

新工科建设·计算机类系列教材

DevOps原理与实践

◆ 张琰彬 蒲 鹏 王 伟 编著

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

前 言

DevOps

工欲善其事，必先利其器。

——《论语·卫灵公》

背景

在无数个“偃师造人”的梦想下，人工智能走到了今天，同样在“无数个提篮印花机”对通用计算的追求下，程序设计已经成了人类意志对机械进行重塑的最有效方法之一。伴随着半导体的巨大进步、计算机从“旧时王谢堂前燕”，一跃发展到“飞入寻常百姓家”，这一巨大成功也将一大批软件、硬件、互联网行业带到了发展的快车道。随着《中共中央关于制定国民经济和社会发展第十四个五年规划和二〇三五年远景目标的建议》的发布，“数字化”一词首次进入国家五年计划，成为举国上下共同的发展目标，在当今数字化转型时代下，疫情反复更是加速了各行各业数字化转型的步伐，软件应用已经成为了各行各业的核心基础设施。企业转型与快速发展带来的业务异构化导致了整体软件生态多样化、异构化，整个行业走向更为松散、分布式、标准的服务架构，而大规模应用AI、自动化、区块链、物联网、5G、云计算和量子计算的能力使得整体软件研发生态系统变得规模化和复杂化。软件工程的复杂度在不断地提升，软件工程本身的规模复杂度也在不断提升。对云原生和全面数字化时代中的软件产品，如何适应并响应用户快速变化的需求、如何支持大规模用户在线活动并提升系统的稳定，如何提升产品交付质量、如何增加自主可控和更低的成本等变成各行各业迫在眉睫需要解决的问题。

从提出到现在，DevOps 已历经了十几年的时间，是基于精益、敏捷、丰田生产系统、柔性工程、学习型组织、安全文化等产业实践知识体系之上总结出来的工程文化和工程流程，被越来越多的企业采纳和使用，也是目前企业数字化转型的核心技术点。

相关人才需求与高校教学现状

结合 CSDN 深度调研的《2022 中国开发者大调查》和依据 GitHub 发布的《Octoverse 2021

年度报告》，我们得出了这样的结论：

截至 2022 年上半年，仅中国的软件开发人员约有 750 万左右，这个数字还在持续增长中。云原生成为了驱动业务增长的重要引擎。但是国内有 70% 的开发者完全不懂或者只了解一些概念。75% 的开发者在刚上手的工作中接触了 DevOps 相关实践，有三年以上经验的占比仅仅 7%。

在软件工程领域，软件工程本身的规模随着团队的业务架构复杂度在不断提升，导致需要足够复杂的业务来完成一个软件的构建，即：要完成一个软件产品从设计到最终的发布需要很多其他角色通力合作，包括设计团队、研发团队、测试团队、交付团体、需求团队等，这些也体现在我们研究与实践项目水杉在线平台的研发过程中。传统课程与实验教学中学习到的软件开发测试技能逐渐成为整个软件研发流程中的一个小的环节，数字化协作工作流程和方式不断发生更新迭代，如何通过实验让同学们在掌握基础软件研发技能的同时，掌握整个基础软件研发流程中人员协作、流程规范、协同链路，掌握端到端的设计思维、敏捷和 DevOps 实践框架，无缝地构思、构建、评估、迭代和扩展的解决思维与方法。这些是数字化企业软件开发协同的新生态工具链成为一个校内这类课程的问题和难点。

本书就是基于这样的背景，通过原理和实践，将 DevOps 文化理念与工程实践方法传递给从业者或者即将踏入相关岗位的学生。

内容介绍

本书内容分为理论篇和实战篇，具体如下。

原理篇包括 5 章，具体为：第 1 章，DevOps 诞生与发展；第 2 章，DevOps 标准与落地框架；第 3 章，软件交付；第 4 章，基础设施即代码；第 5 章，软件质量管理。

实战篇包括 5 章，具体为：第 6 章，DevOps 基础实践；第 7 章，DaseDevOps 示例程序；第 8 章，DaseDevOps 测试用例；第 9 章，CI/CD 实践；第 10 章，发布平台监控与日志实践。

全书理论阐述深入浅出，实践操作翔实有据，非常适合软件领域的从业者阅读，也是相关领域学生的优秀教参。本着开源放开共享的精神，实验项目中的案例代码也放在 GitHub 的 OpenEduTech/DaseDevOps 网页中，读者可以在此下载、共建，也可以在线和我们进行沟通。

本书由华东师范大学数据科学与工程学院张琰彬和蒲鹏老师组织撰写，王伟老师参与审阅。原理篇由业界知名一线专家撰写。实践篇，除了行业专家的参与，华东师范大学数据科学与工程学院研究生陈烨、李锦路、司琦、陈可璇、宁志成、王原昭、雷镇豪、郑泽洪、张天赐、张欣然等同学参与了整理和校对工作。还有很多在出版过程中给予帮助的人，限于篇幅，不再一一列出，在此一并感谢！

作 者

目 录

DevOps

理 论 篇

第 1 章 DevOps 诞生与发展	3
1.1 DevOps 概述	4
1.1.1 DevOps 文化.....	4
1.1.2 DevOps 实践.....	5
1.1.3 DevOps 生命周期.....	5
1.2 DevOps 的诞生	7
1.2.1 DevOps 的历史.....	7
1.2.2 DevOps 的优势.....	9
1.3 DevOps 的现状和发展趋势	10
1.3.1 DevOps 的现状.....	10
1.3.2 DevOps 的发展趋势.....	12
1.4 DevOps 与开源	17
本章小结.....	19
参考文献.....	19
习题 1.....	19
第 2 章 DevOps 标准与落地框架	20
2.1 DevOps 标准概述	21
2.2 DevOps 标准主要内容	21
2.2.1 DevOps 标准总体架构.....	21
2.2.2 DevOps 标准名称和主要内容.....	22
2.3 站点可靠性工程 SRE	27
2.3.1 SRE 概述	27
2.3.2 SRE 的核心原则	29
本章小结.....	29
参考文献.....	29

习题 2.....	30
第 3 章 软件交付.....	31
3.1 软件交付流程介绍.....	32
3.2 软件交付涉及的工具.....	32
3.3 持续集成.....	35
3.4 持续部署.....	38
3.5 渐进式部署.....	39
3.6 基于容器的交付.....	40
本章小结.....	45
参考文献.....	45
习题 3.....	46
第 4 章 基础设施即代码.....	47
4.1 传统的基础设施.....	48
4.2 基础设施即代码.....	48
4.3 GitOps 实践	50
4.3.1 Argo CD	50
4.3.2 Flux CD	52
本章小结.....	54
参考文献.....	54
习题 4.....	54
第 5 章 软件质量管理	55
5.1 测试自动化.....	56
5.1.1 测试自动化与 DevOps 的关系.....	56
5.1.2 测试数据构造.....	57
5.1.3 单元测试.....	60
5.1.4 接口自动化测试.....	67
5.1.5 UI 自动化测试	73
5.1.6 客户端性能测试.....	76
5.1.7 服务器性能测试.....	79
5.1.8 兼容性测试.....	89
5.1.9 客户端稳定性测试.....	92
5.1.10 服务器稳定性测试.....	94
5.2 线上监控体系.....	97
5.2.1 接口自动化巡检.....	97
5.2.2 UI 自动化巡检	99
5.2.3 用户反馈监控.....	99
5.2.4 资源监控.....	102
5.2.5 业务质量指标监控.....	107

5.3	质量标准化与可视化.....	110
5.3.1	质量标准化管理.....	110
5.3.2	质量标准化和可视化实施.....	113
5.4	测试智能化.....	118
5.4.1	测试智能化与 DevOps 的关系.....	118
5.4.2	精准测试.....	118
5.4.3	引流测试.....	120
5.4.4	契约测试.....	124
5.4.5	MLOps 简介	127
	本章小结.....	130
	参考文献.....	131
	习题 5.....	131

实 践 篇

第 6 章 DevOps 基础实践..... 135

6.1	阿里云容器镜像云基础实践.....	136
6.1.1	实验目的和实验环境.....	136
6.1.2	实验步骤.....	137
6.2	Git 基础实践	140
6.2.1	实验目的和实验环境.....	140
6.2.2	实验步骤.....	141
6.3	GitHub 基础实践.....	142
6.3.1	实验目的和实验环境.....	143
6.3.2	实验步骤.....	143
6.4	JihuLab 基础实践.....	149
6.4.1	实验目的和实验环境.....	149
6.4.2	实验步骤.....	149
6.5	Docker 基础实践.....	153
6.5.1	实验目的和实验环境.....	153
6.5.2	实验步骤.....	153
6.6	Python 基础实践	157
6.6.1	实验目的和实验环境.....	157
6.6.2	实验步骤.....	157
6.7	Java 基础实践	159
6.7.1	实验目的和实验环境.....	159
6.7.2	实验步骤.....	160
6.8	Node.js 基础实践	164
6.8.1	实验目的和实验环境.....	164
6.8.2	实验步骤.....	164

