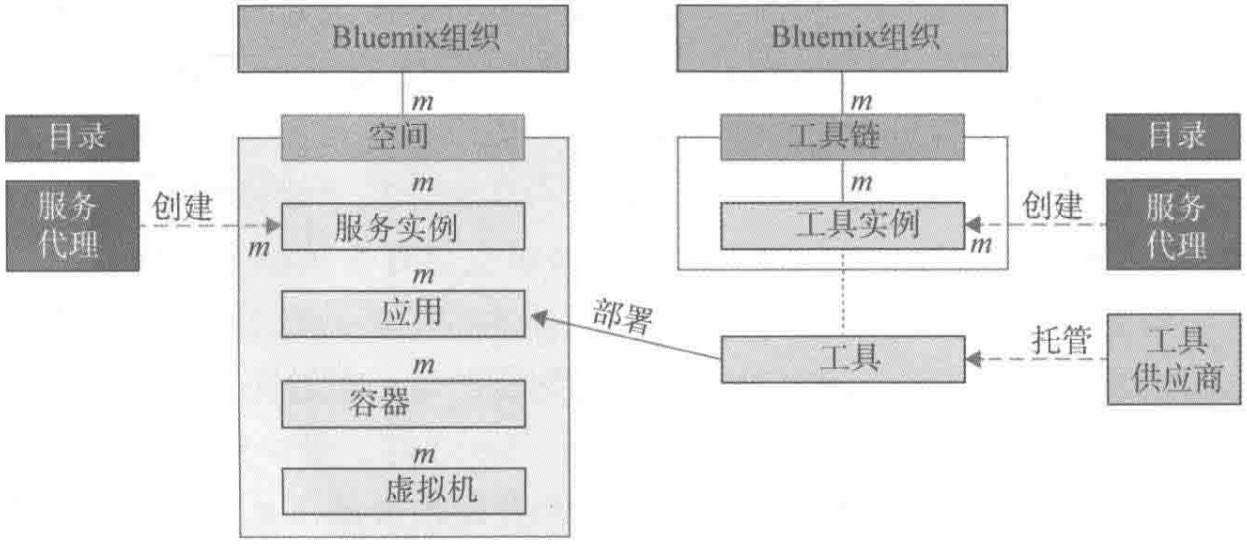


图 6-3　在 IBM Bluemix 上构建 DevOps 工具链使用服务

平台一直都是需要标准化的。尽管团队会自行选择适合的平台，但

平台还是要能够被执行团队接受，并且可以纳入 IT 组织的治理模型中。否则，就会出现影子 IT。

正如第 4 章所讨论的内容，对于云平台，利用 OpenStack、Cloud Foundry 或者 Docker Container 这类的技术可以实现平台自动化，而不会限制团队所需的敏捷性，从而与之前的选择保持一致。比如，一个项目可能选择利用 VMware vCenter 在虚拟基础设施上进行部署；另一个项目则可能会选择利用亚马逊公有云；其他项目可能更倾向于 IBM Blue Box 中托管的物理机环境 OpenStack IaaS。利用 OpenStack Heat 将基础设施定义为标准化模板或模型，开发人员便可以提取所有的这些解决方案。

6.7　方案：DevOps 的安全性考虑

守门员的重要性

当听到球队即将签下 20 个进球记录的前锋或是连最顽固的防守都能攻破的创造性天才这样的消息时，你总会激动不已。与实际经验不足的新中卫或者经验丰富的门将签约时却感觉不到如此兴奋。

在球迷眼中，漂亮的进攻球才是追求的目标。回顾历史，最吸引球迷的始终都是巴西队的自由流动天赋以及巴塞罗那队的跑位传球。意大利队稳定、可靠的风格很难激发热情。

然而，人们只需要看看意大利队在俱乐部比赛以及国际赛事中所获得的那些奖杯，就会不禁怀疑或许他们的风格中有什么能够促进成功的东西。他们是四次世界杯冠军，仅次于巴西和德国，尽管欧洲赛场上都是像国际米兰和尤文图斯这样的强劲对手。

在一期《金融时报》专栏中，西蒙·库珀引用了切尔西赛事总监麦克·福德一段有趣的话，他说的是："相比于进球与最终排名的关系，不失球(Clean sheet[①])与最终的综合排名关联性更强"。

—— DW 运动，2012

① Clean sheet：球队在比赛中没有被对方进球的情况。

大型组织引入 DevOps，尤其是像持续交付以及快速交付这类概念时，其中一个最大的阻力是来自于安全领域的利益相关者。安全团队中的高管到运维人员，都表达了他们的担忧，担心这会影响到他们维护组织预期安全状态的能力。

安全团队，就像是足球比赛的门将，没能受到应用交付团队足够的重视。事实上，大多数应用交付团队都把他们视为创新及变更速度的阻碍。应用交付团队的动机是快速交付新的创新能力。安全团队的使命则是确保新的系统与功能是安全的，且安全功能是强劲的。这些目标似乎是冲突的。

然而，就业务成果而言，这些目标其实并不矛盾。提高时间价值、交付创新业务方案、创造高质量产品、增加市场份额，所有的这些业务成果都是既需要应用交付团队持续交付能力，又需要安全团队不断保证安全。团队有着相似的目标，只不过实现目标的方式不同而已。

DevOps 引进了一种方法，将应用交付组织所交付的能力按照小批次进行持续交付与持续测试。安全团队可以利用这种交付方式来降低安全风险。通过持续保护这些小批次发布的功能，他们能够在生命周期的早期识别出安全漏洞并减少其影响。

安全团队仔细检查并确保 DevOps 引入的应用交付流程安全时，他们还必须与开发及运维团队协作以确保 DevOps 交付流水线及流程的安全。DevOps 并非要以牺牲安全为代价来实现速度最大化，而是旨在利用最短周期时间，在交付小批次能力过程中提供快速反馈。快速交付及持续反馈周期有助于增强安全性。将安全团队纳入 DevOps 生命周期，能够确保所交付应用与系统的安全保障成为整个交付生命周期中一项持续进行的流程，而非只是在交付周期后期才加入的一个步骤。

正如精益生产彻底改变了工厂自动化与产品交付，DevOps 改变了应用的交付。工厂自动化的出现要求制定相应的实践，来确保产品交付装配线的安全。输入组件、生产线工人、自动化专家、装配流程以及其他要素都要确保安全并通过验证。同样，安全专员也需要与应用交付团队合作，来确保对应用交付人员自身、流程以及自动化工具的安全性以及有效性，要成为 DevOps 实施不可分割的一部分(Elder，2014)。

6.7.1　管理安全相关风险

企业担心他们使用的所有软件都存在遗留漏洞的风险。这些风险包括：

- 供应链相关漏洞
- 内部恶意攻击
- 源代码的丢失或损坏
- 开发流程破坏
- 开发项目中的错误与误解
- 设计、编码及集成缺陷

包括瀑布式项目、敏捷项目或广泛实施 DevOps 方法的项目在内，任何形式的软件开发生命周期或方法论都可能存在这些风险。基于 DevOps 项目的优化本质以及先进的自动化特点，必须在整个交付生命周期内，对这些风险相关的事件与条件进行检测并作出反应。

DevOps 实施需要对每一个风险领域进行专门思考，特别是在整个组织范围内推广 DevOps 实施的时候。

1. 供应链相关漏洞

如第 4 章内容所述，任何软件项目只要包含了在项目之外创建的软件组件，就可以称为软件开发供应链。这些组件可能由公司内、外部的供应商创建，也可能由拥有或交付软件项目的组织所创建。软件供应链中那些软件的安全特性会对项目中创建的那些软件的安全性产生深远影响。

在传统的开发项目中(包括瀑布项目及迭代项目)，通常由开发团队来评估供应链软件的安全特性。评估包括审查组件文件，寻找许可及支持性审批文件，还有运行安全扫描。

通过在整个项目周期中对设计、代码以及集成进行实时决策，DevOps 开发团队获得了最大的灵活性。因此，开发团队可能会选择功能更强大、更易于集成，而对组件的安全性及稳定性要求不高的供应链组件。

要控制这种局限性，就必须通过实施持续测试在软件交付流程中构

建严格的质量检查体系。这种实践体系中包括了交付周期中每个阶段所进行的测试。测试需要涵盖组件的安全测试以及对每一个所交付组件手动或自动进行的代码审查。DevOps 提倡在较短的周期时间内交付每个组件的小批次变更，结果是，组件的小批次变更都能在交付时得以进行持续测试。这种方法减少了相关的风险，并能更快识别出安全漏洞。

2. 内部恶意攻击

尽管确切的数字尚不清楚，但业界有证据显示，过去几年间，网络犯罪案件中内部人员所为的占了相当大的比例。这些攻击可以造成源代码丢失、源代码损坏或者开发流程破坏。这些攻击可能源于内部人员直接的恶意所为，也可能是由开发环境内使用的网络、工作站或服务器上的恶意软件感染所致。

在传统开发环境中，要减少这类攻击发生的可能性，通常需要确保开发基础设施的安全性并且将异常检测及告警仪表盘化。在先进的、自动化 DevOps 平台上，仪表盘化检测及告警可以将源代码丢失、源代码损坏、恶意软件部署或者开发流程破坏的难度增加。通过将白盒与黑盒安全测试纳入在交付周期中执行的测试任务清单，可以相应地减少这种局限性。在每一次迭代中运行这些安全测试，可以检测出所有这类恶意攻击。

虚拟化的软件定义基础设施使其配置得以受变更管理并能够进行审核。不断打破并重建部分 DevOps 平台的能力有助于减少恶意软件在这些部分的持续出现。

3. 开发项目中的失误与错误

传统开发项目，不管是瀑布式的还是迭代式的，通常都是由项目管理、项目跟踪工具及系统提供支持，这些工具及系统能提供精心编排的工作流及任务完成检查点。特别是在完成的软件产品发布之前，会有一个严格的内容宽泛的项目审查，其中就包括检查主要任务或里程碑的完成证据。

应用敏捷或 DevOps 方法的项目会在更短的交付周期内，更加频繁地交付软件产品的小型组件或小批次变更。虽然不可能将每组变更都交

付给顾客或用户，但是缩短周期时间能够减少严格项目审查的次数以及仔细检查主要任务还有里程碑完成情况的次数。

这些捷径可能会使开发项目的失误以及错误进入到未经检查的开发周期。然而，DevOps 项目的目标就是将软件项目的小型组件交付给测试团队再到项目审查流程，以从一开始就能减少项目失误与错误发生。通过更加频繁地交付小批次软件组件变更来发现失误，能够降低总体的风险。

4. 设计、编码及集成缺陷

如果供应链的安全得到了充分保证，内部人员的恶意攻击以及项目失误得到了控制，开发项目中残留的最大风险就是引入缺陷了，而这些缺陷是可以在软件部署后被利用的。整个开发项目的设计、编码以及集成流程中都可能出现这些缺陷。利用以下策略中的任意一个，都可以减少软件缺陷出现的可能性：

- 执行迭代测试与补救策略
- 实现安全的设计策略

迭代测试与补救策略可以用于成本与进度不受限制，并且使用全面测试工具的小型项目。 安全设计开发策略随着 DevOps 项目的出现日益成熟。以 IBM 安全工程架构(IBM Secure Engineering Framework)(Whitemore，2012)为代表的安全设计策略可应用于 DevOps。

6.7.2　解决 DevOps 流程与平台的安全问题

DevOps 实施的自动化类似于生产系统从劳动密集型到自动化流水线的转变。第 3 章中已对软件供应链与生产供应链进行了详细的比较。生产流程不再只是在适当的时间以及生产车间的适当位置交付库存，现已发展为定位生产线步骤，进而转为可控的、精确的、高速的机器人系统，而不再依靠人的双手去安装、连接、移动以及装配元件。

1. 供应链中的漏洞举例

生产环境中存在着多家供应商向供应链交付组件的情况，由此便产生了漏洞。这些供应商可能有意或无意地供应劣质或有缺陷的组件。在

传统的供应链中，人们发现供应的组件出现问题时会举起一面小旗来警告生产线。这种手动流程降低了风险。

在广泛应用自动化的精益或 DevOps 项目中，自动化流程能否检测到供应的问题，取决于自动化元件是否具有验证所输入供应物的质量检测功能。

这两种情况下的解决方案都是要管理并核实供应链的来料。对于精益或 DevOps 项目，这其中涉及增加测试门禁以取代过去由人来执行的监测功能。例如，可以执行一系列的自动化测试来验证刚刚收到的新版本开源工具包是否能在应用可接受的范围内运行。

2. 防止内部恶意攻击举例

生产过程中，生产线工人可以故意留下连接错误的配件，可以不进行连接，可以在装配线插入异物，也可以蓄意破坏其他人的工作，之后再把设备打扫干净以掩盖其行径。

在 DevOps 应用交付环境中，自动化取代了个体运维人员。然而，自动化工具程序(比如自动化工具 Chef “大厨”的秘诀在于 recipes“菜谱”或者云计算提供的 Heat 模式)也可能会出现破坏自动化的恶意行为，部署恶意软件，蓄意破坏配置，或篡改系统。

这两种情况下，要让工人之间相互制约，自动化环境下，则涉及自动化代码创建中的多重制衡。通过控制范围、审核以及在发布前进行多重签出与审批的要求来防止内部攻击。可以创建类似的门禁作为测试用例嵌入到自动化中。考虑到自动化本身的创建也可能会成为攻击点。在变更管理控制下的软件定义的基础设施中，每一个软件定义的元件都是版本化的，使用这样的基础设施有助于减少攻击点。

3. 源代码丢失或损坏举例

如果将软件开发比作生产制造，那么源代码不是装配生产线上制造产品所用的原材料，就是工人制造产品所要遵循的设计图和装配计划。不论是销毁或删除代码或计划，还是篡改代码或计划，都将影响装配生产线上最终制造出来的产品。

在 DevOps 应用交付环境下，源代码处理不当(编译器构建二进制组

件所使用的原材料或源代码)可能导致代码篡改或损坏。篡改用于自动化开发(机器人运行或部署自动化)的设计材料或指令可能导致类似不符合预期的结果。

装配生产线的解决方案是对原材料、方案与设计进行严格控制与审核，同时还要定期进行质量测试，确保装配出的产品符合设计要求，还要确保原材料未经篡改。在 DevOps 软件交付模式下，更多是利用自动化测试来验证装配出的产品是否与设计规格相符。此外，监控、审核、强制访问设计材料、源代码以及自动化源代码(代码装配及部署的机器人行为)，可以确保其不存在任何的安全缺陷或漏洞。

4. 开发流程破坏举例

以装配生产线为例，生产线工人可能不会依照设计好的装配生产线流程或程序来工作。装配生产线上的每个工人都必须遵循标准作业程序(Standard Operating Procedure，SOP)。不按照这些程序作业，就会生产出残次品。在应用交付环境下，从事软件编码、集成、测试、部署以及类似工作的运维人员也要遵循标准作业流程(SOP)。不按照这些程序工作，就会导致软件缺陷。对于自动化框架而言，错误的自动化可能会引发失误。

2012 年，国际金融服务公司骑士资本集团(Knight Capital Group)的一次交易失误就导致公司损失了 4 亿 4000 万美元。原因可以追溯至一个部署工程师未正确遵循部署标准作业流程(SOP)(Popper，2012)。之所以未能及时发现失误，是因为公司没有通过任何形式(自动化或人工)进行足够的质量检查去验证部署是否依照适当的流程进行。

要降低装配生产线的风险，必须对生产线工人就工艺流程与作业程序进行足够的培训，还要进行监督与质量检查，以确保工人始终遵循流程作业。在应用交付环境下，利用流程自动化及监控工具，流程监督与质量检查可以自动化进行。

