

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 affiliates, 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：1~3000

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

IV DevOps 实施手册 在多级 IT 企业中使用 DevOps

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

X DevOps 实施手册 在多级 IT 企业中使用 DevOps

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

有些贡献者是以前在 IBM 工作过的，他们是：

■ Alan Sanie	■ Jan Svoboda
■ Ashok Reddy	■ Mike Lundblad
■ Bowman Hall	■ Murray Cantor
■ David Grimm	■ Steven Pogue
■ David Myers	■ 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，

XII DevOps 实施手册 在多级 IT 企业中使用 DevOps

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 瓦解了北卡罗来纳战术，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。

XIV DevOps 实施手册 在多级 IT 企业中使用 DevOps

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。

XVIII DevOps 实施手册 在多级 IT 企业中使用 DevOps

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 实施需要先从小项目开始启动，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一步完善与调整。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	4.3.5	方案：持续交付	120
3.9.1	业务线	77	4.3.6	方案：测试前移	133
3.9.2	IT 组织	77	4.3.7	方案：运维参与前移	139
3.10	战略伙伴	78	4.3.8	方案：持续监控与反馈	145
3.10.1	业务线	78	4.3.9	方案：发布管理	151
3.10.2	IT 组织	79	4.4	专注核心方案	154
3.11	成本结构	79	4.4.1	方案：移动设备 DevOps	154
3.11.1	业务线	79	4.4.2	方案：大型机的 DevOps	161
3.11.2	IT 组织	79	4.4.3	方案：物联网 DevOps	165
3.12	总结	80	4.4.4	方案：DevOps 用于大数据及分析	168
第 4 章	DevOps 方案之优化持续交付流水线	81	4.5	总结	173
4.1	DevOps 作为优化运动	82	第 5 章	DevOps 驱动创新方案	175
4.2	核心主题	88	5.1	优化创新	176
4.2.1	缩短周期时间	89	5.2	Uber 综合症	178
4.2.2	缩小批次规模	91	5.3	创新与技术的角色	178
4.2.3	建设正确文化理念	95	5.3.1	商业模式创新	179
4.3	DevOps 实施方案	99	5.3.2	商业模式实验	180
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			

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	7.1.2 DevOps 变革的 反模式	290
6.5 方案：DevOps 团队 模型	264	7.2 方案：发展协作信 任的文化.....	293
6.6 方案：工具与流程 标准化	267	7.2.1 可见性促进 信任	294
6.7 方案：DevOps 的 安全性考虑.....	271	7.2.2 一切都关乎人... 295	
6.7.1 管理安全相关 风险	273	7.3 方案：业务线的 DevOps 思维.....	296
6.7.2 解决 DevOps 流程 与平台的安全 问题	275	7.3.1 业务线与 IT 的 接触	297
6.7.3 API 经济与 安全	279	7.3.2 参与 DevOps 变革	298
6.8 方案：DevOps 与 外包	280	7.3.3 让影子 IT 走出 阴影	298
6.8.1 战略外包	281	7.4 方案：利用试点 项目启动.....	299
6.8.2 IT 供应链	282	7.4.1 试点项目选择... 301	
6.8.3 利用外包实现 DevOps	283	7.4.2 高层管理者 支持	302
6.9 总结	283	7.5 方案：在航空母舰 上培养独角兽.....	302
第 7 章 引领企业的 DevOps 实施.....	285	7.6 总结	306
7.1 方案：DevOps 作为 变革运动	287	附录 A 案例研究.....	307
7.1.1 令人信服的行动 理由	289	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

2 DevOps 实施手册 在多级 IT 企业中使用 DevOps

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·阿尔斯帕瓦和保罗·哈蒙德(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，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，还没有成为大型企业的主流。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。包括库克伯恩和马丁·福勒在内的 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；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。我们支持的商人都是很聪明的。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，揭示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。

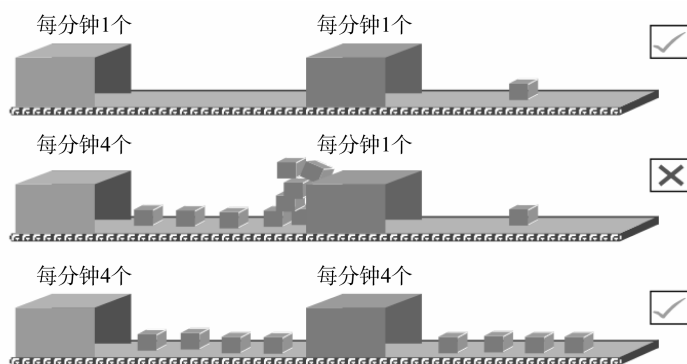


图 1-1 交付流水线瓶颈

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 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筒仓，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筒仓中，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性能。运维想要并期望能创建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性能。如果截至发布日期，他们没能开发出新的性能，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，他们有足够的時間对新功能进行测试，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性能。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

8 DevOps 实施手册 在多级 IT 企业中使用 DevOps

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不相似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接纳快速变化。