本章小结	166
第 7 章 DaseDevOps 示例程序	167
7.1 Java 微服务后端程序	168
7.1.1 实验目的和实验环境	168
7.1.2 实验步骤	170
7.2 基于 Vue 的前端程序	176
7.2.1 实验目的和实验环境	176
7.2.2 实验步骤	176
本章小结	179
第 8 章 DaseDevOps 测试用例	180
8.1 静态代码扫描	181
8.1.1 实验目的和实验环境	181
8.1.2 实验步骤	181
8.2 单元测试	182
8.2.1 实验目的和实验环境	183
8.2.2 实验步骤	183
8.3 用户界面测试用例	190
8.3.1 实验目的和实验环境	191
8.3.2 实验步骤	191
8.4 接口测试用例	198
8.4.1 实验目的和实验环境	198
8.4.2 实验步骤	199
8.5 压力测试用例	204
8.5.1 实验目的和实验环境	206
8.5.2 实验步骤	206
本章小结	211
第 9 章 CI/CD 实践	212
9.1 基于 JihuLab 的 CI/CD	213
9.1.1 实验目的和实验环境	214
9.1.2 实验步骤	214
9.2 基于 GitHub 的 CI/CD	220
9.2.1 实验目的和实验环境	222
9.2.2 实验步骤	222
9.3 基于 Jenkins 的 CI/CD	231
9.3.1 实验目的和实验环境	232
9.3.2 实验步骤	232
9.4 基于 JihuLab+Argo 的 CI/CD	242
9.4.1 实验目的和实验环境	243

9.4.2 实验步骤.....	244
本章小结.....	254
第 10 章 发布平台监控与日志实践	255
10.1 监控系统实践.....	256
10.1.1 实验目的和实验环境.....	256
10.1.2 实验步骤.....	256
10.2 日志系统实践.....	268
10.2.1 实验目的和实验环境.....	268
10.2.2 实验步骤.....	268
本章小结.....	270

第 1 章

DevOps

DevOps 诞生与发展

1.1 DevOps 概述

DevOps 一词来自 Development 与 Operations 的组合，是一种重视软件开发人员（Dev）和运维人员（Ops）之间沟通的文化、运动或惯例。通过将人、流程和技术结合起来，DevOps 利用自动化的流程来构建、测试、发布软件，使“软件交付”与“架构变更”变得更加快捷、频繁和可靠。DevOps 帮助团队提高软件和服务的交付速度，使团队能够更好地为客户服务，并提高其在市场中的竞争力。

简而言之，DevOps 可以定义为通过更好的沟通和协作，使开发和运维保持较高的研发效率和文化的一致。

DevOps 的诞生是为了填补开发端到运维端之间的信息鸿沟，改善团队之间的协作关系。随着 DevOps 的发展，产品经理、开发、运维、测试和信息安全工程师均加入其中，形成一套针对多部门间沟通与协作问题的流程和方法。

DevOps 并未对任何特定的流程、工具和技术有偏好，也并未规定任何特定的方法和环境。DevOps 并不是框架或者方法，实际上只是一系列的原理和实践经验，这些原理和实践经验并不会执行任何特定的程序、工具或技术，而是使用 DevOps 原理和经验来指导使用这些程序、工具和技术。一个不认可 DevOps 文化和原理的团队，无论使用怎样的技术和工具，都不能算是真正地实践了 DevOps。

1.1.1 DevOps 文化

要真正了解 DevOps 首先要理解 DevOps 文化。在学习之初，用户需要暂时摒弃市面上各种眼花缭乱的 DevOps 工具的宣传，应该把握 DevOps 的本质，这样才能在今后的实践中不被各种复杂的技术和高深的术语所迷惑。在组织层面，团队相关成员之间需要持续沟通、协作和共担责任，打破团队间的孤岛状态，不需交接，也不需等待其他团队的审批，同心协力，保证快速且持续的创新和高质量的交付。

DevOps 文化是团队协作的文化，需要团队所有成员的理解和认可，单凭个人是无法实践 DevOps 的，包括如下。

1. 协作的可见性

不同的团队或成员的 DevOps 流程、优先级和关注点互相可见，有统一的目标和衡量标准。这是一个健康的 DevOps 文化的标志，也是高效协作的基础。

2. 负责范围的转变

团队成员需要参与他们角色对应工作范围之外的阶段。例如，研发人员不仅要对开发阶段负责，还要对软件发布后运维阶段的性能和稳定性负责；运维人员不仅要对软件运行的环境负责，还要对软件的安全性、合规性负责，并且在软件规划阶段就要参与。

3. 缩短发布周期

缩短发布周期可以让计划和风险管理更容易，同时减少对系统稳定性的影响，还可以让组织适应和应对不断变化的客户需求和竞争压力。

4. 持续学习

DevOps 提倡快速迭代和快速试错，融入失败经验，不断改进，从而加速创新并适应市场变化，不断学习，不断成长。

1.1.2 DevOps 实践

任何企业的 DevOps 实践的最终目标都是确保相关人员(包括用户)的期望和需求能有效、快速的开展。DevOps 满足用户以下需求和期望：① 获得他们想要的功能；② 在任何需要时都能获得他们想要的反馈；③ 更快、更新的功能；④ 发布的新版本的质量够高。

只有当企业能满足这些用户需求，用户的忠诚度才会高，从而企业的市场竞争力才能得到提升，并最终增强企业的品牌和市场价值。DevOps 对企业的顶线和底线有直接影响，企业可以在创新和用户反馈上投入更多，从而持续改变其系统和服务，以保持用户的黏性。

任何企业或者团队在实践 DevOps 的过程中都会受其所处的行业和领域的影响，所以要把握住 DevOps 实践的核心原则和核心做法。

DevOps 实践的核心原则：① 协作机制和沟通机制；② 响应变化的敏捷度；③ 软件设计能力；④ 快速试错；⑤ 持续学习和创新；⑥ 自动化流程和工具。

DevOps 实践的核心做法包括：① 版本控制；② 持续集成 (Continuous Integration, CI) / 持续部署 (Continuous Deployment, CD)；③ 配置管理；④ 基础设施即代码；⑤ 持续监控；⑥ 持续迭代。

1.1.3 DevOps 生命周期

DevOps 生命周期（以线性方式描述时，也称为持续交付管道）是一系列迭代式、自动化的开发流程（也称为工作流程），在更大的自动化和迭代式的开发生命周期内执行，旨在优化高质量软件的快速交付。工作流程名称和数量因人而异，通常可归结为 6 类（如图 1-1 所示）。

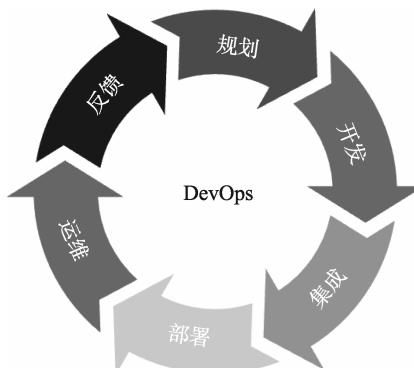


图 1-1 DevOps 生命周期涉及的工作流程

1. 规划（或构思）

根据划分了优先级的最终用户反馈和案例研究，以及来自所有内部利益相关方的输入，团队确定下一个发行版中的新功能或特性的范围。规划阶段的目标是通过生成在交付后可实现预期成果和价值的待办工作列表，最大程度提高产品的业务价值。

2. 开发

编码环节，开发人员根据待办工作列表中的用户情景和工作项，编码、测试和构建新功能和增强功能。常用的实践组合包括测试驱动开发（Test-Driven Development, TDD）、配对编程和同行代码评审等。开发人员通常使用本地工作站来执行代码编写和测试的“内部循环”，再将代码交给持续交付管道。

3. 集成（或构建，或持续集成和持续部署）

新代码集成到现有代码库中，然后执行自动化测试并打包到可执行文件中，以进行部署。常见的自动化活动包括：将代码变更合并到“主”副本中，从源代码存储库中检出代码，自动执行编译、测试，然后打包为可执行文件。最佳实践是将持续集成阶段的输出存储在二进制存储库中，以备后续阶段使用。

4. 部署（通常称为持续部署）

（来自集成的）运行时，构建输出部署到运行时环境，通常是执行运行时测试的开发环境，用于验证质量、合规性和安全性。如果发现错误或缺陷，开发人员有机会在任何最终用户看到问题之前拦截并修补任何问题。通常存在开发、测试和生产环境，每个环境都需要逐步“严格”的质量关口。部署到生产环境的最佳实践通常是首先部署到最终用户的子集，然后在产品趋于稳定后，逐步向所有用户部署。