5. 开发流程失误及错误举例

在生产环境中，人们在工作时会犯错误或出现失误。手动作业更容易出错。生产线的工作以及为生产线工人设计流程的人都可能造成失误。

在应用交付流水线环境下，失误有各种形式：代码或脚本编写错误、文档错误、数据输入错误以及其他类似情形。

要降低装配生产线的风险，需要实施监督与质量控制，还需要创建能够防止失误或能够尽早发现错误的强大系统。在应用交付系统中，随着时间的推移，也已经开发出减少失误的方法。可以将测试嵌入到代码中以验证代码，同时验证应用中的代码组件是否正确使用。这又回到了第 5 章中关于抗脆弱系统的讨论，问题的解决方案就在这里。假定失误会发生，并准备自动化构建以降低影响并从失误中恢复。如前所述，一些组织已完全采用抗脆弱系统，任何一个服务器实例出现了错误，都会自动替换为新的实例并卸载刚刚发生错误的实例。

6. 设计、编码及集成缺陷

在生产过程中，设计师(建筑师及机械工程师)、工艺工程师(工业工程师及团队负责人)以及装配工(机械师及装配工)之间的交接与沟通不畅，会导致产品不合格，生产过程中还要进行改变，弄弯部件才能装配，替换其他可用部件以及其他情况发生。对企业外部外包商及供应商的依赖更加剧了这种挑战。沟通失误的一个不好的例子是阿波罗(Apollo)13 号事故，案例中，调整探月火箭子系统的电压需求时没有与外包商进行沟通，导致在登月飞行期间险些发生灾难(Christofes，2014)。

在应用交付中，团队将代码移交给开发代码的其他团队或负责集成、测试、构建及部署的团队时，就会出现这种交接失误。需要多个供应商跨越公司界限完成这些交接时，这种挑战将进一步升级。

为了降低生产环境中的风险，就要制定相应标准供所有供应商遵循。这些标准，连同有关组件规格以及交接质量检验的大量文档与沟通，可以帮助减少这些问题的发生。在应用开发中，组件接口的行业标准已经制定，但是团队还必须依靠合同及服务水平协议(SLA)来协助克服这些交接挑战。利用自动化工具，而不是手动进行交接与部署，有利于降低交接过程中可能出现的风险。将组件从供应商移交给组织的应用交付团队时，充分利用整个组织及其供应商的集成工具链至关重要，这样可以确保降低移交风险。

6.7.3　API 经济与安全

除了这些安全漏洞，API(应用编程接口)经济的发展趋势引发出另外的安全问题。随着越来越多的 API 出现，API 本身或者其恶意用户所带来的安全漏洞风险会破坏所开放的系统。要降低这种风险，就需要将强测试协议应用于 API 及使用 API 的应用。

API 供应商必须确保不对可能破坏系统的恶意用户开放其所开发的 API。这些安全漏洞可能是由恶意开发人员有意或无意引入的。API 顾客必须确保其通过 API 访问或提交的数据是安全的，并且其应用能够正确使用 API，而不会将 API 暴露在任何安全风险之中。供应商及提供者必须使用适当的身份验证与供应协议来确保 API 的唯一许可使用权，任何第三方不得滥用所提供的 API 访问权限。如图 6-4 所示，API 的安全性是大多数 API 管理工具的核心组成部分。

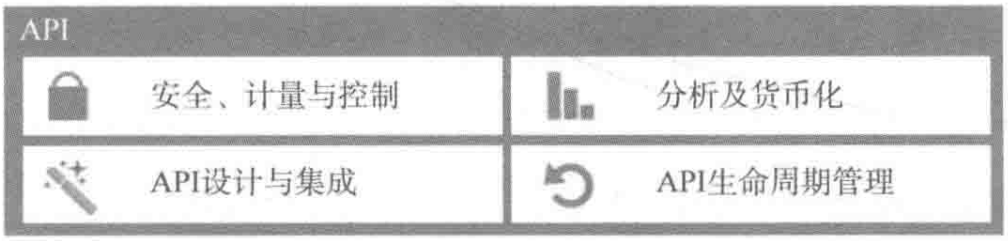


图 6-4　API 管理工具的 API 安全性

要利用 DevOps 原则降低这种风险，就必须持续进行严格的安全测试以确保 API 安全的健壮性。API 的供应商与用户都必须对 API 的每一个新版本执行严格的自动化测试。他们必须进行持续测试，以确保能够及时发现并处理滥用或破坏 API 的行为。API 本身的部署流程中必须要包括这些测试，以确保能够对 API 以及应用 API 的所有应用的部署流程进行持续安全测试，从而确保应用的安全性。安全测试与安全监控是跨越开发、测试与生产环境的一个连续过程。

综上所述，精益原则在工厂里及软件开发上的应用减少了浪费与返工。同样，正如工厂自动化导致了一系列新的潜在安全风险并且需要采取措施去降低这些风险一样，实施 DevOps 实践也会引发新的安全风险。本章介绍了一些有关 DevOps 实施的风险以及相应的应对方法。

通过快速交付以及软件开发、测试及交付的敏捷方法，实施 DevOps 所引发的安全风险是明确且易于解决的。但这些安全风险不容忽视。正如忽略工厂自动化相关的安全风险会带来严重的质量挑战，不解决 DevOps 实践中所暴露出的相关安全风险同样也会导致严重的质量问题。随着 DevOps 从具有整套指导原则的哲学理论发展为具有相应实施路径的定义明确的实践，解决这些安全风险必须成为实践中固有的组成部分。

通过将组织的安全团队纳入到 DevOps 生命周期中，实施 DevOps 的组织与团队能够确保安全风险的降低。这些团队需要成为负责分析决策的利益相关者，去分析并确定组织中不同项目的相关风险，并制定策略去处理或降低这种风险。安全团队必须为 DevOps 环境提供以安全为中心的质量门禁，作为协作的一个示例。

6.8 方案：DevOps 与外包

NBA 与外包

事情发生在几年前，却与 2016 年 NBA 选秀第一轮的情况相吻合，那场比赛见证了一个令人震惊的情况。

第一轮比赛中，30 位选手中有 14 位是外籍球员。如果再加上多曼塔斯·萨博尼斯，第一轮选拔中就有一半的选手是外籍球员了，他没出现在名单里，是因为这个立陶宛的大个子出生在波特兰，而他声名显赫的父亲正效力于波特兰开拓者队(Trail Blzers)。

NBA 历史上从未有过这么多的外籍球员出现在第一轮比赛中。

至少有一个国外长大的未来杰出球员相信，球员们在国外能够更好地学习传球、运球以及团队合作。

科比·布莱恩特去年反驳说，“我只是认为欧洲球员的技术更加娴熟。”“他们只是在小的时候就学会了正确的方法。”

外包同样适用于 NBA 的底线。哈基姆·奥拉朱旺、保罗·加索尔还有德克·诺维茨基不仅帮助球队赢得了冠军，自己也收获了联赛球迷

和财富。外国人关注 NBA 是因为 NBA 关注外国人。比起终极格斗冠军赛(UFC)以及美国职业足球大联盟(MLS)在电视转播合约中的收益，联赛在中国的流媒体交易中每年都能获得更多的收益。当美国人正在为迈克尔、马吉克或者拉里美好时代的逝去而惋惜时，国际球迷们已将现在视作黄金时代。

—— Flynn，2016

谈及实施 DevOps 的组织，另一个随之而来的问题总是关于外包的。大多数组织至少将应用交付或 IT 运维的部分工作外包给外部供应商。这可能是传统意义上的离岸外包，即将工作外包给外部的、离岸的并且通常是更便宜的供应商，也可能是真正的供应链模式，即外部或内部的供应商交付的是应用交付供应链中的组件。这两种情形都会对 DevOps 实施产生不同的显著影响。

6.8.1　战略外包

在这种情形下，从业务的角度而言，企业认为将全部或部分的应用交付工作外包给擅长这方面的其他供应商去做会成本更低或者效果更好。做出外包的决定，可能是基于成本考虑，也可能只是单纯因为组织认为其自身不需要具备这项能力，最好是交给其他人来做。最常见的例子应该就是公司雇用像 IBM 这样的组织来运行其数据中心。组织选择不雇用工作人员而是让 IBM 来运行数据中心，是因为这样做是有道理的。另一个例子是零售商雇用外部的供应商来构建并交付移动应用，这是因为他们可能在战略层面上决定不必拥有这些能力。他们决定将业务外包给提供移动应用服务的公司，而不是从零开始创建一个新的移动团队。

在应用构建与运行整体外包的情况下，实施 DevOps 不再是一项很大的挑战。将整个应用生命周期外包的时候，也就是外包了交付流水线。如果将移动应用的开发与运维整体外包，应用开发与运维的挑战就只是确保移动应用能够访问所需的后台系统而已，希望是通过明确定义且管理良好的 API 来实现。在所描述的第一种情形下，如果组织是在内部构

建应用并交付到外部供应商所管理的生产环境中，组织就需要将应用移交给外部供应商，并从供应商那里获取适当的反馈以实现持续改进。只有外部供应商与组织密切合作，规范彼此之间应用交付及反馈的实践与工具，才能在这样的多供应商交付流水线中采用持续交付模式。

这并不意味着轻视所需的计划与协作，但是如果外部供应商是真正的合作伙伴，这是可以实现的。应用所属的组织仍然需要拥有项目组合管理、计划、发布管理以及所交付应用的治理权限。当然，如果供应商因为生效的合约中没能提供 DevOps 方式的合作模式而不愿意合作，组织就需要让律师介入并就合约进行重新谈判。

6.8.2　IT 供应链

在供应链模式下，实施 DevOps 所面临的挑战变得更加有趣，不是外包整个交付流水线，而是单独的组件由供应链中的独立供应商来交付。企业外包的可能不全是外部供应商，如图 6-5 所示，更多的可能是内部与外部供应商的组合。内部供应商比较容易处理。除非是政治原因或者高级管理层不够支持，你也可以运用 DevOps 原则来获取供应商。创建企业级的中心化的构建物仓库(发布的唯一来源)，或者采用标准化的集成工具集来实现端到端的可追溯性，这类最佳实践对于获取所需的支持大有裨益。

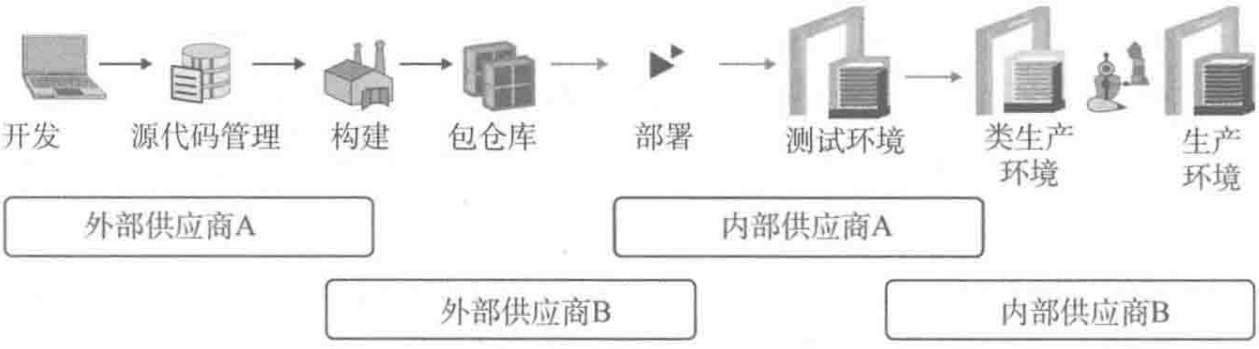


图 6-5　软件供应链

如果组织有外部的供应商，情况将会变得棘手。多个供应商分别开发并测试各自的组件是需要多对多的协调与合作的。合约成为其中的阻碍。如果两个供应商彼此之间无法直接联系，而是不得不通过组织才能进行沟通的话，就有问题了。如果每次业务负责人试图根据反馈实施变

更(根据 DevOps 实施的要求)的时候，供应商都拿出合同或者向组织收取变更费用，那肯定会有问题的。供应商每次将组织的外部开发测试环境变更为基础的虚拟机环境时都会收取 10 000 美元的变更费用，这样的恐怖故事经常可以听到。如果组织负担不起环境调整的费用，那么与生产类似的环境就不再是一个可能的选择。

这里唯一的解决办法就是尽量让外部供应商看到与组织及供应链中的其他供应商合作实施 DevOps 的价值。如果看到了 DevOps 在提高效率及减少浪费方面所能带给他们的价值，而且能够使其在更短的时间内、利用更少的资源交付更高质量的软件，他们就会接受 DevOps 实施。然而，如果合同约定了更快交付、更高效交付或者需要更少人员，而这触碰了他们的底线，就做不了什么了。

6.8.3　利用外包实现 DevOps

所以，外包是 DevOps 的坟墓吗？或者 DevOps 是外包的终结者吗？根本不是这样。组织内部不可能拥有所有的 IT 技能。他们需要从外部供应商引入专业知识，所以需要外包的存在。DevOps 的出现，以及实施 DevOps 所需的协作、敏捷以及响应性反馈的需求，要求未来做合同时必须把这些目标都要考虑进去。这并非是不合理的期望。大多数的系统集成商已经在合作伙伴发来的需求建议书(Request For Proposals，RFP)中看到了这点，这些组织希望在实施 DevOps 的道路上能与其建立伙伴关系。这实际是一个必然趋势。降低成本、提升创新速度、更加敏捷以及快速响应市场的需要，所有这些外部的压力都迫使组织必须实施 DevOps。他们也迫使外包商进行角色转变，从单纯的供应商转变为客户合作伙伴。DevOps 带来了全新的外包模式。

6.9　总结

总结本章内容，在多个独立的、协同工作的、自组织的团队中，为企业级程序及项目推广 DevOps 实施，其关键全都无外乎是文化及团队

合作。本章介绍的七个方案如下：

- DevOps 能力中心
- 发展大规模创新文化
- 发展持续改进文化
- DevOps 团队模型
- 工具与流程标准化
- DevOps 的安全性考虑
- DevOps 与外包

这些方案都是关于如何实施 DevOps 文化以及如何实现必要的团队合作，其中包括何时可以实施软件外包或者软件供应链模式。此外，由于安全是推广 DevOps 实施的主要障碍，要确保将解决安全问题作为 DevOps 变革中不可或缺的部分。

本章的主题强调了文化变革以及职能仓筒、团队、利益相关者、项目之间进行适当沟通与协作的必要性，这些主题是：

- 组织文化
- 工具与实践的标准化
- 有组织的实施
- 打破组织仓筒

最终，这一切都取决于组织克服文化惰性的能力。将组织现在的文化转变成互相信任的文化，即利益相关者可以自由地沟通与协作，而不必受制于不必要的管理、晦涩难懂的政策、脱节的工具以及僵化的团队结构。变革文化，使流程衔接的操作人员能够互相信任。

第 7 章

引领企业的 DevOps 实施

总经理 VS 教练

在国家橄榄球联盟中，总经理或许是每个球队最为重要的职位了。星期天，总经理却大多会被遗忘，取而代之的是场上人员。我们的注意力都在球员和教练身上，但握有球员和教练生杀大权的却是他们的总经理。一个不好的总经理可以毁掉任何一支球队。一个好的总经理却可以让你赢得超级碗。

从 2008 年到 2010 年，斯科特·麦克洛汉暂时放下手头一些研究性的事物，在旧金山集结了 49 名队员打入 2011 国联决赛(NFC Championship)以及 2012 超级碗的选手名单。2010-2014 年，他也为西雅图海鹰队做了同样的事情。海鹰队比 49 人队做得更好，并在 2013 年赢得了超级碗比赛。

——Foss，2015

在专业的美式橄榄球队中，总经理扮演着至关重要的角色。总经理做得好，可以成就一个球队，也有很多球队的总经理做得并不好(Breer,2013)，球队因此被毁掉。这是因为虽然总经理位居幕后，却主宰着参加比赛的球员与教练以及比赛的输赢，负责把球队真正凝聚起来的人是总经理。总经理还会通过管理薪资以及更为重要的球队文化来鼓舞士气。不管这个角色是由所有者(想象一下杰瑞·琼斯和达拉斯牛仔队)