- 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。

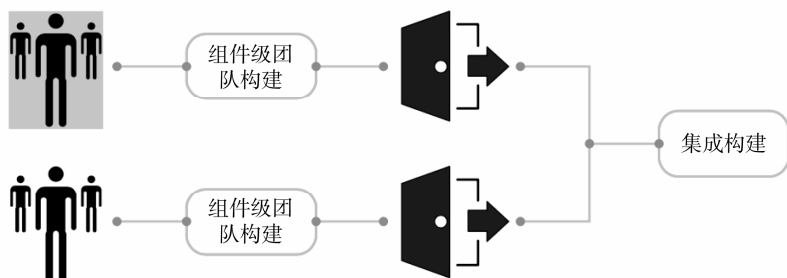


图 1-2 持续集成

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，帮助打破筒仓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与持续交付的实践——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求从一个环境部署到另一个环境。

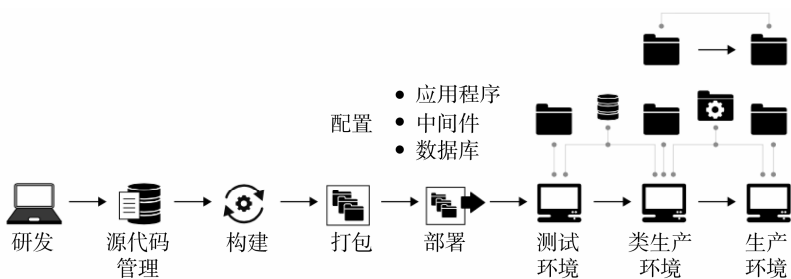


图 1-3 交付流水线

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。

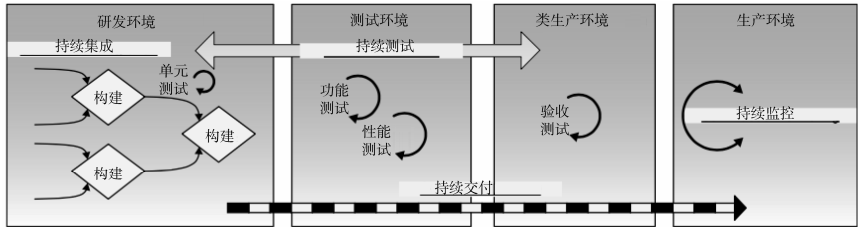


图 1-4 持续交付

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性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。这些可能并非完整的性能;几

⑤ <http://code.flickr.net>

天之内，由多个开发人员完成的一些这类更改有可能构成一个完整的可用性能。这部分性能可能是客户根本看不见的；只有在全部性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抓住了问题的本质，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交替使用。

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性能或一组性能，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 9 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性能与修复缺陷的能力。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，但是当前数周的周期时间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，可以同时解决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。

正如在配置管理系统(SCM)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。

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。流行的架构中包括 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。运维操作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, 以决定要改变什么以及如何行动。

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不同阶段而进行更改。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、添加新性能、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获取部署与生产环境以及交付应用的任务。

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。

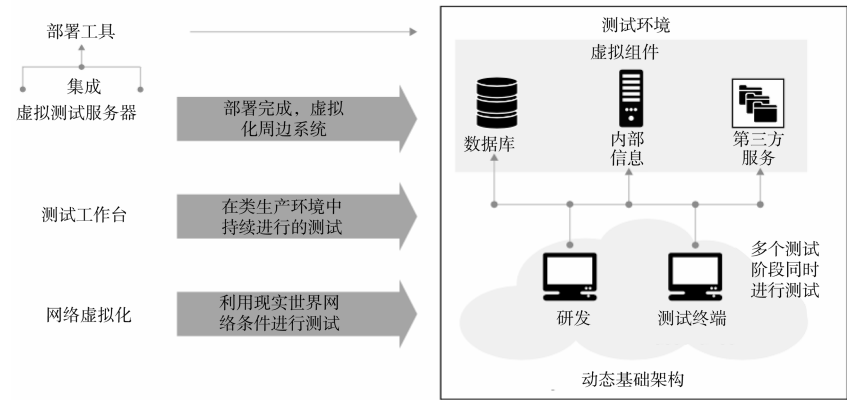


图 1-5 测试虚拟化案例

谈到 DevOps 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，在反馈数据预测性分析领域中，反映出巨大的市场潜能。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维技术。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

协同开发是由 IBM 推出而流行的，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筒仓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。前移能识别缺陷与问题的任务，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缺陷对质量存在重大影响。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筒仓。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操作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，操作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、测试人员及其他操作人员，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，我在思想中开始关注架构。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，架构终于开始在企业交付领域获得所需的关注。你无法通过大型的单片系统实现持续交付。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，并非每次都把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，通过修复或回滚可能更容易地降低风险。

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. Garvin Amy 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。改进的是什么呢？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。**可以从操作人员或利益相关者的经验中学到什么可以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 方法还有广泛的精益创业，已经完成成为学习型组织或团队所需要的相应构建，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“重要他人”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线划分职责分工；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。诸如 Slack、HipChat、Yammer 和 Wrike 这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筒仓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；取决于组织能够吸收的变化速度。为了实现业务增值，必须采取变革，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。