5. 运维

如果将功能交付到生产环境描述为“第1天”，那么功能在生产环境中运行就可称为“第2天”。监控功能的性能、行为和可用性可确保功能为最终用户增值。运维确保功能平稳运行，并且服务中没有中断，也就是确保网络、存储、平台、计算和安全态势都正常。如果出错，运维可确保发现事件，提醒适当的人员，确定问题，并应用修复措施。

6. 学习（也称为持续反馈）

收集来自最终用户和客户对特性、功能、性能和业务价值的反馈，根据这些反馈，规划下一个发行版中的增强和功能；还包括来自运维活动的任何学习和待办工作项，帮助开发人员主动避免将来再次发生以往错误。这就是规划阶段的“总结”，我们“不断改进”。

除此之外，还有另外三个重要的持续工作流程。

1. 持续测试

典型的 DevOps 生命周期还包含需要在集成和部署之间完成一个单独的“测试”阶段。而 DevOps 更进一步，可以在各工作流程中执行测试的某些要素，如：在规划中执行行为驱动的开发，在开发中执行单元测试与合规测试，在集成中执行静态代码扫描、CVE 扫描和开源代码分析（Linting），在部署中执行冒烟测试、渗透测试、配置测试，在运维中执行混沌测试、

合规性测试，在学习中执行 A/B 测试。测试是一种强大的风险和漏洞发现方法，为产品提供接受、缓解或补救风险的机会。

2. 安全

虽然瀑布式方法和敏捷实施在交付或部署后添加了安全工作流程，但 DevOps 努力从一开始（规划）就整合安全工作流程（以确保能够以最低的代价尽早解决安全问题），并且在整个开发周期中持续整合安全工作流程。这种安全性方法称为“左移”。一些组织的左移工作不甚理想，这也导致了 DevSecOps 的兴起（后面章节会详细介绍）。

3. 合规性（治理和风险）

在开发过程的整个生命周期都能有效进行合规和风险的治理、把控。受监管行业通常必须强制性满足特定级别的可观察性、可跟踪性和可访问性，以表明自己如何在运行时环境中交付和管理功能。这需要在持续交付管道和运行时环境中规划、制定、测试和执行策略。合规性措施的可审计性对于向第三方审计机构证明合规性而言极其重要。

【小结】随着近些年 DevOps 理念被开发者与团队认可，越来越多的公司开始实践 DevOps；同时，越来越多的理念、工具和角色被纳入 DevOps 的范畴，如敏捷开发、自动化测试、软件安全测试等，已成为 DevOps 生命周期的一部分，并广泛应用于软件开发实践。

1.2 DevOps 的诞生

1.2.1 DevOps 的历史

所有关于 DevOps 的故事都会提起一个名字：Patrick Debois。他最早在比利时根特市发起了 DevOpsDays 活动，被称为“DevOps 之父”。DevOps 的历史也从他开始讲起。

沮丧和失望引发的思考

2007 年，Patrick Debois 受比利时政府部门的委托，协助某部门的一个大型数据中心进行迁移工作，负责认证和验证测试，因此他需要协调开发（Dev）和运维（Ops）团队的工作。这显然不是一份轻松的工作，采用敏捷开发的开发团队和采用传统运维模式的运维团队仿佛生活在两个世界，不同的工作方式像是一堵墙，这使得在两个团队之间来回切换的 Patrick 非常沮丧和失望，也让 Patrick 开始思考是否有方式可以解决这个问题。

第一届 Velocity 大会和 The Agile Admin 博客

2008 年，O'Reilly 公司在美国加州旧金山举办了第一届 Velocity 技术大会，讨论的主题是“Web 应用程序的性能和运维”。大会吸引了来自奥斯汀的几个系统管理员和开发者，他们对大会分享的内容十分感兴趣，并共同开设了一个名为 The Agile Admin 的博客，以分享敏捷开发在系统管理工作中的应用和实践。

Agile Conference 2008

2008 年 8 月，敏捷大会（Agile Conference 2008）在加拿大多伦多举办，软件工程师 Andrew

Shafer 提交了一个名为“Agile Infrastructure”的临时话题，但对这个话题感兴趣的人并不多。所以，这个话题开始时仅有一个人出席，就是 Patrick。Patrick 在这次会议上分享了自己的想法，即“如何在运维工作中应用 Scrum（迭代式增量软件开发过程）和其他敏捷实践”，十分想把这些经历进行分享。

最终，Patrick 与 Andrew 进行了一场漫长的讨论后，他们意识到，在这次会议之外会有很多的人想继续探讨这个广泛而又系统化的问题，随后在谷歌小组(Google Group)上建立了 Agile System Administration 讨论组，来讨论这个话题。最初虽然有一些话题和参与者，但是访问者依旧寥寥无几。

一个影响深远的演讲

2009年6月，第二届 Velocity 大会如期举行，最大的亮点就是 John Allspaw 和 Paul Hammond 分享了一个名为“10+ Deploys Per Day : Dev and Ops Cooperation at Flickr”的演讲。此后，几乎关于 DevOps 的资料都会引用这个演讲。他们提出的“以业务敏捷为中心，构造适应快速发布软件的工具（Tools）和文化（Culture）”的观点非常准确地描述了 DevOps 的理念，哪怕当时 DevOps 这个名词还未诞生。

Patrick 也在网上看到了这个演讲视频，在推特（Twitter）上询问如何才能参加 Velocity 大会。而 Paul 直接回复：“嘿，Patrick，你想在比利时召开自己的 Velocity 吗？我们都会去参加，这一定会很棒。”

DevOpsDays 和 DevOps 的诞生

于是乎，“社区版 Velocity 大会”就由 Patrick 在推特（Twitter）上进行了召集，于同年 10 月 30 日在比利时召开。关于会议的名称，Patrick 首先就想到了 Dev 和 Ops，而这个会议持续两天，所以他加上了“Days”，于是就有了“DevOpsDays”。

这届大会非常成功，众多开发和运维工程师、IT 管理人员和工具爱好者从世界各地蜂拥而至。两天的会议结束后，参与 DevOpsDays 的人们把这次会议的内容带向了全世界。

在推特（Twitter）上，关于 DevOpsDays 的讨论越来越多。由于推特（Twitter）的 140 个字符的限制，大家在推特（Twitter）上去掉了 DevOps 中的 Days，保留了 DevOps。

于是，“DevOps”这个称谓正式诞生。

What is DevOps

自 DevOpsDays 开始，越来越多的人认为 DevOps 将是 IT 部门正确的运作方式，而 DevOps 成为了一种促成开发与运维相结合的运动。各种各样、不同职责、不同背景的人不断地在各种场合分享自己关于 DevOps 的实践经验和理解，由此还催生了很多工具和实践。但随之而来的是，由于每个人对 DevOps 的理解不同，关于 DevOps 而产生的争议和冲突也越来越多。

于是，The Agile Admin 发表了 *What is DevOps*（什么是 DevOps）的文章，给出了 DevOps 的详细定义，并且依据敏捷的体系构造出了 DevOps 体系，包括一系列价值观、原则、方法、实践和对应的工具，并梳理了 DevOps 的历史和对 DevOps 的一些误解。

《持续交付》问世

2010 年，在第 2 届 DevOpsDays 上，《持续交付》的作者 Jez Humble 做了关于“持续交

付”的演讲。针对 Patrick 和 Andrew 最初遇到的问题，书中提到的实践给出了最佳实践。也有人说，如果《持续交付》早两年问世，也许就不会出现 DevOps。然而，随着 DevOps 理念的传播，DevOps 概念的外延越来越广，已经超出了《持续交付》涵盖的范畴。

“持续交付”是“持续集成”的延伸，而这恰恰与 2008 年敏捷大会中的观念一致。但由于发生时间的先后关系，“持续交付”被看成敏捷开发和 DevOps 文化的产物。而今，持续交付仍然作为 DevOps 的核心实践之一被广泛谈及。

《凤凰项目》发布

2013 年，由 Gene Kim、Kevin Behr、George Spafford 共同完成的《凤凰项目》发布，并大受欢迎。这是 DevOps 的一个标志性事件，因为它以小说的形式向读者介绍了 DevOps 的理念。故事讲述了一个虚构的 IT 经理 Bill Palmer 临危受命，在未来董事的帮助和自己“三步工作法”理念的支撑下，最终挽救了一家具有悠久历史的汽车配件制造商的故事。

DevOps 从小众走向主流

2016 年，Garter 预测 DevOps 将从小众战略过渡到主流战略。当年，全球 2000 强企业中有四分之一的企业采用了 DevOps 战略。

DevOps 之年和中国 DevOps 元年

Forrester Research 称 2017 年为“DevOps 之年”，同时报告说，有高达 50% 的组织正在实施 DevOps。同年，DevOpsDay Beijing 成功举办，2017 年成为“中国 DevOps 元年”。