总经理(本该如此)还是由教练(又如比尔·贝利奇克和新英格兰爱国者队来扮演)，这份责任都不容轻视。

前面的章节关注的都是球员与教练。这一章，我将焦点移向了食物链终端的高管们，他们有权决策为 DevOps 变革聚集力量、准备商业案例、提供资金并能够引领 DevOps 的变革。他们是 DevOps 变革所交付业务成果的所有者。他们是创造 DevOps 转型愿景以及如何实现该愿景的决策者。同球队一样，不同组织中的职位名称各不相同：CIO、CTO、高级副总裁、副总裁、CDO(Chief Digital Officer)甚至 DevOps 副总裁，不管如何称呼，其目标都是一致的。

最重要的是，高管就该身先士卒。他们是推动文化变革的人。他们必须有能力抽身出来，以识别出组织中现存的需要克服的文化惰性。他们必须对所需要的目标文化有清晰的愿景。他们需要授权其相关负责人对流程、管理政策、衡量标准以及拟定的业务目标进行必要的调整，从而改变团队人员的行为模式。他们还需要创建自上而下直到操作人员层面的授权文化，以便操作人员能够发现流程、管理政策、衡量标准所存在的问题并提出相应的改进建议。换言之，他们需要创建从最高管理层直到基层的持续改进文化。

组建成功团队

教练拉德与加利福尼亚康科德 De La Salle Spartans 球队的故事绝对是有史以来最伟大的领导力故事之一，因为它道出了组建成功团队的核心问题，不管是体育运动队还是业务团队。这是一项非常宝贵的团队建设经验，创造了体育史上最长的连胜纪录。

胜利源于更大的使命。伟大的教练知道如何执行获胜的战术，但是激励往往来源于超越亲吻及拥抱的无形力量。海因斯说："鲍勃·拉多塞尔所做的任何事情都可能导致 De La Salle Spartans 球队不能获胜，他们取得胜利是因为他们自己。"

教练拉德代表着比赢得比赛更大的意义；他代表着承诺、责任，还代表推动人类成就的极限。"作为教练，你知道该阻止什么人以及采用什么战术，但是如果孩子不能正确认识自己，这些都毫无意义。"拉多

塞尔说道。“我们的孩子不是为了胜利而战。他们在为我们所代表的信念而战。”

尼尔·海因斯向【interviewer Carmine Gallo】讲述了他意识到必须讲述教练拉德的故事时的情形。事情发生在比赛的中场休息时间，他的球队当时表现不佳。教练走进更衣室，他的队员看着他们的教练，请求得到智慧，即他的指导。拉德没有像往常那样讲一番鼓舞士气的话，而是说道：“为什么必须解决问题的人总是我？集体解决问题是你们将受用一生的技能。搞定它。”随即，史上最成功的高中橄榄球教练走了出去，留下队员们自己寻求解决办法。教练拉德的更大使命是利用橄榄球作为工具传授生活的经验。这个例子完全符合这点。

—— Gallo，2014

这一章中，我会介绍具体的方案，但不再有其他章节中那样的主题。这是因为所有这些方案只有一个主题：引领组织的 DevOps 变革。

7.1　方案：DevOps 作为变革运动

改造科技巨头

IBM 首席信息官(CIO)杰夫·史密斯想推广敏捷软件开发及项目管理实践，就规模而言，他指的是员工数量与迈阿密竞争对手相当的公司。

史密斯先生于 2014 年年中加入 IBM，此前在一家澳大利亚金融 Suncorp 集团担任 Suncorp Business Service 首席执行官。他在 Suncorp 引领了一场技术变革，这个项目是与技术供应商 IBM 合作进行的，其付出的努力使 Suncorp 在质量、周期时间以及成本结构方面获得全面改善。所以，IBM 聘请其出任新的首席信息官。

史密斯先生在 IBM 领导一个 20 000 人的全球 IT 团队，这个团队为 IBM 约 380 000 名员工的工作创建工具并提供服务。很少公司有这样的经营规模，这几乎与人口 418 000 的迈阿密市一样大了。他立即着手进行 IBM 的 IT 改革。

“使命就是实现创新和小公司的速度……，看看我们能否在规模上做到这一点。”他说道。《华尔街日报》曾报道称，对于正处于痛苦转型期的 IBM 而言，这是刻不容缓的事情。《华尔街日报》的罗伯特·麦克米伦写道：“由于 IBM 退出了低端服务器业务，转而将重点放在盈利更高的 UNIX 和大型机上，其硬件销售在第一季度继续下滑。”IBM 将未来寄希望于软件与服务上。

史密斯先生称金融危机期间，他关于协作和工作流的想法更加强烈，那段时期，Suncorp 的所有资金链断裂，濒临破产。他负责引领技术开发及其他业务服务，帮助 Suncorp 在残酷的变革中生存下来。

而今，在正面临着业务挑战的 IBM，他已经转变了公司的传统软件开发模式。此前，公司的 IT 部门被划分为两个分支，一个是变革或是开发团体，一个是运行团体。他说道：“这更像是典型的 IT 商店架构模式。”

有一些 IT 使用了敏捷开发与 DevOps，将项目分成小的单元，通过这种方式，小型团队得以高速运行。

二月份，他用分别有专人负责的 25 个域替换了旧系统。这些域涵盖了从开发云环境到负责市场的所有团体，IBM 的员工可以在此下载所需的工具。“重要的一点是如何分解大的问题以及如何利用我们遍布世界各地的人力资源。”他说道：“最难的部分就是获取人才。而我们已经拥有。人才和工具都准备好了。工作方式正在改变。简而言之，那就是我的使命”。

—— Rosenbush，2015

组织迫于市场及竞争对手的压力而实施变革。变革总是需要有改革能力的领导者。在 DevOps 背景下，变革领导者应该来自 IT 组织，就像 IBM 的首席信息官杰夫·史密斯，他一直在引领 IBM 进行 IT 流程与系统变革，以使其变得更加敏捷。虽然 IBM 的一些 IT 部门以及部门或分支机构中的一些开发交付团队已经完成了变革，杰夫还要驱动整个组织都进行变革，建立流程及系统以将变革推广到全公司范围内。

对于任何实施变革的组织而言，除了变革本身，还需要一个触发器

或催化剂。实施变革必须要有令人信服的理由。变革理由一经确定，领导们就要据以采取相应的变革行动。商业史上不乏这样的公司，他们清楚变革的需要，却因为领导不力或者没有能力利用令人信服的理由采取变革行动，以致变革失败并最终走向灭亡。他们看到了威胁，却未能成功采取行动，不论这威胁是来自于市场的转变(伊士曼柯达消失于胶片到数码摄影的转变中)，来自于竞争对手(Blockbuster 在 Netflix 和 Redbox 的威胁下破产)，还是来自于法律问题(整个 Napster 商业模式都忽略了小小的“版权”问题)。

7.1.1　令人信服的行动理由

让我们看看有哪些令人信服的行动理由。IBM 副总裁 Rashik Parmar 在驱动云实施时也提到了这样一个类似的清单。然而，大多数的理由都是各类变革，特别是 DevOps 变革。最常见的令人信服的行动理由包括：

- **兼并、收购与剥离。**兼并、收购与剥离是实施 DevOps 变革最为常见的理由。兼并与收购会带来新的人员、应用、系统、工具以及平台。他们还会将自己的流程以及更为重要的文化引入团队。这些团队需要整合到一个新的标准化状态，以适应新的联合组织。相反，剥离导致人员、应用、系统、工具以及平台的流失，从而衍生出一个机会，可以重新审视剩余部分并进行优化。这两种情形都是驱动整个组织范围内实施 DevOps 变革的绝佳机会。
- **意外服务中断。**没人愿意因为服务中断而接受新闻报道。不管能否成为新闻，所导致的业务及声誉上的损失对组织而言都是毁灭性的。不管你的组织是家银行，顾客因无法进入账户而不满；还是无法处理客户付款的咖啡店，或者是不得不将所有航班停飞几个小时的航空公司，那都没有关系。当服务中断，尤其是影响广泛的服务发生中断时，这种情况是会发生的。通常组织会进入战备状态，第一时间进行恢复并确保中断不再发生。后面的步骤就可以用来启动 DevOps 变革。
- **发起创新倡议。**无论发起创新倡议是为了对抗竞争对手、应对

市场变化，还是因为外聘了新的高管，这些全都是可以用来启动 DevOps 变革的令人信服的理由。这种行动理由为发起创新倡议的高管在组织中发展边缘创新提供了机会。

- **提高 IT 生产力。**对某些组织而言，创新是令人信服的行动理由，但对于另一些组织而言，创新却是显著提高 IT 组织生产力的需要。提高生产力的核心是优化，而实施 DevOps 方案却是实现优化的必经之路。
- **增强 IT 敏捷性。**在当今的 IT 环境中，关于增强敏捷性的任何讨论中都需要包含关于云计算的对话。如前所述，云计算是 DevOps 的推动者(因此而敏捷)，而 DevOps 又是云计算的核心驱动力。如果组织已经踏上云计算之旅，就需要启动 DevOps 会谈，来讨论如何部署到新的云托管平台，如何利用云托管平台进行交付，才能最大限度地发挥云服务的优势及功能。因此，云计算应用通常是 DevOps 变革的一个扩展集。

不管采取行动的理由是什么，它都为 DevOps 变革提供了创建业务及技术案例的理由和必要机会。这样的机会在组织的生命周期中并不常见。至少其中的某些理由(如服务中断或者兼并、收购)，你会希望越少出现越好。当这些机会出现，就应该作为组织应用、流程、工具、平台以及文化整体变革的机会加以利用。

7.1.2 DevOps 变革的反模式

变革只有被当作变革对待时才能发挥作用。虽然这个表达听起来多余，但却是至关重要的。太多的组织不能成功实施 DevOps，就是因为他们的领导没有专注管理变革，也没有为此付出必要的努力。他们陷入了常见的变革反模式。这里列举了一些关键的反模式：

- **DevOps“项目”。**根本没有所谓的“DevOps 项目”。DevOps 实施不是一劳永逸的事情。DevOps 实施是一项变革，需要对流程、工具自动化、平台以及文化都产生影响，而且这种影响需要是持续的，以最终实现文化的持续改进。
- **缺乏所有权。**尽管高管们可能亲自负责 DevOps 变革，明确流

程、工具、平台以及文化中每一项变革能力的所有权还是极其必要的。这种所有权必须一路下行至基层。没有明确的所有权和责任，即谁来负责哪方面的变革，利用什么资源，以及什么时候进行，这些事情不明确，变革就无法进行。法令无法让人做出改变。

- **只关注度量指标的错误做法。**如前所述，DevOps 变革要求识别出注重改进的适当衡量标准，这是先决条件。然而，过分强调注重改进的度量指标也是有害的。如果只把改善某一套指标作为人员的衡量与奖励标准，而不去关注学习，那么当这些指标无法改善，或者没能实现改善的时候，团队就开始对这些指标“动手脚”了。这就导致了一种有毒的文化。
- **在孤立组织中进行 DevOps 实施。**DevOps 实施通过试点项目启动(本章后面部分将讨论的内容)。然而，试点项目的目标是学习并复制从其他项目中学到的成功实践及经验教训。在高管将这种复制与自己的领域，即自己的单元、部门或者程序隔离开来时，常见的反模式就出现了。如果他们的领域是独立的，并且不与组织的其他部分进行交互，那么 DevOps 实施就无法实现预期的结果。这肯定无法改变整个组织。在介绍“利用试点项目启动”方案时，我会讨论选择合适试点项目以及利用每个试点项目经验教训的价值。
- **改变组织汇报结构。**第 6 章介绍的分队与部落团队模式导致了跨职能团队的产生，以及规模很小但也可能会扩展为部落那样的大型结构的团队。然而，有很多常见的反模式却导致了错误的架构重组：
  - **新的领导角色。**任命 DevOps 副总裁，配备向其汇报的开发与运维团队，这解决不了多少问题。决策、冲突解决、预算分配以及一些沟通或许会变得更加容易，但仍然有两个仓筒组织没有改变，沟通、协作与信任方面也没有得到改善。
  - **新的仓筒。**另一种不起作用的重组方法是建立新的仓筒。许多组织将他们的利益相关者重新分配到新的汇报架构中，这

些架构不会促进所需职能间的跨职能协作，但仅限于新团队中的利益相关者。这只会导致旧仓筒为新仓筒所取代。

- **DevOps 团队。**关于创建 DevOps 团队的优势与劣势，已经存在很多争论(Minick，2015)。这些 DevOps 团队通常只有开发与运维人员。由于其他人员被排除在外，DevOps 团队无法产生足够的附加价值。但在跨职能部落中，情况却并非如此。而且，有些组织将 DevOps 团队打造成每一个项目所必需的利益相关者。这最终使 DevOps 团队不再只是另一个官僚团队，现在必须通过项目来证明其价值。

- **外包 DevOps 实施。**尽管顾问、专家、供应商和外包商对于组织引进流程、工具、平台以及文化变革知识通常都是必不可少的，但 DevOps 实施的所有权不得外包。如果团队没看到自己的高管在负责并引领变革，那么就无法在整个组织内驱动变革的主动性与紧迫性。文化惰性是无法通过外部顾问或依靠阅读这本书来克服的。

- **沟通与协作。**利益相关者之间需要真实、直接的沟通与协作。他们能做到这点吗？还是他们总是需要通过各自的汇报链才能进行沟通？协作只能通过任务单来实现吗？有没有阻碍利益相关者跨越组织边界工作的退单要求？向供应商提出的要求是否总能触发正式需求变更流程？

- **合同。**这已成为外部生产商和供应商的一项挑战。如果合同严格要求生产商和供应商只能以特定的方式进行沟通，或者要求流程的每一项变更都要通过正式的合同变更流程来执行，那就不存在 DevOps 了。

处理这些反模式是引领变革的高管们的责任。这可能会导致一些艰难的决策以及激进的变化。可能需要部署新的协作工具。管理层需要放弃控制权，以允许自由的沟通与协作。跨职能仓筒间的协作如果仅限于依靠任务单进行的沟通，就需要彻底消除。需要为操作人员提供关键 IT 服务的自助服务目录。合同需要重新谈判，供应商需要更换，而且(希望如此)律师和会计也要解雇。

7.2　方案：发展协作信任的文化

> 球员需要相信并尊重这样一个事实，那就是如果我做好自己的工作，我们就拥有了成功的最佳时机。我不必制定所有战术；当需要我的时候，我只需要制定填补空白的战术，而且相信我身边的人也会这样做。
>
> ——尼克·萨班，阿拉巴马大学橄榄球队总教练

为什么大型组织中的运维团队不能像初创公司那样给予开发人员直接自助访问生产环境的权限以进行持续部署呢？原因很简单，那就是他们不相信开发团队能交付稳定、安全和可靠的应用。当然，事实并非只是缺乏信任这么简单。还有一些其他的原因，其中包括禁止开放生产环境访问权限的合规性要求。此外，有些不信任实际上是积极的，因为这能推动运维团队去质疑并验证开发人员所交付的产品。然而，完全缺乏信任就不合理了。在大型组织中，缺乏信任似乎已不再只是开发与运维的专利。开发不相信业务分析，企业架构不相信运维，测试不相信开发，审计与合规管理团队不相信任何人，没有人相信管理等。缺乏信任导致团队无法有效进行跨职能仓筒的沟通与协作。不信任是关乎文化的事情。

李·里德(他担任 IBM 的 DevOps 架构师)在其撰写的 *The Simple Math of DevOps* 一文中提出了一个计算总交付时间的公式。如下所示，在公式中，将信任作为分母，因为信任决定着利益相关者之间交接时的效率，而且，如果利益相关者不信任从其他利益相关者那里接收到的构建物，那么交接时就可能会产生浪费。引自李·里德的文章：