DevOpsDay 在中国

2018 年，北京、上海、深圳成功举办了 DevOpsDay；同年，美国安排了 30 余场 DevOpsDay 活动。DevOps 的理念在全球迅速推广，其中不断有基础设施团队和运维团队提出了更多有关 DevOps 的倡议，越来越多的组织和企业开始采用 DevOps。

【小结】通过 DevOps 的历史可以看到，DevOps 诞生至今依旧保持着非常旺盛的生命力，依旧有众多的企业和工程师基于自己的实践经验不断提出不同的倡议，从而解决开发（Dev）与运维（Ops）之间的矛盾。

1.2.2 DevOps 的优势

DevOps 的优势很大程度体现在组织层面的协作和沟通上，实践 DevOps 的团队可以更快、更好地工作，简化事件响应，并改善团队之间的协作和沟通。

1. 速度

DevOps 高速运转，可以更快速地针对用户进行创新、更好地适应不断变化的市场，同时更有效地推动业务成果。DevOps 模式能够帮助开发人员和运维团队实现这些目标。例如，微服务和持续交付能够让团队充分掌控服务，然后更快速地发布更新。

2. 快速交付

DevOps 提高了发布的频率和速度，以便更快速地进行创新并完善产品。发布新功能和修

复错误的速度越快，就越能快速地响应用户需求并建立竞争优势。持续集成和持续交付是自动执行软件发布流程（从构建到部署）的两项实践经验。

3. 可靠性

DevOps 确保应用程序更新和基础设施变更的品质，以便在保持最终用户优质体验的同时，更加快速、可靠地进行交付；使用持续集成和持续交付等实践经验来测试每次变更是否安全，以及能够正常运行；监控和日志记录实践经验能够帮助实时了解当前的性能。

4. 规模

DevOps 可以用于大规模运行、管理软件的基础设施及开发流程；自动化和一致性可在降低风险的同时，有效管理复杂或不断变化的系统。例如，“基础设施即代码”有助于通过一种可重复且更有效的方式来管理部署、测试和生产环境。

5. 增强合作

建立一个适应 DevOps 文化模式的更高效的团队，强调主人翁精神和责任感。开发人员和运维团队密切合作，共同承担诸多责任，并将各自的工作流程相互融合。这有助于提升效率、节约时间（例如，缩短开发人员和运维团队之间的交接时间，编写将运行环境考虑在内的代码）。

6. 安全性

在快速运转的同时保持控制力和合规性。利用自动实施的合规性策略、精细控制和配置管理技术，可以在不牺牲安全性的前提下采用 DevOps 模式。例如，“基础设施即代码”和“策略即代码”可以大规模定义并追踪合规性。

【小结】 DevOps 更多强调人与人、开发与运维之间的通力合作，共同探索出一条符合自己团队安全、高效的 DevOps 模式。团队在进行 DevOps 转型时需要把握核心目标，切勿照本宣科，不要为了实现 DevOps 而做 DevOps。

1.3 DevOps 的现状和发展趋势

1.3.1 DevOps 的现状

DevOps 经过了 10 多年的发展，很多企业或组织都在实践 DevOps。根据 Gartner 发布的 2022 年 Agile（敏捷）和 DevOps 成熟度曲线（如图 1-2 所示）显示，DevOps 的实践处于多点开花阶段（工具链、持续交付、DevSecOps 等）。这充分说明 DevOps 已经经过了理论阶段，向着实践的深水区走去。

根据中国信通院（中国信息通信研究院的简称，下同）发布的《中国 DevOps 现状调查报告（2021）》DevOps 进行了级别划分。

- ❖ 初始级：在组织局部范围内开始尝试 DevOps 活动，并取得初期效果。
- ❖ 基础级：在组织较大范围内推行 DevOps 实践，并获得局部效率提升。

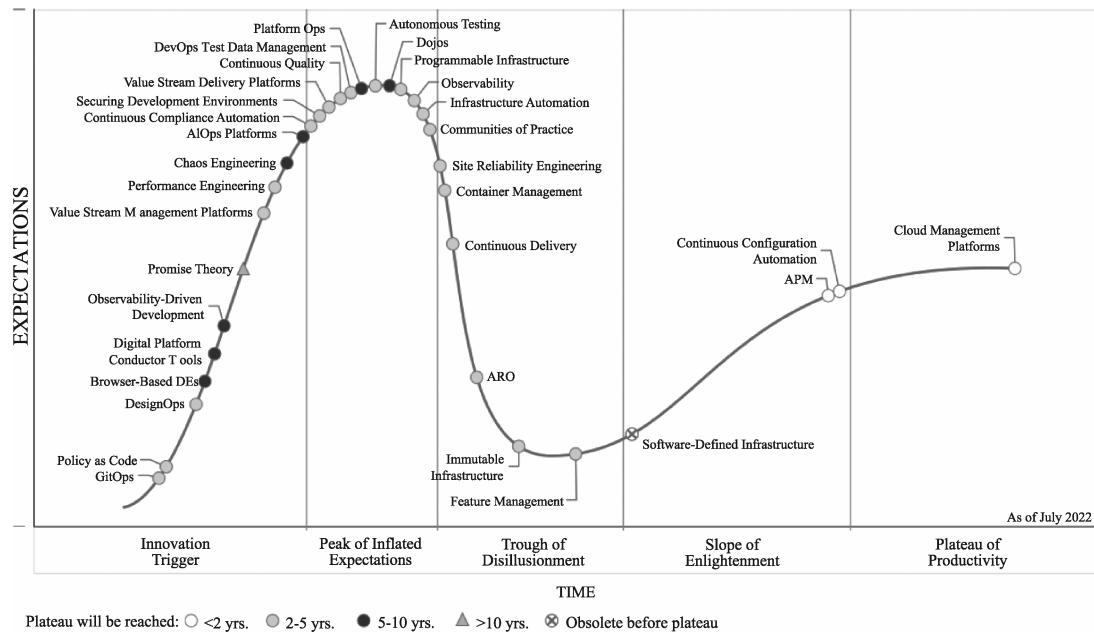


图 1-2 Gartner 2022 年 Agile & DevOps 成熟度曲线

- ❖ 全面级：在组织内全面推行 DevOps 实践并贯穿软件全生命周期，获得整体效率提升。
- ❖ 优秀级：在组织内全面落地 DevOps 并可按需交付用户价值，达到整体效率最优化。
- ❖ 卓越级：在组织内全面形成持续改进的文化，并不断驱动 DevOps 在更大范围内取得成功。

2021 年，成熟度处于全面级的企业最多，为 35.4%，具备工具化、自动化、规范化的特点，同比增长 8.84%；而 16.53% 的企业的实践成熟度处于优秀级，具备平台化、服务化、可视化与度量驱动改进的特点；0.87% 的企业处于卓越级，能够实现 DevOps 的高度智能化、数据化和社会化的特点，如图 1-3 所示。

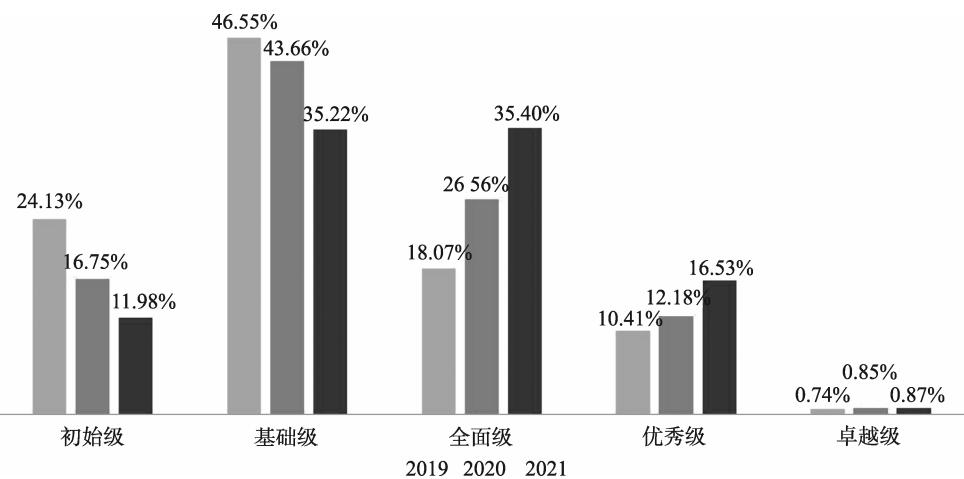


图 1-3 企业 DevOps 成熟度分布

这些年，DevOps 逐渐沉淀出了持续集成/持续交付（部署）(CI/CD)这种被认为是落地实践 DevOps 的核心手段。同样，《中国 DevOps 现状调查报告（2021）》显示：持续集成是最受

欢迎的工程实践，与自动构建、单元测试、持续交付占据前四，其比例分别为 85.16%、81.61%、81.53% 和 80.66%，如图 1-4 所示。

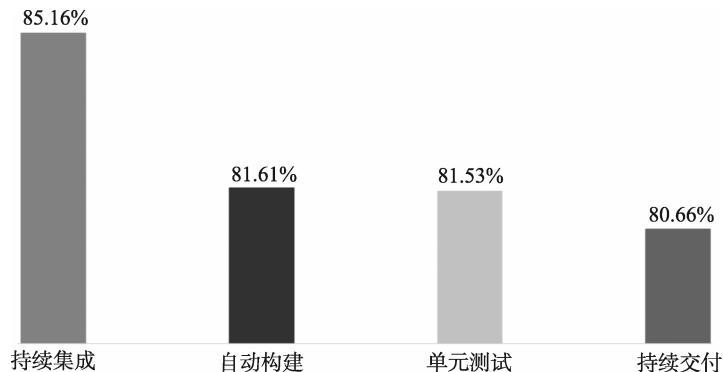


图 1-4 DevOps 最受欢迎的四大工程实践（中国）

下面简单介绍 CI/CD 的概念，第 3 章将详细介绍。

CI 和 CD 是实践 DevOps 的两大核心，也是 DevOps 实践中比较容易实现的。

CI (Continuous Integration，持续集成)：指开发人员将代码变更频繁地向主干分支集成的过程，目的是将代码变更可能造成的系统破坏性降到最小，一旦有问题，也能够快速地识别和隔离变更集，保证主干分支实时可用。当然，这种“小步快跑”的开发有一个前提，就是代码的集成要经过一系列既定的检测或者测试流程。

CD 包括两种：Continuous Delivery (持续交付) 、Continuous Deployment (持续部署)。

持续交付是一组通用的软件工程原则，允许通过使用自动化测试和持续集成频繁的发布软件新版本。持续交付可以认为是持续集成的延伸。

持续部署是指通过定义测试和验证来将风险最小化，从而将变更自动部署到生产环境。

为了推动 CI/CD 发展，Linux 基金会专门成立了持续交付基金会 (Continuous Delivery Foundation, CDF)。CDF 的目标是改善软件的交付能力，让软件交付变得更快、更安全。目前(截至本书编写时，全书同)，CDF 托管有 Cdevent 、 Jenkins 、 Jenkins-X 、 Ortelius 、 Screwdriver.cd 、 Shipwright 、 Spinnaker 、 Tekton 共 8 个与持续交付相关的开源项目。

【小结】 DevOps 经过了 10 余年的发展，已经取得了不错的成果，而且在实践的过程中沉淀了以 CI/CD 为核心关键能力的实践方法。很多企业或组织在实践 DevOps 的时候，CI/CD 往往成为第一入手点，而且围绕 CI/CD 产生了大量的开源项目，这也促成了 CDF 的成立。当然，CI/CD 并不是 DevOps 的全部，也不等于 DevOps。DevOps 还在继续演进。

1.3.2 DevOps 的发展趋势

DevOps 从未停下向前演进发展的脚步，从目前看， DevSecOps 、 Cloud Native (云原生) 是两个非常明显的发展方向。

1. DevSecOps

DevSecOps 可以认为是 DevOps 的外延或者扩展，目的是将安全融入 DevOps，让安全能

力渗透到软件开发生命周期的各阶段，从而为软件研发和交付提供全方位的安全保障能力。所以，DevSecOps 可以定义为：DevSecOps 描述了一个组织的文化和具体实践，这些文化和实践能够打破开发、安全、运维部门之间的壁垒，使得开发、运维和安全能够通过通力协作和敏捷开发来提高工作效率，实现软件的更快速、更安全交付。

DevSecOps 有以下三个特点。

(1) 安全左移

“左”或“右”是针对软件开发生命周期来说的。

在传统的软件开发模式下（典型如瀑布式开发，甚至 DevOps 出现的早期），安全介入的时间比较晚，一般是在软件开发的测试阶段，甚至更靠后，从软件开发生命周期看，是“向右”的。当软件交付的周期比较长（半年甚至一年以上）时，这种模式并不会有太多的问题，也是业界比较常用的模式，因为交付周期足够长，安全介入即使较晚，也有足够的时间来完成相关的工作。但随着用户需求的多样化、敏捷化，软件发布的频率必须提高才能响应用户日益增长的需求，所以软件的敏捷开发逐渐盛行，月发布、周发布甚至天发布都是稀松平常的。在这种发布频率下，还要保持软件的安全性，就给软件开发带来了很大的挑战。

为了应对这种挑战就需要让安全尽量早地介入，在编码甚至计划阶段就介入，从软件开发生命周期看，是“向左”的，这也是安全“左移”的由来，如图 1-5 所示。



图 1-5 安全“左移”

安全“左移”的背后有一个安全问题修复成本与软件开发生命周期关系（如图 1-6 所示），也是为什么要实现安全“左移”的最大原因。可以看出，如果在软件开发生命周期早期（计划，编码阶段）发现安全问题，修复的成本是非常低的，越往后（测试，甚至生产阶段），修复的成本越高，而且这种成本是非线性增加的。

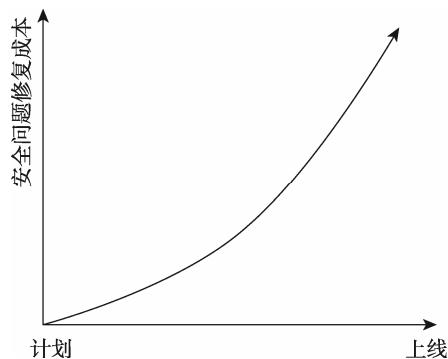


图 1-6 安全问题修复成本与软件开发生命周期的关系

(2) 持续自动化

要保证软件安全，需要有多种安全防护手段来为软件提供从计划到上线，从静态到动态的安全防护能力，诸如 SCA、SAST、DAST、IAST、Pen Testing 等。如果都采用手动测试，那

么对于开发、安全和测试来说，都是极具挑战的，不仅增加了工作量，还可能导致这些人员在重复的体力劳动中丧失工作的积极性和创造性。因此，应该尽量实现安全防护手段的自动化，并且将其嵌入 CI/CD，做到持续的自动化。

这样做有以下好处：

- ❖ 减少研发、测试等人员的工作量，减少重复的体力劳动，让他们把更多的精力放在业务创新和赋能上。
- ❖ 持续自动化能够针对每次代码变更都做到全方位安全防护，让每次代码变更都以安全方式交付。

(3) 人人为安全负责

虽然从定义看，DevSecOps 只包含开发（Dev）、安全（Sec）、运维（Ops）三个团队，但是现代软件的开发仅仅靠这三个团队是无法完成的，还需要其他团队的协作，如市场、销售、产品等。因此，任何与软件交付的团队或个人都应该为软件的安全交付负责，每个人都可能成为安全的瓶颈点，最终导致软件的不安全。这与给 F1 赛车换轮胎非常像（如图 1-7 所示），每个人都有自己的工作职责，但是最终的目的是一致的：在最短的时间内换好轮胎，并且保证安全性，从而提高车手获胜的可能性。