决定价值产生速度的关键因素是信任。就如何在给定的客户端完成工作来绘制一幅价值流图时，这点非常明显。软件交付团队成员随着生命周期的推进渐渐对工作有效性失去信任，由此导致大量的返工和浪费。用数学术语来表达，我们的方程式变为：

$$T_{交付}=\frac{T_{计划}+T_{设计}+T_{开发}+T_{构建}+T_{部署}+T_{测试}+T_{修复}+T_{发布}+T_{评估}}{\%\ 信任(0-1)}$$

也就是说，我们在交付周期中所做的任务是受交接过程中我们彼此之间的信任程度影响的。如果信任度为零，那么我们的 $T_{交付}$将是无穷大的(除以零)。信任度为 100%，我们的 $T_{交付}$将只会受到每项任务执行速度的制约。

如何创建信任的环境与文化呢？当然不是通过“背摔”(fall-back-into-the-arms-of-the-person-behind-you)练习来实现。信任来自于使命感及团队意识的建立，要使人们开始有意识地与身边的人合作，并且理解信任身边的人是通往成功的唯一途径。他们需要相信共事的人，理解他们的角色，并且相信他们不仅擅长所做的工作，而且会尽力做到最好。这种信任感必须从头开始建立。必须利用像第 6 章中所介绍的声田(spotify)部落模型那类的模型来创建小型团队，由此建立信任文化。要授权这些团队作为拥有明确共同目标的一个整体跨职能仓筒运行，这个共同目标是需要各个团队齐心协力来完成的。然后他们需要在整个组织范围内推广这种模型。最终，互相认识的人彼此信任。他们信任交往的人并乐在其中。他们信任那些对自己周围的人给予信任的人。

7.2.1 可见性促进信任

如果说有一个因素能够驱动信任的环境和文化产生，那一定是跨越功能仓筒的可见性。可见性还能帮助引领 DevOps 变革的高管们确保组织进行正确的投资。由于交付流水线中的利益相关者彼此合作并相互移交构建物，可见性因此成为信任的驱动力。如果功能仓筒之间没有可见性，又出现了一些问题，使用构建物的利益相关者将对移交构建物给他们的利益相关者失去信任。比如，如果测试人员从开发人员那里收到的

代码总是无法通过基础测试，那么他们就会对开发人员传过来的所有代码产生怀疑。但是，如果他们对开发人员在将代码交付给他们之前所做的各种质量检测具有可见性，他们就会与开发人员一起确定测试失败的原因。此外，如果开发人员知道测试人员对其在代码上执行的任务具有可见性，他们就不会将未经验证的代码交付给测试人员。这个例子可以扩展应用于彼此传递构建物的任何一组操作人员。

可见性还能让操作人员看到团队中谁正在从事什么工作任务，谁是合适的专家，谁负责他们所需的构建物，由此驱动信任。这种可见性还能够促进团队间更好地沟通。完全的可见性驱动全然的信任。

7.2.2　一切都关乎人

> 有效的领导者会避免成为破坏个人激情和球队整体表现的杂草。相反，他们会成为“水源”，在超级巨星团队中培育、指导并培养个人激情的种子。
>
> ——蒂姆·霍华德，前美式橄榄球运动员

归根结底，一切都是关乎人的。因此，领导者的首要任务就是培养和发展人。首先从招聘开始，接着是把合适的人放在一起组成团队，最终是要提供一个有趣的环境，使团队得以茁壮成长并实现生产力最大化。

如果你有机会去参观一家初创企业，而那家公司已经从咖啡店搬到了真正的办公室，你会看到一个完全不同于传统企业工作环境的办公空间。没有隔断，就只是开放的办公空间。似乎没有头衔，甚至连工作时间表都没有。人们穿着各种各样的衣服到处跑，说成“到处跑”一点都不夸张；事实上，他们可能是骑着滑板车或是溜着滑板的。会议室里有沙包；事实上，会议室里可能就没有正式的桌椅。到处都有白板。体育设施遍布各处，乒乓球台、桌上足球、视频游戏机、台球桌，一应俱全。是的，还有食物和啤酒，大量的啤酒。所有这些事情都是大型组织中的公司生产力“独裁者”和人力资源部门所极力反对的。大型组织认为“不合适”的这些事情，对于创建有趣的工作环境而言却是必不可少的，有

趣的工作环境能让人们享受工作的时光。

另一方面，大型组织在每间办公室的每个位置都打上标准化的隔断。没有体育设施。中午 12 点到下午 1 点是午餐时间。沉默是金。有谁会不想在 5 点钟下班的时候立即离开这样的地方呢？

创建有趣迷人的工作环境已经成为吸引优秀人才的前提条件。并非每个人都期待每天下午的免费按摩或者休息室里随时可用的啤酒桶，但是，年轻的技术型专业人才是不愿意在其所不喜欢的环境中浪费生命的。对于当今的年轻一族，尤其是千禧一代而言，工作/生活平衡并不仅仅意味着工作之外要有生活，在工作中拥有生活也是很重要的。

创建一个团队，需要人们愿意彼此合作，享受并热爱自己的工作。创建这样的环境是高级管理层的责任。

7.3 方案：业务线的 DevOps 思维

球队老板改变了 NBA

“极客们将接管地球”，而今已是一个老生常谈的传说，讲的是那些主宰着二十一世纪商业世界的科技精英们势不可挡的崛起。美国体育业在激烈的角逐中并没有明显的财富增长。通常，在说到美国职业运动队所有权的问题时，人们总会想到个性鲜明、夸夸其谈的商业大亨及其银行余额。但是正如企业已经开始跟随科技新人类的脚步翩翩起舞一样，体育业也发现自己正日益陷入科技行业领导者垄断财富的控制，这些人如果违约，后果将是毁灭性的。尤其是 NBA 比赛中，诸如洛杉矶快船队的史蒂夫·鲍尔默(Steve Ballemer)、达拉斯小牛队的马克·库班以及萨克拉门托国王队的维韦克·拉纳迪夫这类的特许经营者发挥了重要作用，撼动各自球队以及更广泛的联盟以使其能够满足技术性目标。每个人都采取了截然不同的方法来管理自己的所有权，但进行数字化工作和思考的时候，所有人都处于自然状态下，并且不需要浪费时间利用技术去获取竞争及商业优势。

——Robbins，2015

DevOps 变革需要业务线(Lines Of Business，LOB)的参与。毕竟，IT 的奋斗目标是完善自己，以交付现状条件下所无法实现的业务需求及目标。因此，业务线要成为 DevOps 变革的支持者，通过提供投资与时间、通过改变参与及 IT 交互方式来参与变革。他们要成为变革过程中全面的利益相关者。他们需要提出问题以帮助 IT 组织找出业务需求与其交付能力之间的缺口，并帮助 IT 组织进行变革以弥补识别出的缺口。需要提出的三个关键问题是：

- IT 组织能否利用现代架构快速交付新的创新型应用？
- 能否现代化现有的应用，使其能够实现更快交付及创新？
- 他们能否实施成功所需的文化、工具及流程？

7.3.1　业务线与 IT 的接触

让我们来研究一下业务线与 IT 之间的接触这一话题。通常，这种接触是正式的并且很少发生的。业务线识别出某些需求，帮助确定下来，然后结束，而 IT 开始创建解决方案以交付这些需求。这不是那种持续互动以帮助进一步细化需求，或者在 IT 运行实验去识别需要解决的问题及寻找正确的解决办法时参与其中的关系。业务线能够发挥至关重要的作用。IT 进行实验时，他们不仅能够代表自己，而且也能代表客户以及更广阔的市场。

业务线还需要持续改进自身的需求，明确想要交付给顾客的业务价值，并不断加深其对于尝试解决的业务问题的理解。通过正确利用来自 IT 以及生产中使用应用的那些用户的反馈，无论这些反馈是发生在实验过程中还是在发布之后，业务线是可以做到这一点的。他们可以接收并利用的反馈包括：

- **应用使用模式。**客户如何来使用应用？他们的使用模式符合业务线的预期吗？客户使用了应用中的哪些部分，又有哪些部分没有使用？
- **新的客户用例。**客户是否按照所设计的方式在使用应用？这种使用模式能否带来业务尚未识别的新用例？用户在使用应用的同时还做了什么其他的事情？例如，如果他们同时在使用其他

的应用，那么是不是应该将外部应用的特性加入到自己的应用中(比如，用户一直切换的地图应用)？

- **用户角色**。各种不同的用户类型是否与业务所识别的角色相符？有没有识别出新的角色？有没有角色需要废弃？
- **用户情绪**。他们使用应用时的情绪如何？用户是高兴还是沮丧？他们是抱怨还是喜欢应用，并会促进应用的完善？

利用这种以及其他各种各样的反馈，业务线形成一种持续改进的模式，并致力于加强其对于用户及用户需求的理解。

为了使业务线能够利用这些层面的反馈并据此采取行动，反馈需要是：

- 可以使用的
- 可以操作的
- 及时的

确保反馈能够为业务线所利用，这是 IT 组织的责任。发送服务器日志毫无意义，应用使用模式或者以前版本的用户情绪数据也同样没有意义。

7.3.2 参与 DevOps 变革

一旦 DevOps 变革启动，在实施过程中，业务线除了参与应用交付周期并根据所提供的反馈采取行动外，还能够利用其自身影响力帮助切实推动 DevOps 的实施。他们需要与包括 CIO、CTO 或者高级应用开发管理人员在内的利益相关者合作，共同开展以下工作：

- 在新的技术领域及平台，为开发团队寻找学习机会以超越竞争对手，实现快速反应并加快创新。
- 优先处理现有应用的优化领域、增加创新并释放创新资源。
- 支持 DevOps 价值流图工作坊并参与其中。

7.3.3 让影子 IT 走出阴影

最后，业务线还需要解决影子 IT 的问题。业务线在组织内创建一些影子 IT 以协助执行其认为自己的 IT 组织无法满足的那些 IT 需求。

更好的办法是翻转影子 IT 模型，在自己的 IT 组织内部投资开发影子 IT 所能提供的技能与敏捷。

如果确实有特定的需求，因为内部 IT 无法按时交付而必须利用外部的 IT 供应商，那么业务线与外部 IT 供应商接触时应该与内部 IT 一起。这就将影子 IT 带出了阴影，并使其能够在内部 IT 领域进行利用，利用其中一些核心的流程，却不会抑制其创新能力。一个常见的例子就是，使用像 IBM Bluemix 这样的私有 PaaS 的企业 IT，IBM Bluemix 由 IBM 托管，但可以在组织 IT 自己的防火墙内存在，利用第 5 章中介绍的本地部署模型，甚至还可以让其在自己的数据中心中存在。

7.4　方案：利用试点项目启动

吸引球迷实验

纽约喷气机橄榄球队营业部执行副总裁马特·希金斯认为，在某种意义上，浏览能力是无可比拟的，它是如此吸引人以至于可以让人从头至尾观看一场直播比赛。在喷气机队位于新泽西的新体育场，通过在球迷就座时将其插入数据流，球队正在实验如何保持球迷对于比赛的兴趣。

他说道："如果没有全面的经验，就会产生认知失调。"

一个实验是与位于纽约市的小型技术创业公司 Pre Play Sports 合作进行的。该公司设计了一款手机游戏，球迷们在观看比赛时预测每场比赛的结果会获得积分，越难预测的，积分越高。

最初的版本中，Pre Play Sports 只允许球迷对某些比赛的结果进行预测。但用户很快就要求提供权限以猜测每次开球的长度以及每个教练挑战的结果。安德鲁·戴恩斯说道，他去年从康奈尔大学毕业后不久就创办了这家公司。

"为了持续吸引用户，我们必须要提供越来越多的内容。"他说道："我们做到了 99%，最后的 1%却是他们最想要的。他们想要预测抛硬币的结果。"

> 根据戴恩斯先生的说法，Pre Play Sports 的玩家用在此款应用上的平均时间是 40 分钟，按照智能手机的标准，这已经算是非常长的时间了。他认为如此容易就能说服人们去玩的原因是，他们就已经在做的事情收获了源源不断的积极反馈，即在 3-4 个小时的橄榄球比赛中，为每一次微小的进展注入细微的情感赌注。
>
> —— Brustein，2011

验证一个想法，并在验证过后进行微调的唯一途径就是运行实验。实施特定的 DevOps 能力也是如此。随着 DevOps 实施路线图的完成，为了处理识别出的浪费领域，就需要对实施所需的一系列能力进行定义(通常是通过运行价值流图实践)。可以单独实施这些能力，也可以利用本书中所介绍的方案同时实施一组能力。这些能力将影响 DevOps 的各个方面，包括流程、工具、平台以及文化。应该通过运行实验并利用试点项目来实施这些能力。

理想的办法是为实施的每项能力确定一个试点项目。每个试点项目的目标都是要做到几下几点：

- 验证所实施的能力正在交付预期的结果。
- 验证实施能力所需的投资符合预算，并能产生预期的投资回报。
- 验证实施能力所采用的方式、方法是正确的。
- 确定需要对能力或者实施方法进行哪些改进，使之更好地适用于后续项目。
- 确定从能力实施过程中所学到的经验教训。

这些目标能够改善能力以及实施方法，并可以应用于组织中的其他项目。

每个试点项目只实施一种能力，是为了将实验按照能力隔离开来。如果一个试点项目同时实施多项能力，刚才列出的问题就会产生难于在多项实施能力间进行解析的答案。确定适当的衡量标准并预先设定这些标准的改进目标，对于实现期待通过实施 DevOps 能力而实现的业务目标与技术成果是至关重要的。此外，在试点项目进行之前，要先确定衡量标准的基线，以用于衡量改进的成果，这是必不可少的步骤。随着试

点项目的推进以及指标的评价，DevOps 变革的领导者还需要为其他问题找到答案：

- 能力可否扩展？其能否应用于组织中的其他项目？
- 为了实现组织内的再利用，需要对能力及相应的实施方法进行哪些改进？

7.4.1　试点项目选择

要回答这些问题，选择合适的试点项目至关重要。选定的试点项目应该是组织中的典型示范项目，要全面相关。以下是选择试点项目时应该考虑的标准清单：

- **技术栈。**试点项目使用的是组织中广泛使用的技术栈还是其所特有的技术栈？
- **部署平台。**同样，试点项目是部署到组织的标准平台还是其特定的平台？
- **团队地理位置。**团队的工作地点是集中一地还是各处分散的？组织的规范是什么？
- **团队构成。**运行试点项目的团队是典型的员工加外包构成模式吗？是否是经验丰富的老员工加上新进员工的典型组合模式？
- **团队经验。**团队是否能熟练应用工具、实践及平台，并在团队相互合作方面拥有丰富经验？是否通过为试点项目配备经验丰富的团队来确保成功？
- **项目时机/阶段。**是否在项目生命周期的适当时间实施了适当的能力？两周的发布时间是否降低了能力实施对于项目团队的优先级？
- **项目重要性。**组织是否在意这个项目？这个项目的成功对于组织而言是至关重要的吗？如果没人在意，项目就将无法获得所需的关注与资源。相反，如果是一个关键任务型的项目，则会受到太多的关注，获得太多的资源。这两类项目最终都会出局，不能成为组织的典型代表项目。

这些都是选择试点项目时应该提出的问题。为了确定每一项实施能力的适合项目，常常需要会见和访问几十个项目的负责人。

7.4.2 高层管理者支持

这些试点项目的成功离不开高层管理者的支持。高层管理者的支持要满足以下几个目标：

- 确保项目获得所需的适当资源与投资。
- 为项目提供保护，使其免受僵化流程的限制及监督管理，以避免这些僵化流程及管理阻碍其发挥实施目标能力所需的那些能力。
- 进行干预，以确保其他高管、业务负责人或利益相关者不会给项目施压，使其放弃或推迟能力实施，以满足项目既定的时间期限。
- 允许试点项目推迟甚至失败，不进行处罚，以便从经验中汲取好的教训。学习是试点项目的首要目标。