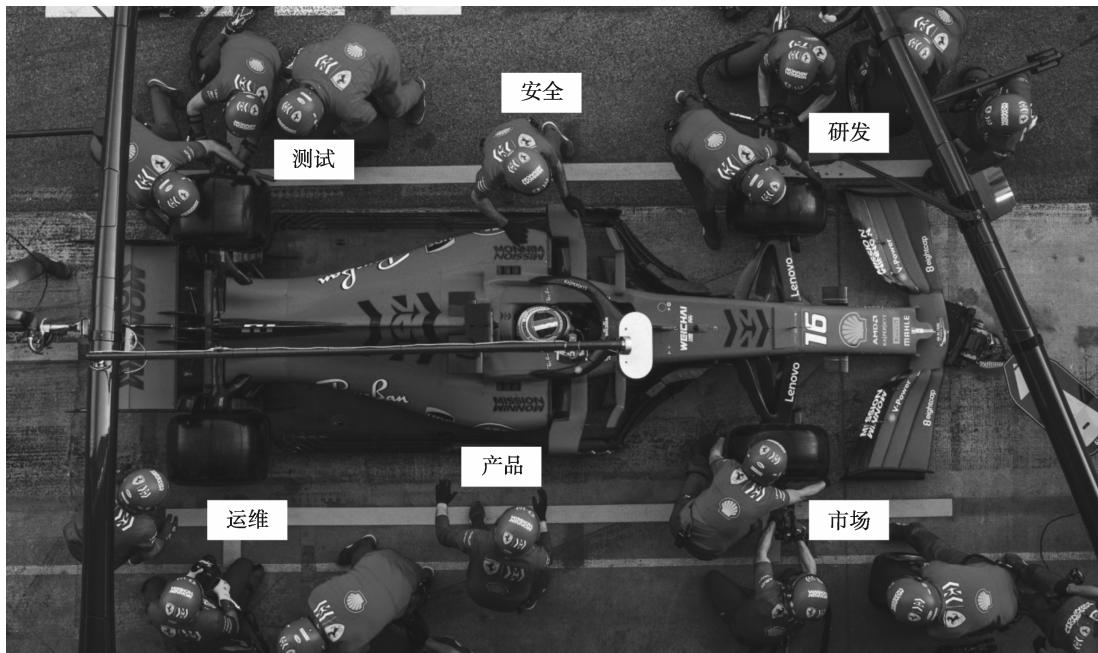


图 1-7 人人为安全负责（来自 fastcompany 网站）

2. 云原生（Cloud Native）

云原生（Cloud Native）是由 Pivotal 公司的 Matt Stine 在 2013 年提出的。Matt Stine 在他的著作 *Migrating to Cloud-Native Application Architectures* 中定义了云原生的几个特点：① 符合十二要素之应用（12-factor applications）；② 微服务（microservices）；③ 自敏捷基础设施（self-service agile infra-structure）；④ API 协同（API-based collaboration）；⑤ 反脆弱（Antifragility）。2017 年，他做了如下修改：① 模块化（modularity）；② 可观测性（observability）；

③ 可部署性 (deployability); ④ 可测性 (estability); ⑤ 可替换性 (replaceability); ⑥ 可操作性 (handleability)。

Pivotal 公司官方也对云原生有一个定义：云原生是一种充分利用云计算交付模式来构建和运行应用程序的一种方式方法，通常具有 4 个特征，即 DevOps、持续交付(部署) Continuous Delivery 、微服务 (Microservices) 、容器 (Containers) ，如图 1-8 所示。

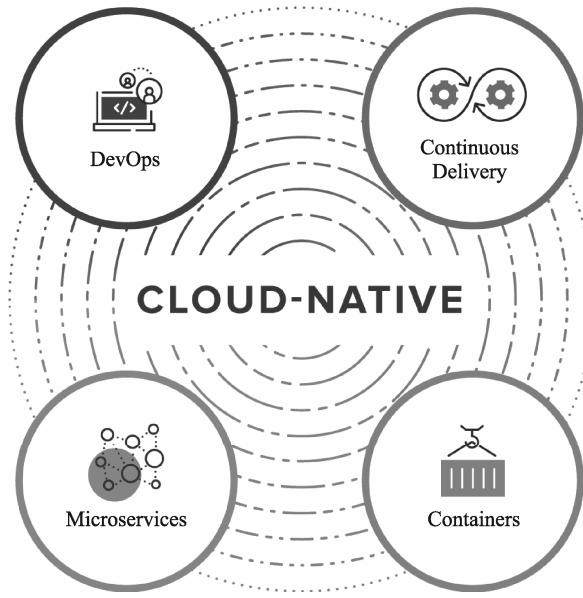


图 1-8 Pivotal 最初对云原生的定义

不过，目前业界引用最多的是云原生计算基金会（ Cloud Native Computer Foundation , CNCF ）的定义： 云原生技术有利于各组织在公有云、私有云、混合云中构建和运行可弹性扩展的应用。

云原生的代表技术包括容器、服务网格、微服务、不可变基础设施和声明式 API。

CNCF 与 CDF 一样，也是 Linux 基金会的一个子基金会，是 2015 年由 Google 、 RedHat 、 IBM 、华为、 Docker 、 VMware 等公司共同成立的，致力于推广先进的容器技术。CNCF 目前托管 141 个（ 18 个已经毕业、 37 个正在孵化、 83 个处于沙箱，并且有越来越多的项目被捐赠到 CNCF 中）开源项目，涵盖计算、存储、网络、数据库、持续交付等领域。

3. Kubernetes

下面讲述的 Kubernetes 和 GitOps 这些年在云原生领域非常热门。Kubernetes 是云原生的基座，而 GitOps 是云原生应用实现持续交付的新范式。

Kubernetes 是一个可移植的、可扩展的、开源的平台，主要用于通过声明式配置和自动化来管理容器化的工作负载和服务。Kubernetes 的名字源自希腊语，意思是舵手或者飞行员。由于开头字母“ K ”与结尾字母“ s ”之间有 8 个字母，因此通常简称为“ K8s ”。

Kubernetes 是谷歌 (Google) 公司 2014 年的一个开源项目，前身是 Borg 系统，始于 2003 年，是在谷歌公司内部使用的一个容器管理系统。Kubernetes 管理着来自数千个不同应用程序的数十万个作业，涉及许多集群，而每个集群拥有多达数万台计算机。 2013 年，另一个 Omega

系统在谷歌公司内部应用，可以看作 Borg 的延伸，在大规模集群管理和性能方面优于 Borg。但是 Borg 和 Omega 都是谷歌公司内部使用的系统，也就是所谓的闭源系统。2014 年，谷歌公司以开源的形式推出了 Kubernetes 系统，同年 6 月 7 日在 Github 上完成了第一次提交；7 月 10 日，Microsoft、IBM、Redhat、Docker 加入了 Kubernetes 社区。2015 年 7 月，Kubernetes 发布了 V1.0 版本，到目前是 V1.25 版本。

Kubernetes 集群主要的组件分为控制平面和节点(如图 1-9 所示)。控制平面(Control Plane)用来管理整个集群(如调度、检测和响应集群事件等)；节点(Node)用来运行 Pod(Kubernetes 集群最基本的调度单元，其中运行一个或者多个容器，而应用程序运行在容器中)。

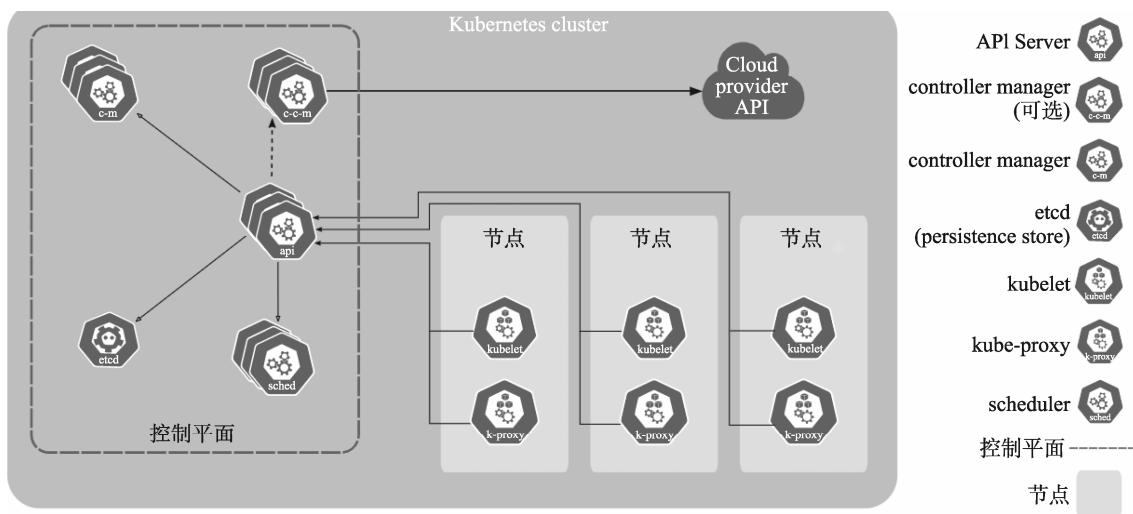


图 1-9 Kubernetes 集群组件

控制平面的主要组件包括如下。

- ① API Server (api): 面用来对外暴露 API 的组件，也是所有请求的唯一入口。
- ② etcd: 具有一致性和高可用性的键值存储组件，主要用来存储 Kubernetes 集群中的所有数据。
- ③ scheduler (sched): 对于新创建的 Pod 进行调度，为其选择一个合适的节点来运行。
- ④ controller manager (c-m): 用来运行控制器进程的组件。Kubernetes 有多种控制器，如用来管理运行一次性任务的 Job 控制器，给每个新的命名空间创建默认账号的 Service Account 控制器，以及为其创建 API 访问令牌的 Token 控制器等。
- ⑤ cloud controller manager (c-c-m): 内嵌的特定的云端控制器逻辑，用于将用户集群与云供应商的 API 进行连接，并将与该云平台交互的组件与只与用户集群交付的组件进行隔离，只运行用户指定的特定云厂商的控制器。