7.5 方案：在航空母舰上培养独角兽

培养特立独行的运动员

特立独行的运动员指的是从属于非主流群体及最佳实践，但能赢得比赛的人。在体育运动中，这些人了解当前的界限，并且会有意或无意地发展自己的个人策略/技能以突破这些界限。特立独行的运动员生活标新立异，因其与众不同而令人兴奋。

高山滑雪项目中，有非常著名却也备受争议的博德·米勒，自行车项目中，有 90 年代初的格拉尔米·欧伯利，F1 赛事中，有米卡·哈基宁，拳击运动中有 90 年代末的拳击王子纳西姆·哈梅德，显然，还有 60～70 年代的穆罕默德·阿里。作为各自领域中特立独行的运动员，他们并没有严格遵守当前的最佳实践技术，却取得了比赛胜利。在各自的运动项目中，他们都是变革的代言人。

最重要的是要记住一点，胜利才是特立独行的关键。运动员可以发展自己的风格(技术、技能、战术、心理)，但只有这种风格能让他们获胜、占据上风甚至消灭竞争对手，他们才能保持这种特立独行的状态。然而，获胜并非只是与众不同就可以；真正的特立独行是通过不同的行事风格产生超越对手的绩效表现。穆罕默德·阿里击败了他那个时代几乎所有的顶级选手；作为拳击项目中特立独行的运动员，他非常擅长自我激励，他在赛前、赛中以及赛后的心理战术已经成为传奇。然而，正是他的运动天赋及拳击技术，使其登临顶峰并多年屹立不倒。

几年前，在一级方程式赛车比赛中，一位采访者和车队老板正在讨论为什么选择签约新的年轻车手米卡·哈基宁，他速度虽然快，但不够稳定。车队老板的回答是："米卡是一个非常快的车手，我们可以教给他更强的技术，却无法教他开得快"。他要强调的是，"在他看来"，米卡在速度方面的技术水平/舒服区并非基于已经建立的最佳实践技术，尽管这些最佳实践技术是用以实现快速的技术性驾驶的。米卡有自己的一套快速驾驶的方法，而车队也想利用他的这套方法。

—— Hewitt，2015

无论称其为"特立独行者"还是"独角兽"，组织总有一些这样出类拔萃的人。他们是创新型的，他们不按常理出牌，他们是天生的领导者。他们很容易被公司的架构、政策、等级制度以及管理要求所扼杀，无法成为最具生产力的贡献者及团队负责人，反而失去了魔力，或者选择离开并把其技能带到其他地方。

"如果一只独角兽在航空母舰上朝东走，但航空母舰却向西航行，那么独角兽的终点会是哪里呢？"当然，独角兽的结局要么是落水，要么是放弃继续向东，转而向西走。

试着将独角兽想象成特立独行的人，将航空母舰想象成组织。培养这种独角兽的唯一办法就是把他们放到独角兽农场里，让他们与其他独角兽一起自由奔跑。创建自给自足、小型的(如部落)、能够自由运行并充满乐趣的团队就是在创建独角兽农场。信任与协作的文化培育出这样的农场，在那里自由沟通胜过层级汇报。非但不抑制，而且还鼓励创新

与实验的文化是关键要素。即使实验实际上失败了，也要鼓励从中学习并发现经验教训。

高管团队需要领导组织发展并培育这样的文化以及特立独行的态度。他们需要从自身做起，树立榜样。他们需要与团队一起努力工作，不是要榨干他们身上的生产力，而是要帮助他们建立成功的文化及态度。

培养理念

> 早些时候，我们所有的电影都很烂！
>
> ——艾德·卡姆尔，Pixar 总裁

光是培养独角兽是不够的。还需要培养他们的理念。这个理念可能并不令人印象深刻，但是大多数的理念开始时都是如此。然而，他们却可能孕育出伟大的事物。位于领导职位的高管们需要给予这种理念实验的机会。设计思维技术使原始理念得以精炼，精益创业技术利用最小化可行产品使理念得以实验，随着这两种技术的出现，验证与提炼理念的相关风险及成本已大幅降低。培养理念的黄金法则是：迅速失败、经常失败、廉价失败。实验上百个这样的理念，就可以识别出几个需要进行投资的理念，需要发布的数量甚至更少，但足以对业务产生显著影响。

引入设计思维与精益创业的最佳实践，提供弹性平台，通过建立创新优势快速开发实验用的最小化可行产品(MVP)，这是高层管理者的责任。他们不必提出理念，甚至都不需要理解这些理念。他们只需要提供相应的环境，对这些理念进行实验，并对成功的理念进行适当的开发投入。他们需要培养创新文化。他们需要让独角兽来掌管航空母舰。

总之，就像优秀的运动队老板、总经理及教练一样，组织的高层管理者也需要从业余选手中培养出冠军。

辅导 1980 年奥运会上的“冰上奇迹”

赫布·布鲁克斯不是球员们的教练。

他无情地训练球员，使他们疲惫不堪，让他们恨他。这是他作为一名教练的定位，并将伴随他的整个教练生涯。布鲁克斯和鲍勃·奈特是好朋友，这并不奇怪。和耐特一样，布鲁克斯也是一位军事型领导者。队员必须按照他的方式行事。向来如此。

在一次著名的事件中，他对球队在对战挪威国家队比赛中的表现不满意，于是警告他的球员们，如果他们不能在比赛中有所作为，就得在比赛后继续训练。最后阶段过后，布鲁克斯仍然不满意，他把年轻人带到冰面上，随着赛场慢慢空出来，强迫他们进行自杀性训练。最后，当球员们即将崩溃时，他让他们离开了冰场。他要传递的信息很清楚，要么训练，要么回家。

他后来告诉人们，在某种程度上，他希望来自东部的球员与来自中西部的球员都来恨他，而不是彼此憎恨。这两个群体之间天生对立。球员中有 14 位来自明尼苏达或威斯康星，其中又有 9 位曾在明尼苏达大学打比赛，4 位球员来自波士顿地区，并曾供职于波士顿大学球队。

1976 年，在 NCAA 冰球锦标赛半决赛期间，明尼苏达大学和波士顿大学队曾发生激烈的争执。争执太过激烈以至于比赛不得不暂停 30 分钟。1980 年的比赛中，场上的 5 位球员还曾互相挥拳。

布鲁克斯这个奉行强势训练纪律的人也有另外一面。创造奇迹后，他总是尽快离席，以便让球员们享受自己的荣耀。电视摄像机只捕捉到他离开时穿着棕色运动外套的背影。“不需要言语，只要图片”，麦克尔斯动情地说道，随即将镜头切换到冰场上欢腾的球员。

布鲁克斯后来称 1980 年奥运会那年是他冰球执教史上最为孤独的一年。戴上暴君的面具，是因为团队需要的就是这个。麦克·埃鲁齐奥内后来所言，“我坚信他爱我们的冰球队，但是我们不知道。”

—— Witnify，2014

鉴于引领 DevOps 变革是高层管理者的职责，这里还需要有一个方案，即：为 DevOps 变革创建商业案例。但是，第 3 章整章讨论的都是

这个话题。关于如何利用所介绍的工具与技术为组织创建商业案例，建议你详细阅读第 3 章的内容。

7.6 总结

总结下来，本章仅聚焦一个主题：高层管理者要引领整个组织的 DevOps 变革，需要做哪些工作？这里所介绍的方案就是高管实施手册的组成内容，高管指的是组织的教练或总经理。

- DevOps 作为变革运动
- 发展协作、信任的文化
- 业务线的 DevOps 思维
- 利用试点项目启动
- 在航空母舰上培养独角兽
- 为 DevOps 变革创建商业案例(见第 3 章)

DevOps 是一项变革运动，而不是一个项目。DevOps 变革不是一次性的，而是要持续推进的。需要不断地投入时间与精力去改变组织。世界上所有的自动化工具及过程改进都不能持续为组织提供价值，除非伴随着协作、沟通及信任这种文化上的转变。需要克服组织中的文化惰性才能实现变革。这就要求业务线发展 DevOps 思维与文化。他们也需要变革。

DevOps 变革通过试点项目启动。这些试点项目展示了 DevOps 流程、工具以及组织变更的价值。此外，试点项目能帮助组织发现如何基于需求、成熟度以及对变化的接受能力来实施这些流程、工具以及组织的变更。

高层管理者还负责照顾及培养特立独行的员工，他们是组织变革的代言人。他们负责给予这些独角兽茁壮成长的自由，允许他们按照自己的理念工作，让他们引领变革。

最后，要执行变革，甚至是启动 DevOps 变革，都需要高层管理者创建相应的商业案例。商业案例需要展示投资或业务变更的价值，还需要说明投资回报率以及变更的内容。

附录 A　案例研究

案例研究：DevOps 实施路线图案例

本附录收录了为一个大型的跨国金融服务机构开发 DevOps 实施路线图的案例。某些细节已被修改，使得路线图更通用，并且删除所有特定于公司相关的信息。当然所有的名字已经被删除了，以及特定的地理信息也已经模糊化处理以避免伤及无辜。这个特定的路线被选中是因为其广泛应用了各种 DevOps 实施场景，以及涉及一系列的行业和大小各异的组织。客户特定信息已被移除或泛化，该客户还提供了一组业务驱动因素，它们的当前成熟状态，以及技术约束、组织和财务情况，这是非常普遍的。

本节的目标是提出一个非常广泛的、可信的、典型的路线图，可以作为你的组织模板。你的旅程将有所不同，因为没有，而且永远不会有一刀切的实施路线图。本书中讨论的一些非常重要的方案将不适用于此。本节将作为一个案例研究推荐给你，该案例讲述了如何从业务驱动因素和当前状态开始，为你的组织选择合适的 DevOps 方案集合，并开发类似的路线图，用以达到你想要的最终状态。

A.1　组织背景

这里提到的组织，其假想名为大型银行和金融集团(以下简称为 MBFG，Massive Bank and Finance Group)，是一家传统的银行和金融服

务的跨国金融服务机构：

- 零售银行(分行和数字化)
- 商业银行
- 私人银行
- 证券交易

像大多数金融服务机构在房地产泡沫崩溃后的情况一样，MBFG 很快通过收购和兼并成长。因为其跨国公司的属性，它们面临着来自不同市场的各种不断变化的监管要求。他们面对需要通过创新来开发市场的挑战，从而触及发达市场中没有银行账户的千禧一代以及发展市场中没有银行账户的非千禧一代。首先，他们面临的金融技术的严峻挑战是(FinTech)初创公司提供银行服务(或与银行类似的服务)给没有银行账户的客户，不需要成为一家银行，因此，不会被银行牌照带来的监管所约束。他们需要降低传统分支银行的成本，并大力投资建立数字业务。是的，他们必须遵守不断调整的监管环境，从银行和信托法律法规的变化，到与英国退欧类似的地缘政治事件。这些不断变化和约束驱动着对敏捷性、速度和创新的需求，同时，MBFG 一直保持高质量和管理低成本，就像在 21 世纪的其他组织一样。

A.2 路线图组成

当详细描述第 3 章创建 DevOps 变革方案的过程，或者说实施路线图，需要三个核心成分：

- 具有目标状态的明确定义(业务目标和驱动)。
- 了解当前状态(当前的能力和成熟度)。
- 确定采取的最佳路径，或是实施方案(风险价值投资平衡)。

让我们回顾一下创建一个 DevOps 实施路线图的过程：

- 目标状态是由该组织的业务驱动因素和目标的理解和记录而确定。业务要求 IT 提供的服务是什么，如果没有变革，IT 将无法交付这些服务吗？

- 当前状态是由组织的当前成熟度定义的，并且这些因素抑制其交付业务线所需的能力。通过确定交付流水线中的瓶颈或低效，由价值流映射实践来定义当前状态。
- 路线图是通过确定合适的方案发展而来的；这些都是 DevOps 的能力，这些能力被用于解决每个确定的瓶颈。当你选择方案时，需要考虑技术、组织和财务因素。我在第 3 章中详细讨论了这些因素。

我在下一节中介绍实施路线图，其中讲述了前文提到的所有领域。

A.2.1　DevOps 的优化与创新工作坊

以下路线图是为 MBFG 实施的一个 DevOps 优化和创新工作坊的输出结果。这是一个正式的半天的工作坊，由 IBM 举行，用以帮助组织开发一个 DevOps 实施路线图并启动他们的 DevOps 的转型之旅。实施路线图的结构是如下五部分：

(1) 业务驱动因素
(2) 方案或计划中的技术举措
(3) 价值流图结果
(4) 根本原因
(5) DevOps 能力(方案)

DevOps 的创新和优化工作坊由 5 个练习的输出成果组成，处理上面提到的五个领域。

IBM 为 MBFG 的数字业务部门举行工作坊，这是一个独立的业务线，提供零售银行和商业银行的业务，并提供网络和手机的业务。该部门通过引入新技术引领全球银行的变革，利用开发创新的应用和商业模式，扩大其全球足迹，并且获得了金融科技(FinTech)。他们就是理想的 DevOps 实施候选人，不仅在他们主要经营的边缘创新[①]方面，而且还在于对所使用的核心业务进行优化。由于这些因素，他们需要

① 如果你浏览过前面内容，第 3 章介绍了边缘创新和优化核心业务。

优化和敏捷。

工作坊由该部门的首席技术官(CTO)发起。以下技术主管也出席了会议：

- 部门首席架构师
- 应用开发总监(包括 QA)
- 企业架构总监
- 数字运营总监
- 服务交付总监
- 首席运营官

IBM 出席工作坊的有作为促进者的 DevOps 的主题专家(Subject Matter Expert，SME)和与 MBFG 有交流经验的 IT 专家。工作坊总共花了大约六个小时，包括一小时的工作坊准备电话。在为期两周时间里，与 MBFG 联合制定出报告，然后提交给首席技术官和其他与会者。它被移交给 MBFG 作为他们现在拥有的路线图。

路线图目前正在执行中。IBM 与 MBFG 高层定期会议审查转型成果并在需要时更新路线图。

A.2.2 背景和上下文

为了更好地理解路线图，IBM 团队在交付工作坊之前就知道，重要的是了解 MBFG 的背景信息和特定的数字银行部门。出于节省时间的目的，这些信息为研讨会提供了上下文，不需要在研讨会中讨论。

1. 业务背景信息

在本节提供的业务和市场环境中，MBFG 正在考虑实施 DevOps。业务线的目标和限制是什么？

- 市场—— MBFG 总体情况和数字业务部门具体细节，是针对两个主要增长的市场：
  - 千禧一代需要的是银行服务，而不是使用传统的银行。市场中主要的竞争对手包括新金融科技，比如 Venmo 和 Square。
  - 不断壮大的无银行账户的中产阶级，尤其是在发展中世界的

增长市场。在世界各地的发展中国家中，MBFG 收购了一些银行，那里有着快速成长的中产阶级。在市场中，主要的竞争对手包括新金融科技，像 Novopay(Standard，2014)，在印度该公司通过角落里的门店提供银行业务。

- 生态系统中的合作伙伴——MBFG 正在通过发展合作伙伴生态系统寻找额外的收入来源。因此，为了实现这一目标，他们需要创建 API 经济。
- 监管问题——在监管环境中 MBFG 运营了不同的市场。需要符合所有监管要求，特别是新法规，如 PSD2(Payment Services Directive 2，支付服务指令二)，在欧洲，IT 资源被严重消耗。
- 安全的担忧——“我不想我的名字出现在华尔街日报，”这是 MBFG 首席信息安全官(CISO)描述的他们的安全目标。最近大型金融机构违规行为导致了对安全的更多投资，从而导致了其他地方的投资减少。
- 服务中断—— MBFG 集团中的一个银行发生了一次严重的服务中断事件。MBFG 无法重现中断来确定问题的根本原因。很显然最新的企业架构的缺失是无法重现服务中断的原因。因为服务中断期间客户无法访问他们的账户，监管机构正在研究这种故障，甚至无法理解服务中断的发生方式和原因。MBFG 希望确保任何变更被良好地组织，所有架构的变化都有据可查，使得可以对目前的企业架构进行维护。