节点的组件包括如下。

- ① kubelet: 运行在每个节点上的代理，用于保证 Pod 内容器的正常运行。
- ② kube-proxy (k-proxy): 运行在每个节点上的网络代理，主要用于实现 Kubernetes 中服务概念的一部分。
- ③ Container runtime: 容器运行时 (runtime)，用于运行容器。Kubernetes 支持多种容器运行时，诸如 containerd、CRI-O 和其他实现了 Kubernetes CRI 的容器运行时。

4. GitOps

GitOps 是一个比较新的概念(由 Gartner 的 2021 Agile 和 DevOps 成熟度曲线也可以看出, GitOps 处于萌芽期),由 Weaveworks 公司在 2017 年提出,是一种针对云原生应用进行持续部署的方式。其本质是利用云原生不可变基础设施和声明式 API 的特征,将云原生应用的状态描述文件存储在 Git 系统上(GitHub/GitLab 等),任何变更的发起都在 Git 系统上,一旦有任何变动,变动会自动部署到目标系统上(通常是 Kubernetes 集群),如图 1-10 所示。



图 1-10 GitOps 原理

GitOps 有如下三个特征。

- ① 以 Git 为单一可信源。所有内容都存储在 Git 系统上,包括应用程序的源代码、配置文件等。常见的 Git 系统包括 GitHub、GitLab、JihuLab 等。
- ② 一切皆代码。应用程序的描述、部署、基础设施(利用 Infrastructure as Code, 基础设施即代码)等都是通过代码的方式来进行描述的,然后存储到 Git 系统上。
- ③ 以声明式系统为基座。声明式系统(如 Kubernetes)的好处是只需设定应用程序的期望状态,而不需关心整个过程,系统会自动将期望状态和实际状态进行对比,如果实际状态与期望状态有偏差,就会自动进行校正,直到期望状态和实际状态保持一致,如图 1-11 所示。

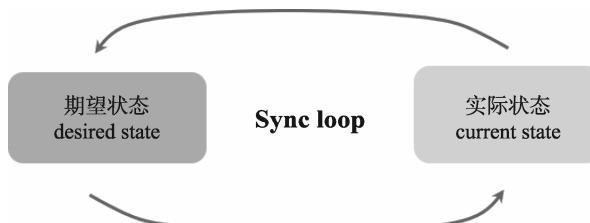


图 1-11 声明式系统原理

GitOps 的落地和实践请参阅第 4 章。

【小结】 DevOps 目前的发展趋势是将安全融入 DevOps, 打造 DevSecOps, 同时由于云原生浪潮的推动, DevOps 和云原生在互相推动着彼此的发展, 目前逐渐被认可和实践的就有 GitOps。

1.4 DevOps 与开源

DevOps 与开源看似两个不相关的领域,其实是相辅相成的。开源是 DevOps 发展的巨大推动力, DevOps 又是推动开源发展的有效手段。

1. 开源是 DevOps 发展的巨大推动力

发展至今, DevOps 的内涵和外延都发生了很大的变化。诸如 CI/CD、DevSecOps、GitOps 等,这一切落地实践的支撑就是工具。目前来看,绝大多数工具都是开源的, 使用率最高的工

具也是开源的。

从 CNCF 在 2020 年发布的持续交付技术雷达图可以看到(如图 1-12 所示),目前受欢迎、使用频率比较高的持续交付工具都是开源的。而这只是其中一小部分。

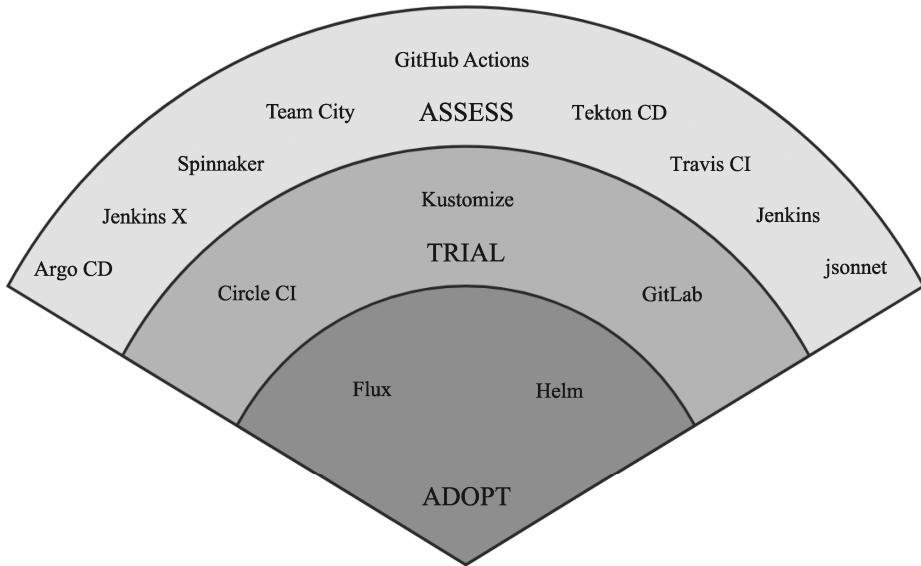


图 1-12 CNCF 在 2020 年发布的持续交付技术雷达图

中国信通院发布的《中国 DevOps 现状调查报告》显示,在持续集成与流水线中使用的工具中,Jenkins 以 64.2% 的占比排名第一, GitLab CI 以 8.86% 的占比排名第二,而这两款工具都是开源的。

目前,与持续交付相关的工具还在源源不断地涌现并且开源。

2. DevOps 是保证开源软件交付的利器

DevOps 已经成为一种用来加速软件交付、保证交付软件质量的普遍方法,开源软件的交付也不例外。比如,全球著名的开源项目 GitLab 本身就有 CI/CD 功能,因此使用自身的 DevOps 能力来开发开源项目,称为“狗粮文化”(dogfooding)。其他开源项目则会采用 GitHub 的 Action 功能来构建自己的 CI/CD,从而保证代码变更在被合入之前要经过一系列验证。

3. 协作是开源和 DevOps 的立足点

开源是一种全球异步协作的软件研发模式,协作是关键。DevOps 的出现背景和目的就是让软件研发相关的所有人员通过协作来加速软件的交付。因此,协作是开源和 DevOps 共同的立足点。

【小结】 DevOps 与开源有着密不可分的关系,两者都具有协同、协作、开放的理念,同时涌现的大量与 DevOps 相关的开源工具在持续推动 DevOps 的发展,而 DevOps 的方式也在助力开源软件以快速安全的方式进行发布交付。

本章小结

作为整本书的开篇,结合 DevOps 概述、诞生的历史、现状和发展趋势,以及开源与 DevOps 的关系,本章全面介绍了 DevOps,以便让读者对 DevOps 的全貌有整体的了解,为后面章节的学习做好基础知识的积累。

参考文献

- [1] CNCF 官网.
- [2] CDF 官网.
- [3] 敏捷宣言官网.
- [4] Kubernetes 官网.
- [5] GitOps tech 网站.
- [6] 中国信息通信研究院. 中国 DevOps 现状调查报告 (2021).
- [7] Gartner 2021 Agile 和 DevOps 成熟度曲线.
- [8] DevOps 实施手册. 在多级 IT 企业中使用 DevOps.

习题 1

- 1-1 我们为什么需要 DevOps?
- 1-2 为什么说, DevOps 是一种文化?
- 1-3 DevOps 都有哪些优势?
- 1-4 为什么说, CI/CD 并不是 DevOps 的全部?
- 1-5 DevSecOps 中安全“左移”的背后逻辑是什么?
- 1-6 GitOps 的核心目的是什么? 与 DevOps 有什么关系?
- 1-7 GitOps 都有哪些特征?
- 1-8 为什么说开源是 DevOps 背后的巨大推力?