2. IT 的背景信息

在本节提供的 IT 部门背景信息中，MBFG 正在考虑实施 DevOps。IT 部门的目标和限制是什么？

- 混合系统—— MBFG 通过收购和兼并已经长成为大企业，也已经存在了几十年。因此他们有大量的遗留技术栈，从大型机到 IBM System i，再到 Unisys 系统，以及运行在老系统中已经淘汰版本中的间件，这些都需要进行升级。
- 影子 IT——影子 IT 泛滥，尤其是数字银行部门。多条业务线产

生了许多商业应用，这些应用运行在像 AWS(亚马逊 Web 服务)、Rackspace 和 SoftLayer 这样的公有云上面。他们雇用了行业领先的公司来构建这些应用程序。

- 数据驻留——数据的存储位置是个大问题。一些国家要求将客户的数据存储在本地。由于收购和兼并的原因，现在 MBFG 的国家定制版本的应用程序和系统运行在本地数据中心，由他们自己或由外部供应商管理。
- 没有云发展策略——与其他的公司相比，MBFG CIO 向董事会和新闻界声明，“我们是云优先”战略。但在现实中，没有全面的云战略。多云厂商的影子 IT 泛滥。实验是由几个组织在 OpenStack、Docker 和 Cloud Foundry 中进行。CIO 仍然对使用公共云犹豫不决，没有官方立场。

A.3 实施路线图

本章余下部分的内容记录了在 MBFG 实施整体发展路线图。当然，一些直接的参考和具体细节已经模糊化、虚构化，用来泛化该路线图。

A.3.1 业务驱动因素

以下业务驱动因素是在工作坊中为 MBFG 数字银行部门制定的：

- 为了做得更多，进行成本管理。
  - 这里的目标不是为了降低成本，而是提高生产率，以便能够在现有预算结构内交付更多的价值。
- 增加频度和开发新的应用吞吐量，以及对现有应用程序的增强。
  - 目标是既要快速实现创新，又要以更快的速度完善现有系统。
- 采取更多的产品化的和以客户为中心的交付方法。
  - 目标是能够获取客户的反馈，并基于这些反馈开发和增强应用。
- 同时提高产品和软件的质量。
  - 目标是减少停机时间和用户可感知的服务中断。

- 调解监管所带来的压力，减少或消除变更窗口期。
  - MBFG，像许多 FSS(Financial Software & Systems，金融软件系统)的组织一样，需要在多个市场中的监管审查的情况下减少计划变更窗口期的停机时间，而在这个过程中客户可能无法访问账户的某些功能。

A.3.2　现有的 IT 举措

MBFG 积极应对面临的挑战，即使在本次工作坊之前也是如此。多个措施正在筹划或已经在计划中。下面列出这些举措，并已纳入最终的实施路线图。

- 环境供给自动化
  - 这一举措特指可能部署的全栈环境。MBFG 目前正在探索云环境供给和编排工具。
- 发布自动化
  - 指的是应用软件部署到生产环境(甚至开发/测试环境)基础设施的自动化。MBFG 一直采用 IBM UrbanCode 工具套件来辅助应对这一挑战。
- 采用 CI(Continuous Integration，持续集成)工具
  - 在 MBFG 组织中，有几个 CI 工具处于不同的实施阶段。目标是规范化最小的一组工具(一个或多个)，可以在 MBFG 组织中创建一套标准，使任何团队都可以接受 CI。
- 实施服务虚拟化
  - MBFG 已经采用了服务虚拟化工具，在数字银行部门，正在迅速扩大其使用范围。
- QA 转型
  - 对 QA 的工作内容进行评估，用来实现自动化测试以及在生命周期的前期进行集成测试(测试迁移[①])。
- 简化应用程序架构

---

① 前移，精益原则，曾经在第 1 章中介绍过。

    - 在数字业务部门具体的挑战是现有的代码库的复杂性。他们目前正在研究如何简化这一点，将更多的模块化引入到设计中。这与企业级架构的工作是分开的。
- 减少架构分层
    - 数字银行部门的进一步挑战是明显过度分层的架构，其中的重复和冗余层可以合并。他们目前正在通过实验实现这一目标，以及了解相关的收益。
- 对设计思维进行试点
    - 数字业务部门已经参加了一系列的工作坊，目的是了解他们如何使用设计思维降低高度官僚化的需求获取和文档的流程。他们已经识别了现有进程相关的重大挑战，这些流程需要大约 34 个不同委员会的批准。最初的目标已经确定，那就是两个星期产出一个 MVP(最低可行产品)。
- 从 DevOps 工具中抽取度量指标
    - 当前的工作报告需要从多个来源提取度量指标，这些数据源来自商业的和自制的软件。大多数向管理提供的报告需要许多天才能完成，在大多数情况下数据都会过时。MBFG 的数字业务部门试图采用 Hygieia(美国一家名为 Capital One 银行开源的项目)，一个开源的 DevOps 仪表盘。

A.3.3 瓶颈

在工作坊实施过程中，价值流图实践识别出了 MBFG 的数字银行部门内的在交付流水线中的瓶颈。工作坊的与会者选择了一个典型的 Java 开发的流水线，并且讨论了价值流图练习的目标。

- 需求管理和设计：
    - 定义阶段成本过高。通常，在定义阶段所产生的巨大成本，使得项目变得非常不灵活。
    - 需求的获取和设计过程中所产出的结果，仅仅包括纸质文件，而不是可以反复修改的构建物和模型。文档很长(通常有 100 多页)，而且通常不会被打算使用它们的团队阅读或

完全遵守。

    - 顶层设计分出不同的平台分支，当付诸行动时，在独立于顶层设计的情况下，每个团队产生一个不同的设计。这导致从高层到低层设计缺失可追溯性。
- 项目治理与管理：
    - 项目管理往往被视为采用观察和报告的态度，而不是积极驱动项目。
    - 项目资金有时是不一致的或是间歇性的，导致一开始就中断了，直接影响项目的持续性。例如，当资金枯竭时，导致人员被释放；然后，当资金有保障时，分配到项目的成员与释放前不同。
    - 治理使得技术决策时间延长。
    - 流程告诉人们做什么，却不是该怎么做。
- 项目团队：
    - 整个组织的理念是把工作分配给合适团队，而不是驱动团队去工作。这导致了领域知识丢失，因为项目团队中的人可能以前从未在该领域工作过，但由于他们当前可用而被分配。
    - 多功能团队(cross-functional teams)使用一个基于工单的系统进行沟通，使得交流过于缓慢和烦琐，导致长时间等待。
- 代码和架构：
    - 目前的代码管理效率很低。从多个项目的代码合到业务发布代码流，再到部署需要 30 天时间。
    - 代码没有任何模块或内部架构。这种应用程序架构的不足导致了大量的低效率和技术债务。
- 应用服务器配置管理：
    - 应用服务器的配置是通过一个大文件包含多个应用程序的大量参数配置信息进行管理的；这导致了配置管理约束，使得配置管理过程很容易出错。
    - 为了节约成本，早期的测试使用 Jetty 的应用服务器，尽管生产环境的部署目标为 WebSphere 应用服务器(WAS)。这需

要每个项目在发布周期中投入一个开发周期(迭代)，以确保应用程序在部署到生产之前可以在 WAS 上运行。

- 部署和发布管理：
  - 部署工作是手工进行的，而且非常昂贵和耗时。大约需要 35 至 40 名全职员工从事部署开发/测试/生产环境的工作。
  - 为业务发布的批量非常大；在一个发布版本中最多有 20 个项目，所有这些项目需要相互协调以及同步发布时间。
  - 多个项目在业务发布的设计阶段存在相互锁定，然后是在系统测试阶段互锁，但两者之间没有关系；这样的结果导致识别几个集成挑战太晚了。

A.3.4　根因分析

前一节列举的瓶颈确定了以下根因。通过检查每个瓶颈，发现这些根本原因。以下是在检查依赖项和重复项之后开发出的综合列表。

- 缺乏统一的应用程序体系结构的演进，而不是许多独立的设计。这样的模型可以在整个生命周期中演化。
- 具有单一的、一刀切的治理过程，因为该过程太烦琐了，因此产生了使项目绕过现有治理的欲望。
- 缺乏一个基于产品的团队，使其具有产品的整个生命周期的所有权。目前的所有权从一个团队交接到另外一个项目团队，导致缺乏端到端的组件和应用的生命周期技术所有权。
- 缺乏自动化，特别是发布和部署自动化。
- 缺乏良好的应用架构和配置管理实践，尤其是基于微服务架构和松散耦合的服务和组件。
- 集成测试在交付生命周期中进行得太晚了。
- 从关注过程到关注产品的方法。

A.3.5　DevOps 实践

下面的 DevOps 能力推荐给 MBFG 数字银行部门，用来解决交付流水线中的瓶颈，使它们走上自己的 DevOps 的转型之旅。

1. 自动化

目标

介绍以下四个方面的自动化工具：

- 部署
- 环境的建立和维护
- 测试
- 发布

自动化提供了使流程可重复、稳定和伸缩的能力。

商业和技术收益

在工作坊期间，识别出了几个可以通过引入自动化解决的低效率的领域。这些低效率会导致浪费和增加等待时间，从而对业务造成重大影响：

- 跨职能团队使用低效的工单系统进行交流。
- 手动测试需要许多天才能完成。
- 回归测试是自动化的，但仍需要许多天才能完成，并且在交付生命周期的后期才实施。
- 部署完整版本由手动的和具有明显等待时间的批次进行，产生大量费用。
- 开发-测试-生产环境不使用相同的中间件和配置。

以上推荐列为最先被采纳的建议，因为它们有着最快的投资回报速度。

DevOps 实施建议

IBM 建议 MBFG 数字银行部门从以下四个方面采用自动化工具解决低效问题，低效问题造成浪费，最终会影响业务。

(1) 实现单一的协作工具，让整个生命周期的所有干系人计划、协作和共享工作项。这将通过更换目前使用的工单系统解决所有协作问题。

(2) 采用部署自动化，允许自动化和可靠地将应用程序组件、中间件代码和配置，部署到交付流水线中的任何环境。

(3) 继续采用和推出测试虚拟化，用以提供应用组件的持续测试能力。在交付生命周期的所有阶段引入持续测试，用来实现测试前移，这将导致问题和架构缺陷尽早识别，并提高整体质量。

(4) 采用发布管理工具，提供协调发布、资源可用性和环境可用性的能力。这确保了合适的发布队列和其组件在流经交付流水线，并且在其各自的流中进行集成。

至关重要的是，正确的过程是自动化的，并进行优化，以达到最大的效率。因此，IBM 建议由主题专家(SME)与 MBFG 数字银行部门检查在所有识别出的四个领域的自动化，并在这些领域从事过程改进。引入自动化工具还提供了探索在手动执行时不可行的、高效组合流程的能力。

2. 减少交付批量

目标

尽可能使交付管道高效，以减少每次迭代的周期时间。减少每次交付物的批量大小，有助于实现这一目标。

商业和技术收益

通过交付较小批量的应用功能变更实现了快速迭代，从而提高吞吐量、降低风险和提高质量。利用自动化工具以及交付流水线中的架构迭代，使交付流水线每次都只产生少量的功能变更，其结果是：交付流水线间的部署更为频繁、对代码及配置的小幅变化所进行的测试更为频繁、组件的集成也更为频繁。从而实现了持续集成、持续交付以及持续测试这类 DevOps 功能。较小的批量也减少了发布计划的挑战，减少了交付流水线可用资源的竞争，并减少了发布周期时间从而减少资源锁定时间。

DevOps 实施建议

(1) 采取主动减少交货批量。这不是一项微不足道的任务，因为它需要重构应用程序和数据结构，以确保应用程序是由较小的、松耦合的组件组成的，这些组件可以独立部署和测试。在应用层，采用基于微服务的架构将是解决这一建议的理想方法。

(2) 使用测试虚拟化使小组件可以更频繁地测试，而不需要等待其

他相关组件是否可用。此外，部署自动化使单个和复合组件以及应用程序在更高频率上频繁且连续地部署。

MBFG 数字银行部门的需要得到架构方面的指导来帮助他们重新设计和重构，使得应用程序适应微服务架构。

(3) 建立产品团队

目标

建立永久性产品管理团队，具备应用产品在 MBFG 数字银行部门之间持续拥有所有权。

商业和技术收益

有了产品管理团队，拥有程序和技术的所有权，为交付应用程序产品增强弹性，同时秉承了 MBFG 数字银行部门的核心治理要求。这些团队是永久的，并保留产品的所有权，其所有权超越了单独项目和跨企业范围。他们召集必要的主题专家和技术人员团队，交付产品的单独项目和总体能力。他们变成带来工作的团队，而不是临时的团队，被集合起来处理工作的单元。

DevOps 实施建议

(1) 确定一套最小的产品，应该有自己的永久性产品管理团队。这些团队应该拥有产品的架构、技术设计和长期愿景，并与业务线协同工作，以满足业务需求。

(2) 开发管理流程和标准来支持产品管理团队。

(3) 向产品管理团队提供合适的技能资源，以确保他们的长期发展和成功，包括产品所有者、解决方案架构师和必要的管理人员。

(4) 使产品管理团队接受设计思维的培训，基于角色期望的用户体验，完成设计的应用路线图、需求和能力。

(5) 确保持续的资金模式，确保这些产品团队的连续性和创造。

这些建议是与组织的、流程的和管理的变革相关的，需要由 MBFG 数字银行部门的高级领导发起和负责。IBM 与 MBFG 共享自己的产品管理团队的经验，解释他们是如何在 IBM 内组织以及运营应用交付组织。

3. 应用架构设计

目标

介绍了应用程序架构的建模能力，以及在 MBFG 数字银行部门如何获得应用程序和系统的架构以及设计，以便为他们在应用交付生命周期对应用程序进行演进。

商业和技术收益

拥有一组通用的模型，用来代表所有应用程序和系统的代码以及架构，对其提供统一视图是必不可少的，而不是使用目前的多个设计方案。保持一组通用的模型使其获得重用能力，以及重构和更好的架构能力。

DevOps 实施建议

发起一项主动行动，获取跨多个组件、应用程序和系统的代码，作为记录良好的应用程序架构文档。每个交付团队中的解决方案架构师在创建这些架构时拥有其所有权，并对应用程序及对其他应用程序和服务的依赖性保持所有权。这些解决方案架构师还应该负责确定架构的重用和重构的机会以及更新企业架构。

IBM 可以为 MBFG 数字银行部门提供工具和架构指导，包括模式和架构管理的最佳实践以及实现工具使用的技术指导。

4. 自助服务开发测试环境提供

目标

为开发和测试人员介绍自助服务门户，使他们能够获得与生产环境类似的环境，并配置他们，以及通过一个按钮将正在开发或测试的应用程序部署到提供的环境中。

商业和技术收益

开发和测试从业者的效率和生产力的最大障碍之一是缺乏所需的类生产环境。对从业人员提供一个自助服务门户网站，包括预设置全栈的模式，可以自动配置，要求不需要运维团队的人工干预，以上能力能显著提高从业者的生产力。确保这些环境是类似于生产环境，使它们在

拓扑结构、配置以及行为上都类似于生产环境，可以显著提高正在开发和测试的产品的质量。

DevOps 实施建议

(1) 开始创建基于云的自助门户。这个门户模式可用，在云上的开发测试环境的配置将由运营团队管理。开发测试团队将在需要时提交一个新环境模式的请求。

(2) 利用基于 OpenStack 的云，使用云管理工具监督模式设计、模式门户网站和目录管理，环境供应使用 OpenStack Heat，用来编排所提供的环境。

A.3.6 实施路线图

本附录中展示的实施路线图可能显得极其艰巨，而且任何一个组织都需要进行大量的工作。重要的是，这样的路线图作为指导进行大规模的组织整体变革。像 MBFG 这样的组织预计不会承担这种“大爆炸”转型的方法，但可以通过一系列的试点项目，独立采用小的建议，此方法在前面的章节中描述过。只有在这些试点项目成功完成并收获经验教训之后，整个组织才能扩大规模。这些试点项目将需要错开分布，以保证时间、资金和主题专家所需资源的投资，并允许合适的项目处于合适的阶段，以便在开发和交付应用程序方面进行重大改变。因此，这是多年的转型路线图。

此外，在与 MBFG 共同管理和优化路线图的工作中，其目的是让他们决定哪些建议需要最先解决。所需的投资，以及每个投资的预期回报，是优先考虑工作的一个主要考虑因素，也是根据业务目标和市场机遇确定的某些能力的价值。

该路线图应被视为一个大型的 DevOps 整个组织范围的实施路线图。基于你的需求和目标路线会有所不同，希望不会那么复杂。

参考文献

前言

Sharma, S. (2016, May 15). *The Ultimate Winning Play?* Retrieved from sportsthrills.wordpress.com: https://sportsthrills.ordpress.om/2016/05/15/the-ultimate-winning-play/

Wikipedia. (2016, September). *DevOps*. Retrieved from wikipedia.org: https://en.wikipedia.org/wiki/DevOps

第 1 章

Bias, R. (2012, February 16). *Architectures for open and scalable clouds (slide* 20). Retrieved from slideshare.net: http://www.lideshare.net/randybias/architectures-for-open-and-scalable-clouds

Caum, C. (2013, August 28). @ccaum. Retrieved from Twitter: https://twitter.com/ccaum/status/372620989257232384

Deming, D. W. (1998, April 27). *Dr. Deming's Management Training*. Retrieved from Dr. Deming's Management Training: http://www.dharma-haven.org/five-havens/deming.htm

Forrester. (2011, July 26). *Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today*. Retrieved from Forrester.com: https://www. orrester.com/report/WaterScrumFall

+Is+The+Reality+Of+Agile+For+Most+Organizations+Today/-/E-RES60109

Garvin, David A., Amy C. Edmondson, and F.G. (2008, March). *Is yours a Learning Organization*? Retrieved from hbr.org: https://hbr.org/2008/03/is-yours-a-learning-organization

Kaz, P. (2013, July 4). *Measurement Myopia*. Retrieved from Drucker Institute: http://www.druckerinstitute.com/2013/07/measurement-myopia/

Kruchten, P. (2002, August 6). *The 4+1 View Model of architecture*. Retrieved from ieee.org: http://ieeexplore.ieee.org/xpl/articleDetails.jsp? arnumber =469759

Lean.org. (2016). *Lean.org*. Retrieved from Lean.org: http://www.lean.org/WhatsLean/History.cfm

McCance, G. (2012, November 19). *CERN Data Centre Evolution* (*Slide 17*). Retrieved from Slideshare.net: http://www.slideshare.net/gmccance/ cern-data-centre-evolution

Rice, R. (2009). *Achieving Software Quality Using Defect Filters*. Retrieved from Randy Rice's Software Testing Site: http://www.riceconsulting.com/articles/achieving-software-quality-using-defect-filters.htm

第 2 章

Cantor, M. (2014, July 15). *Flow measurements for software*. Retrieved from IBM DeveloperWorks: http://www.ibm.com/developerworks/library/d- flow-measurements-sw/index.html

Hartman, B. (2009, November 11). *An Introduction to Planning Poker*. Retrieved from dzone.com:https://dzone.com/ articles/introduction- planning-poker

Martin, K. (2011, August 11). *Value Stream Mapping in Non-Manufacturing Environments*. Retrieved from Slideshare.net:

http://www.slideshare.
net/KarenMartinGroup/value-stream-mapping-in-nonmanu
facturing-environments/24-Key_Metrics_QualityComplete
_and_Accurate

Ohno, T. (2006, March). *Ask 'why' five times about every matter.* Retrieved from toyota-global.com: http://www.toyota-global.com/company/toyota_traditions/quality/mar_apr_2006.html

Poppendieck, M. (2008, October). *Value Stream Mapping.* Retrieved from agiles.org: http://agiles2008.agiles.org/common/pdfs/Poppendieck%20-%20 Value%20Stream%20Mapping.pdf

Wagner, B. (2009, January 9). *Notes on Value Stream Mapping.* Retrieved from thebillwagter.com: http://thebillwagner.com/Blog/Item/2009-01-07-NotesfromPoppendieckValue-StreamMapping

Wikipedia. (n.d.). *Supply Chain.* Retrieved September 16, 2016, from Wikipedia: https://en.wikipedia.org/wiki/Supply_chain

Wikipedia. (n.d.). *Value Stream Mapping.* Retrieved July 10, 2016, from Value Stream Mapping: https://en.wikipedia.org/wiki/Value_stream_mapping

第 3 章

Forrester. (2013, July). *Forrester Research Total Economic Impact Study on Service Virtualization and Test Automation Solutions.* Retrieved from ibm.com: https://www.ibm.com/developerworks/community/blogs/rqtm/entry/forrester_research_total_economic_impact_study_on_service_virtualization_and_test_automation_solutions?lang=en

Forrester. (2015,August). *The Total Economic Impact Of IBM UrbanCode.* Retrieved from ibm.com: https://developer.ibm.

com/urbancode/docs/the-total-economic-impact-of-ibm-urbancode/

Osterwalder, Y. P. (2013). *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers.* John Wiley and Sons. Strategyzer. (2013). *Business Model Canvas*. Retrieved fromwww.businessmodelgeneration.com:http://www. businessmodelgeneration.com/canvas/bmc

Vaccaro, A. (2014, Janurary 8). *An Unlikely Case Study in Fast Growth: Major League Soccer*. Retrieved from Inc.com: http://www.inc.com/adam-vaccaro/mls-commissioner-don-garber.html

第 4 章

ASUM, I. (2016, March 1). *Analytics Solutions Unified Method.* Retrieved from IBM Analytics Services: http://public.dhe.ibm.com/software/data/sw-library/services/ASUM.pdf

Bradbury, J. (2 011, September 26). *A Sports Economist's Thoughts on Moneyball: A Guest Post by J.C. Bradbury*. Retrieved from Freakonomics:http://freakonomics.com/2011/09/26/a-sports-economists-thoughts-on-moneyball-a-guest-post-by-j-c-bradbury/

Burke, T. (2016). *This Is Why There Are So Many Ties In Swimming.* Retrieved from deadspin.com: http://regressing.deadspin.com/this-is-why-there-are-so-many-ties-in-swimming-1785234795

Capital One GitHub. (2015, July 18). *CapitalOne DevOps Dashboard.* Retrieved from Github.com: https://github.com/capitalone/Hygieia

Cheshire, J. (2012, January 9). *Cricket: The Top 10 All-Rounders of All Time*. Retrieved from bleacherreport.com: http://

bleacherreport.com/articles/1017237-the-top-10-all-rounders-of-all-time

Cockcroft, A. (2012, March 19). *Ops, DevOps and PaaS (NoOps) at Netflix*. Retrieved from Adrian Cockcroft's blog:http://perfcap.blogspot.com/2012/03/ops-devops-and-noops-at-netflix.html

Conway, M. E. (1967). *Conway's Law.* Retrieved from www.melconway.com: http://www.melconway.com/Home/Conways_Law.html

Donato, C. (2016, July 14). *Big Data Analytics are Enhancing Women's Tennis*. Retrieved from ZDNet.com: http://www.zdnet.com/article/big-data-analytics-are-enhancing-womens-tennis/

Harrell, E. (2015, October 30). *How 1% Performance Improvements Led to Olympic Gold.* Retrieved from Harvard Business Review: https://hbr.org/2015/10/how-1-performance-improvements-led-to-olympic-gold

Higdon, H. (2011). *Marathon Training Guide - Introduction.* Retrieved from halhigdon.com: http://halhigdon.com/training/51135/Marathon-Training-Guide/

Hodges, R. (2015). *IT Service Management for DevOps.* Retrieved from IBM DeveloperWorks: https://www.ibm.com/developerworks/community/files/form/anonymous/api/library/42529e82-173a-4f45-805b-93d9eb35ffa6/document/19b71c8c-1675-4727-a3ab-b259ba1d49e6/media/ITSM%20Reference%20Architecture%20-%20DevOps%20-%20Whitepaper.pdf

Judge, L. (2016). *Should power hitters bunt against defensive shifts?* Retrieved from The Kansas City Star: http://www.kansascity.com/sports/spt-columns-blogs/judging-the-royals/article78348842.html#storylink=cpy

Kagan, S. a. (2015). Transforming Application Delivery: Executive Advisory Session. *IBM InterConnect 2015*. IBM.

Kim, G. (2013). *The Three Ways: The Principles Underpinning DevOps*. Retrieved from IT revolution: http://itrevolution.com/the-three-ways-principles-underpinning-devops/

Kniberg, H. (2 014, March 27). *Spotify engineering culture*. Retrieved from Spotify Labs: https://labs.spotify.com/2014/03/27/spotify-engineering -culture-part-1/

LinkedIn, Engineering Blog. (2011, October 26). *Continuous Integration for Mobile*. Retrieved from LinkedIn Engineering Blog: https://engineering.linkedin.com/testing/continuous-integration-mobile?cm_mc_uid=92132403740114635134753&cm_mc_sid_50200000=

Medeiros, J. (2014). *The winning formula: data analytics has become the latest tool keeping football teams one step ahead*. Retrieved from wired.co.uk: http://www.wired.co.uk/article/the-winning-formula

Moran, S. S. (2013, September 6). *Test Data Management in the DevOps Lifecycle*. Retrieved from The Invisible Thread: https://www.ibm.com/developerworks/community/blogs/invisiblethread/entry/test_data_management_in_the_devops_lifecycle?lang=en

Pal, T. (2015). *Hygieia Dashboard - Making sense out of your DevOps tools*. Retrieved from capitalone.io: http://www.capitalone.io/blog/hygieia-making-sense-out-of-your-devops-tools/

Pereira, R. (2009). *The Seven Wastes*. Retrieved from iSixSigma Magazine: http://blog.gembaacademy.com/wp-content/uploads/2009/09/7_wastes_isixsigma_magazine_0909.pdf

Peter, U. (2008, August 20). *Tips from a Top Sports Team Coach.* Retrieved from infoq.com: https://www.infoq.com/articles/sport-coaching-and-agile

Pollock, R. (2013, October 17). *Troubled Obamacare website wasn't tested until a week before launch.* Retrieved from Washington Examiner: http://www.washingtonexaminer.com/troubled-obamacare-website-wasnt-tested-until-a-week-before-launch/article/2537381

Powers, J. (2013). *Why pulling the goalie is often worth the risk.* Retrieved from bostonglobe.com: https://www.bostonglobe.com/sports/2013/05/29/pulling-goaltender-may-risky-move-but-bruins-and-other-hockey-teams-have-made-pay-off/u1dbL9XrfejggKHUnxpPUI/story.html

Quirk, T. P. (2004, January 5). *How to quantify downtime.* Retrieved from networkworld.com: http://www.networkworld.com/article/2329877/infrastructure-management/how-to-quantify-downtime.html

Radcliffe, S. S. (2014, July). *Whitepaper: Best practices for a DevOps approach with IBM System z.* Retrieved from IBM Software: https://www14.software.ibm.com/webapp/iwm/web/signup do? source=swg-rtl-sd-wp&S_PKG=ov26345

Reinertsen, D. G. (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development.* Celeritas.

Roberts, A. (2009, May 14). *How To Build a Championship Basketball Team.* Retrieved from Bleacher Report: http://bleacherreport.com/articles/174989-how-to-build-a-championship-basketball-team

Rosenbaum, M. (n.d.). *Strategies for the 4 x 100 Relay Race.* Retrieved from About.com: http://trackandfield.about.com/od/sprintsandrelays/a/400relaystrat.htm

Sun, J. L. (2013, May 7). *Don't Believe the Myth-information about the Mainframe*. Retrieved from share.org:http://www. share.org/p/bl/et/blogid= 2&blogaid=234

Surowiecki, J. (2014). *Better All the Time.* Retrieved from newyorker.com: http://www.newyorker.com/magazine/2014/11/10/better-time

Taylor, B. (2014, April 14).*Why Amazon Is Copying Zappos and Paying Employees to Quit*. Retrieved from Harvard Business Review: https://hbr.org/2014/04/ why-amazon-is-copying-zappos-and-paying-employees-to-quit/

UML. (2005). uml.org. Retrieved from What is UML?: http://www.uml.org/what-is-uml.htm

Ward, S. C. (2012, July 23). *The Fastest Baton to the Finish Line.* Retrieved from nytimes.com: http://www.nytimes.com/interactive/2012/07/23/sports/olympics/the-fastest-baton-to-the-finish-line.html?_r=0

Wikipedia. (n.d.). *Daily Standup Meeting.* Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Stand-up_meeting

Williamson, S. S. (2014, May 13). *DevOps for mobile apps challenges and best practices*. Retrieved from IBM DeveloperWorks: https://www.ibm.com/developerworks/library/mo-bestdevops-mobileapps/

第 5 章

Blue Label Labs. (2016, April 19). *MVPs: Why a Minimum Viable Product Is Your Most Valuable Player.* Retrieved from IdeaToAppster: https://www.bluelabellabs.com/ideatoappster/mvps-why-a-minimum-viable-product-is-your-most-valuable-player/

Bowen, M. (2015, May 21). *What It Takes to Be a Special Teams Demon*. Retrieved from Bleacher Report: http://bleacherreport.com/articles/2470321-what-it-takes-to-be-a-special-teams-demon

Brown, C. (2013, January 25). *Speak My Language*. Retrieved from grantland.com: http://grantland.com/features/how-terminology-erhardt-perkins-system-helped-maintain-dominance-tom-brady-patriots/

Brown, K. (2016). Microservices Architectures for Cloud. *IBM InterConnect2016*. IBM.

Brown, K. (2016, April 13). *Refactoring to microservices, Part 1 and 2*. Retrieved from IBM DeveloperWorks: https://www.ibm.com/developerworks/cloud/library/cl-refactor-microservices-bluemix-trs-1/index.html

CattleTags.com. (2016). *Year Letter Designations for Beef Cattle Numbering*. Retrieved from CattleTags.com: http://www.cattletags.com/ beef-cattle-year-letter-designations

Cavin, J. W. (2014, December 18). *Seattle Sounders Sports Science Weekend: Building the Anti-fragile Athlete: Dave Tenney*. Retrieved y from jameswcavin.com: http://www.jameswcavin.com/clinicians-and-coaches/2014/12/18/yiux6yv6ro6feual4j5qelwremc77i

Chase, C. (2015, September 9). *What is Roger Federer's new SABR move*? Retrieved from usatoday.com:http://ftw.usatoday.com/2015/09/ what-is-roger-federers-new-sabr-move

Chiara Brandle, e. a. (2014, January 7). *Cloud Computing Patterns of Expertise*. Retrieved from IBm Redpaper: http://www.redbooks.ibm.com/abstracts/redp5040.html?Open

Cloud Native Computing Foundation. (2015, November 6). *CNCF Charter*. Retrieved from CNCF.io: https://cncf.io/about/charter

Edelman, D. M. (2015, October). *How to scale your own digital disruption*. Retrieved from McKinsey.com: http://www.mckinsey.com/business-functions/operations/our-insights/how-to-scale-your-owndigital-disruption

ESPN Developer Center. (2015). *ESPN API Explorer*. Retrieved from ESPN Developer Center: http://www.espn.com/static/apis/devcenter/io-docs.html

Farrell, A. (2 008, August 20). *America's Top 10 Olympic Schools*. Retrieved from Forbes: http://www.forbes.com/2008/08/20/olympics-colleges-phelps-biz-sports_cx_af_0820olympics.html

Farres, L. G. (2004, Fall). *Becoming a Better Coach through Reflective Practice*. Retrieved from mindinmotion.ca:http://www.mindinmotion.ca/articles/lgfarres_reflective_practice.pdf

Feinberg, R. (2011, Summer). *Baseball's Newest Farm System*. Retrieved from Americas Quarterly:http://www.americasquarterly.org/node/2752

Fordacell, M. B. (2006, November). *Real Madrid football club: A new model of business organization for sports clubs in Spain*. Retrieved from researchgate. net: https://www.researchgate.net/publication/229478080_Real_Madrid_football_club_A_new_model_of_business_organization_for_ sports_clubs_in_Spain

Fowler, J. L. (2014, March 25). *Microservices*. Retrieved from martinfowler.com: http://martinfowler.com/articles/microservices.html

Frederick, S. (2016, February 10). *Which cloud? Any cloud: Getting started with portable HOT documents*. Retrieved from Freddy on Cloudy Stuff: https://sudhakarf.wordpress.com/2016/02/10/which-cloud-any-cloud-designing-portable-hot-documents-with-urbancode/

He, L. (2013, March 29). *Google's Secrets Of Innovation: Empowering Its Employees*. Retrieved from Forbes: http://www.forbes.com/sites/laurahe/2013/03/29/googles-secrets-of-innovation-empowering-its-employees/

IBM Design. (2016). *IBM Design Thinking Field Guide*. IBM Design.

IBM Institute for Business Value. (2016). *Digital reinvention in action*. Retrieved from ibm.com: https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=GBE03752USEN

Imgur, U. a. (2013, January 28). *Redbox used to be for more than movies. Found this in DC in 2004*. Retrieved from imgur.com: http://imgur.com/HmQNCwj

Jenkins, M. (1 998). *Overtraining Syndrome*. Retrieved from rice.edu: http://www.rice.edu/~jenky/sports/overtraining.html

Joe Loewengruber, K. G. (2016). *High Availability*. Retrieved from IBM Garage Method: https://www.ibm.com/devops/method/content/manage/practice_high_availability/

Kalavalapalli, Y. (2016, April 8). *Sony looks to extend IPL broadcasting rights*. Retrieved from Live Mint: http://www.livemint.com/Companies/NXFJ1Ycguzq1a4vxuv1ruL/Sony-looks-to-extend-IPL-broadcasting-rights.html

Koffel, W. (2014, January 4). *12-Factor Apps in Plain English*. Retrieved from ClearlyTech: http://www.clearlytech.com/2014/01/04/12-factor-apps-plain-english/

Li, C.-S. (2014, March). *Software defined environments: An introduction*. Retrieved from Researchgate: https://www.researchgate.net/publication/261718883_Software_defined_environments_An_introduction

Lopez, A. (2014, May 14). *The ESPN Effect – The Evolution of our Sports Consumption*. Retrieved from page2sports:http://

page2sports.com/6038/espn-effect-evolution-sports-consumption-2

Martin, M. (2014, August 12). *FastCustomer can help get you out of customer service hell*. Retrieved from engadget.com: https://www.engadget.com/2014/12/08/fastcustomer-can-help-get-you-out-of-customer-service-hell/

Martinez, C. (2016, August 25). *2016 Rio Olympics: Current NCAA studentathletes competing by school*. Retrieved from NCAA: http://www.ncaa.com/news/ncaa/article/2016-07-28/2016-rio-olympics-ncaa-olympic-student-athletes-school

NIST, U.S. Department of Commerce. (2011, September). *The NIST definition of Cloud Computing*. Retrieved from nist. gov: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

Olenski, S. (2015, February 20). *How Pro Sports Teams Are Taking The Customer Experience To The Next Level*. Retrieved from Forbes.com: http://www.forbes.com/sites/steveolenski/2015/02/20/how-pro-sports-teams-are-taking-the-customer-experience-to-the-next-level/2/#4c384afe1c31

OpenStack.org. (2016, September 8). *Networking API v2.0.* Retrieved from OpenStack Documentation: http://developer. openstack.org/api-ref/networking/v2/?expanded=create-network-detail

Peranandam, C. (2012, September 10). *Orchestrating the cloud to simplify and accelerate service delivery*. Retrieved from IBM DeveloperWorks: https://www.ibm.com/developerworks/community/blogs/9e696bfa-94af-4f5a-ab50-c955cca76fd0/entry/orchestrating_the_cloud_to_simplify_and_accelerate_service_delivery1?lang=en

Ries, E. (2011). *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business.

Roettgers, J. (2008, May 3). *Whatever Happened to Red Swoosh?* Retrieved from GigaOm: https://gigaom.com/2008/05/03/whatever-happened-to-red-swoosh/

Spiewak, S. (2016, April 29). *Multi-sport Athletes Dominate First Round of 2016 NFL Draft*. Retrieved from USA Football: http://usafootball.com/blogs/americas-game/post/12009/multi-sport-athletes-dominate-first-round-of-2016-nfl-draft

Taleb, N. N. (2007). *The Black Swan: The Impact of the Highly Improbable*. Random House.

Taleb, N. N. (2 012). *Antifragile: Things That Gain from Disorder.* Random House. The Movie Network. (2014, February 26). *The Rental Revolution: How Redbox Revitalized the Movie Rental Industry*. Retrieved from Themovienetwork.com: http://www.themovienetwork.com/article/rental-revolution-how-redbox-revitalized-movie-rental-industry

Timmons, A. (2008, November 2). *The 10 Greatest Multi-Sport Athletes of All Time*. Retrieved from Bleacher Report: http://bleacherreport.com/articles/76583-the-10-greatest-multi-sport-athletes-of-all-time

Triplett, M. (2014, July 3). *Saints' top plays: 'Ambush' onside kick.* Retrieved from ESPN: http://www.espn.com/blog/nfcsouth/post/_/id/55736/saints-top-play-ambush-onside-kick

Tseitlin, Y. I. (2011, July 19). *The Netflix Simian Army.* Retrieved from The Netflix Tech Blog: http://techblog.netflix.com/2011/07/netflix-simian-army.html

Wilson, J. (2010, April 23). *Business Model Generation: Osterwalder rethinks the publishing industry*. Retrieved from MaRS: https://www.marsdd.com/news-and-insights/business-model-generation-osterwalder-rethinks-the-publishing-industry/

Wolverine, E. (2014, September 4). *Michigan Wolverine Football: Coaches Orchestrating The Big Game.* Retrieved from http://

gbmwolverine. com: http://gbmwolverine.com/2014/09/04/michigan-wolverine-football-coaches-orchestrating-big-game/

第 6 章

Boyd, I. (2014, August 20). *Evolving the Option.* Retrieved from SB Nation: http://www.sbnation.com/college-football/2014/8/20/6044003/read-option-pass-play-football-xs-os-diagrams

Brealy, C. M. (2016). Adding your tools to IBM Cloud Platform DevOps Toolchains. *IBM InterConnect* (pp. 12, 14). Las Vegas, NV: IBM.

Christofes, M. (2014, April 26). *Engineering Failures: Apollo 13.* Retrieved from Michael Christoffes Engineering Design Blog http://sites.psu.edu/mchristofes/2014/04/26/engineering-failures-apollo-13/

Davie, B. (2015, October 29). *Football 101: Option football.* Retrieved from ESPN.com: http://static.espn.go.com/ncf/columns/davie/1447132.html

DW on Sport. (2012, February 24). *The Importance of Goalkeeping*. Retrieved from DW on Sport: http://www.sportdw.com/2012/02/importance-of -goalkeeping.html

Elder, M. J. (2014, June 1). *Security considerations for DevOps adoption.* Retrieved from IBm DeveloperWorks: https://www.ibm.com/ eveloperworks/ library/d-security-considerations-devops-adoption/

Flynn, D. (2016, June 24). *Outsourced! Foreign Players Constitute Half of NBA Draft's First Round Picks*. Retrieved from www.breitbart.com:http://www.breitbart.com/sports/2016/06/24/outsourced-foreign-players-constitute-half-of-nba-drafts-first-round-picks/

Gartner. (2016). *IT Glossary.* Retrieved from Gartner: http://blogs.gartner.com/it-glossary/competency-center/

IBM Design Thinking Field Guide. (2016). *IBM Design Thinking Field Guide.* IBM Design.

iSport.com. (n.d.). *History of Gymnastics.* Retrieved from iSport.com Gymnastics: http://gymnastics.isport.com/ gymnastics-guides/history-of-gymnastics

Ivarsson, H. K. (2012, October). *Scaling Agile @ Spotify.* Retrieved from Scribd.com:https://www.scribd.com/document/113617905/Scaling-Agile-Spotify

Parcells, B. (2000, November). *The Tough Work of Turning Around a Team.* Retrieved from Harvard Business Review:https://hbr.org/2000/11/the-tough-work-of-turning-around-a-team

Popper, N. (2012, August 2). *Knight Capital Says Trading Glitch Cost It $440 Million.* Retrieved from New York Times: http://dealbook.nytimes.com/2012/08/02/knight-capital-says-trading-mishap-cost-it-440-million/?_r=0

Smith, J. (2015, June 22). *How "Thinking Like A Developer" Disrupts The Boardroom.* Retrieved from Forbes.com: http://www.forbes.com/sites/ibm/2015/06/22/how-thinking-like-a-developer-disrupts-the-boardroom/#d9436cb41203

TheSportsCampus. (2016, August 19). *Gopichand, the man behind India's successful women shuttlers.* Retrieved from The Sports Campus: http://www.thesportscampus.com/20160819554 75/articles/gopichand-the-man-behind-indias-successful-women-shuttlers

Toyota Production System. (n.d.). *Kaizen.* Retrieved from Toyota Material Handling: http://www.toyota-forklifts.eu/En/company/Toyota-Production-System/Kaizen/Pages/default.aspx

Whitmore, J. T. (2012, August 15). *Security in Development: The IBM Secure Engineering Framework.* Retrieved from IBm

DeveloperWorks: https://www.ibm.com/developerworks/library/se-framework/

第 7 章

Breer, A. (2013, June 10). *Who's really in charge? Power structures vary across* NFC East. Retrieved from NFL.com: http://www.nfl.com/news/story/0ap1000000210494/printable/whos-ireallyi-in-charge-power-structures-vary-across-nfc -east

Brustein, J. (2011, March 12). *A Better Way to Watch Sports*. Retrieved from nytimes.com: http://www.nytimes.com/2011/03/13/weekinreview/13watch.html?_r=0

Foss, M. (2015, March 11). *Chip Kelly reminds the NFL that letting a coach also be a GM is stupid*. Retrieved from USA Today: http://ftw.usatoday.com/2015/03/chip-kelly-reminds-the-nfl-that-letting-a-coach-also-be-gm-is-stupid

Gallo, C. (2014, August 19). *The Coach Behind The Longest Winning Streak In Sports History Shows How To Build A Champion Business Team*. Retrieved from Forbes.com: http://www.forbes.com/sites/carminegallo/2014/08/19/the-coach-behind-the-longest-winning-streak-in-sports-history-shows-how-to-build-a-champion-business-team/#a65422b1e906

Hewitt, J. (2015, June 13). *Why Mavericks are good for Sport*? Retrieved from InternationalRacer.com: http://www.j3sm.com/index.php/modules-menu/alpine-ski-racing-academy-coaches-corner/104-fast-or-technical-and-why-mavericks-are-good-for-sport

Minick, E. (2 015, April 16). *Building a DevOps Team that Isn't Evil*. Retrieved from IBM DeveloperWorks: https://developer.ibm.com/urbancode/docs/building-a-devops-team-that-isnt-evil/

Reid, L. (2015, June 22). *The Simple Math of DevOps*. Retrieved from DevOps.com: http://devops.com/2015/06/22/the-simple-math-of-devops/

Robbins, B. (2015, January 22). *Meet the technology titans transforming the NBA*. Retrieved from TheGuardian.com: https://www.theguardian.com/sport/2015/jan/22/technology-titans-transform-the-nba

Rosenbush, S. (2015, April 27). *IBM CIO Designs New IT Workflow for Tech Giant Under Pressure*. Retrieved from Wall Street Journal: http://blogs.wsj.com/cio/2015/04/27/ibm-cio-designs-new-it-workflow-for-struggling-tech-giant/

Witnify. (2014, February 11). *The 1980 Miracle On Ice: Herb Brooks*. Retrieved from sbnation.com: http://www. sbnation.com/miracle-on-ice-1980-us-hockey/2014/2/11/5400156/the-1980-miracle-on-ice-herb-brooks

附录

Forrester. (n.d.). *Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today*. Retrieved from Forrester.com: https://www.forrester.com/report/WaterScrumFall+Is+The+ Reality+Of+Agile+For+Most+Organizations+Today/-/ERES60109

Garvin, David A., Amy C. Edmondson, and F. G. (n.d.). *Is yours a Learning Organization* ? Retrieved from hbr.org: https://hbr.org/2008/03/is-yours-a-learning-organization

Kaz, P. (n.d.). *Measurement Myopia.* Retrieved from Drucker Institure: http://www.druckerinstitute.com/2013/07/measurement-myopia/

Kruchten, P. (n.d.). *The 4+1 View Model of architecture*. Retrieved from ieee.org: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759

Lean.org. (n.d.). *Lean.org*. Retrieved from Lean.org: http://www.lean.org/WhatsLean/History.cfm

Standard, B. (2014, November 4). *Khosla Ventures incubates startup to enable bank transactions via kirana stores*. Retrieved from Business Standard: http://www.business-standard.com/article/companies/khosla-incubatesstart-up-to-enable-bank-transcations-via-kirana-stores-114110400013_1.html

利用DevOps交付高价值、精简的企业级应用与系统

企业级IT具有独特的需求、能力和局限性，并面临着特殊的挑战。DevOps非常适合交付对速度和敏捷性有要求的高价值应用和系统，能够在不牺牲质量的情况下实现创新。但是，截至目前，大多数DevOps指导方法主要针对的是初创企业以及需求完全不同的互联网公司。

在《DevOps 实施手册　在多级IT企业中使用DevOps》中，IBM杰出工程师Sanjeev Sharma展示了如何基于组织的业务目标、当前的IT成熟度以及IT技术与平台在企业内实施DevOps。根据作者帮助各行各业的公司成功实施DevOps的丰富经验，他阐述了该如何让组织做好准备，迎接挑战，并进入快速发展之路。

与运动项目的战术手册一样，本书也提供了各类方案，供企业在不同环境、不同情况下使用。从中读者既可以学会针对具体业务创建独立的实施计划，又能够让团队朝着共同的目标而努力合作。本书展示了如何：

- 理解DevOps的要素与能力
- 应对多级IT环境的挑战
- 识别出可以与DevOps相互匹配的大型企业的商业目标
- 在大型企业的IT环境中实施DevOps
- 在低成本、低风险的前提下实现高价值的创新与优化
- 通过更高效的产品发布超越业务目标
- 在从大型机、移动设备、容器、微服务到云计算的多重技术栈中应用DevOps

作者简介

Sanjeev Sharma是国际知名的DevOps与云变革思想领袖。他曾经担任首席技术官(CTO)、全球技术销售负责人以及云架构师。他也是一位IBM杰出工程师(IBM Distinguished Engineer)，公认的最高级别的IBM系统内核心技术领袖。作为IBM DevOps实施的全球首席技术官，Sanjeev为来自不同行业、不同地域的实施DevOps与云变革的技术高管们提供建议及指导。

清华社官方微信号
扫 我 有 惊 喜

ISBN 978-7-302-49826-1
9 787302 498261 >
定价：79.80元

WILEY