第 2 章

DevOps

DevOps 标准与落地框架

2.1 DevOps 标准概述

人们常说“一流的企业做标准”，可见企业界对于标准的重视程度，信息科技领域也不例外。随着信息科技的升级和企业的数字化转型，目前 IT 研发运维领域的国际标准 CMMi、ISO27001、ISO20000（ITIL）等主流标准始终发挥着重要的指导作用。

近年来，随着 DevOps 文化和理念持续推动的深入发展，业界对于 DevOps 这个全新的体系提出了强烈的标准化、规范化的要求——互联网时代呼唤 DevOps 国际标准的推出。

2018 年，由中国信息通信研究院牵头，联合各行业专家，共同制定、提出了全球首个 DevOps 标准：《研发运营一体化（DevOps）能力成熟度模型》（ITU 的正式立项项目为 cloud computing-requirements for cloud service development and operation management）。

2020 年 7 月 20 日，在瑞士日内瓦举行的国际电信联盟（International Telecommunication Union, ITU）大会上，来自中国、美国、英国、德国、俄罗斯、韩国等 20 多个国家（或地区）的近 90 名专家代表参与，《研发运营一体化（DevOps）能力成熟度模型》得到了与会代表的一致同意，正式成为国际标准。

国际电信联盟（ITU）和国际标准化组织（ISO）、国际电工委员会（IEC）并称国际三大标准化组织。

2.2 DevOps 标准主要内容

DevOps 标准，即研发运维一体化（DevOps）能力成熟度模型，涵盖的内容很多，包括端到端软件交付的生命周期全流程，是一套体系化的方法论、实践和标准的集合。（注意：Ops 通常指“运维”，但在中国信息通信研究院牵头的 DevOps 能力成熟度模型中被称为“运营”。）

2.2.1 DevOps 标准总体架构

研发运维一体化总体架构可划分为四部分，即过程（含敏捷开发管理、持续交付、技术运维）、应用架构、安全管理和组织结构，如图 2-1 所示。

1. 研发运维一体化过程

研发运维一体化过程的相关内容如下：

- ① 敏捷开发管理：从需求管理、计划管理、过程管理、度量分析四个维度关注需求到开发阶段的有序迭代、灵活响应和价值的快速交付。
- ② 持续交付关注应用软件集成交付环节：通过配置管理、构建与持续集成、测试管理、部署与发布管理、环境管理、数据管理和度量管理领域的能力建设和工程实践，保证软件持续、顺畅、高质量地对用户完成发布。

一、研发运维一体化(DevOps)过程															
敏捷开发管理			持续交付							技术运营					
需求管理	计划管理	过程管理	配置管理	构建与持续集成	测试管理	部署与发布管理	环境管理	数据管理	度量与反馈	监控服务	数据服务	容量服务	连续性服务	运营反馈	
需求收集	需求澄清和拆解	迭代管理	版本控制	构建实践	测试分级策略	部署与发布模式	环境供给方式	测试数据管理	度量指标		数据收集能力	容量规划能力		业务知识管理	
需求分析	故事与任务排期	迭代活动	版本可追踪性	持续集成	代码质量管理	持续部署流水线	环境一致性	测试变更管理		质量体系管理	数据处理能力	容量平台服务	质量体系管理	项目管理	
需求与用例	计划变更	过程可视化及流动			测试自动化					事件响应及处置	数据告警能力	运营成本管理		业务连续性管理	
需求验收		度量分析								监控平台				运营服务管理	

二、研发运维一体化(DevOps)应用架构															
三、研发运维一体化(DevOps)安全管理															
四、研发运维一体化(DevOps)组织结构															

图 2-1 研发运维一体化（DevOps）能力成熟度模型

③ 技术运维环节关注应用系统服务发布后的环节：涉及运维成本服务、高可用架构服务、用户体验服务、客户服务、监控服务、产品运行服务和运营数据服务，保障良好的用户体验，打造持续的业务价值反馈流。

2. 应用架构、安全管理和组织文化建设

设计良好的应用架构有助于系统解耦和灵活发布，也是高可用系统的核心能力；端到端的安全考量和全局规划可以让安全发挥更大的价值，并真正助力全价值链。跨功能团队的组织架构和高度互信协同，责任共担的组织文化同样会对组织能力的提升带来正向作用。

以上几部分相互关联，密切协同，构成了一个有机整体，帮助组织 IT 效能不断进化，最终达成企业的业务目标。

2.2.2 DevOps 标准名称和主要内容

研发运维一体化能力成熟度模型是一个系列标准，包括的标准名称和主要内容如下。

第 1 部分：总体架构

总体架构部分规定了研发运维一体化（DevOps）的概念范围、总体架构及能力成熟度模型，主要内容如下。

- ① 引入配置项、制品、代码复杂度、部署流水线、研发运维一体化等基本概念。
- ② 研发运维一体化（DevOps）能力成熟度的级别分为 5 个，即 1 级（初始级）、2 级（基础级）、3 级（全面级）、4 级（优秀级）、5 级（卓越级）。
- ③ 研发运维一体化总体架构，分为 5 部分：过程（敏捷开发管理、持续交付、技术运维）、应用设计、安全风险管理、评估方法、系统和工具技术要求。
- ④ 过程管理，分为 3 部分：敏捷开发管理、持续交付和技术运维。
- ⑤ 应用设计：有助于系统解耦和灵活发布，及时响应业务变化，也是高可用和高性能系统的核心能力。
- ⑥ 安全及风险管理：安全考量和全局规划，可以让安全发挥更大的价值，并真正助力应用的全生命周期安全管理。

⑦ 组织结构：跨功能团队的组织架构和高度互信协同，责任共担的组织文化同样会对组织能力的提升带来正向作用，主要从组织形态、文化塑造、人员技能、创新管理和变革管理五个维度的指标进行描述。

⑧ 评估方法：指研发运维一体化能力成熟度模型的通用评估方法，规定了研发运维一体化能力成熟度模型相关技术的分级指标内容明细、评估方式与验收条件。

⑨ 系统和工具技术要求：应具备的体系结构、功能要求、接口要求和技术要求，用于指导研发运维一体化（DevOps）平台产品的规划、设计和实现。

第2部分：敏捷开发管理过程

本部分规定了研发运维一体化（DevOps）能力成熟度模型下敏捷开发管理过程的能力成熟度要求和评价方法，主要内容如下。

① 引入用户故事、用户故事地图、影响地图、A/B 测试等敏捷开发术语和定义。

② 敏捷开发管理：一种软件开发方法，应对快速变化的市场和技术环境，更强调价值交付过程中涉及的各类角色（如业务、产品、开发和测试等）之间的紧密协作，能够很好地适应变化的团队组织、协作和工作方式，主张演进式的规划和开发方式、持续和尽早交付，并不断反馈调整和持续改进，鼓励快速和灵活地面对变更，更注重软件开发过程中人的作用。

③ 价值交付管理：包括需求工件、需求活动两部分，体现需求管理过程中的分析、测试、验收三个阶段。价值交付管理主要体现在各环节中使用敏捷方法探寻用户（客户）问题和诉求、业务价值，以及定义有效产品功能的能力、适应需求变化的能力、快速验证反馈的能力，进一步定义需求工件和需求活动。

④ 敏捷过程管理：指产品经理、研发团队，以及与产品相关的干系人围绕业务价值交付进行的软件研发过程，包括价值流、仪式活动两部分，要求产品经理、团队和与产品相关的干系人建立以尽早持续交付有价值的软件为目标，通过高效的沟通方式、高效的可视化工作流程、有效的度量和快速反馈机制实现软件研发业务价值最大化，进一步定义价值流和仪式活动。

⑤ 敏捷组织模式：指团队在研发过程中的角色定义、角色能力及其协作，以及团队结构的工作方式、团队间的协作模式等方面的要求，主要从敏捷角色、团队结构两方面进行定义，进一步定义敏捷角色和团队结构。

第3部分：持续交付过程

本部分规定了研发运维一体化（DevOps）能力成熟度模型下持续交付过程的能力成熟度要求和评价方法，主要内容如下。

① 引入配置项、制品、代码复杂度、部署流水线等术语定义。

② 持续交付：指持续将各类变更（包括新功能、缺陷修复、配置变化、实验等）安全、快速、高质量地落实到生产环境或用户手中的能力。

③ 配置管理：指所有与项目相关的产物以及它们之间的关系都被唯一定义、修改、存储和检索的过程，保证软件版本交付生命周期过程中所有交付产物的完整性、一致性和可追溯性。

④ 构建和持续集成：构建是将软件源代码通过构建工具转换为可执行程序的过程，一般包含编译和链接两个步骤，将高级语言代码转换为可执行的机器代码并进行相应优化，提升运行效率。

⑤ 测试管理：指一个过程，对于所有与测试相关的过程、方法进行定义。在产品投入生产性运行前验证产品的需求，尽可能发现并排除软件中的缺陷，从而提高软件质量。

⑥ 部署和发布管理：泛指软件生命周期中，将软件应用系统对用户可见，并提供服务的一系列活动，包括系统配置、发布、安装等。整个部署和发布过程复杂，涉及多个团队之间的协作和交付，需要完备的计划和演练，保证部署发布的正确性。

⑦ 环境管理：DevOps 持续敏捷交付过程中最终的承载，包括环境的生命周期管理、一致 性管理、环境的版本管理。环境管理是用最小的代价来达到确保一致性的终极目标，主要包括环境类型、环境构建、环境依赖与配置管理三方面。

⑧ 数据管理：为了满足不同环境的测试需求，以及保证生产数据的安全，需要人为准备数量庞大的测试数据，保证数据的有效性，以适应不同的应用程序版本。另外，应用程序在运行过程中会产生大量数据，这些数据同应用程序本身的生命周期不同，作为应用最有价值的内容需要妥善保存，并随应用程序的升级和回滚进行迁移。

⑨ 度量和反馈：强调在持续交付的每个环节建立有效的度量和反馈机制，通过设立清晰可量化的度量指标，有助于衡量改进效果和实际产出，并不断迭代后续改进方向。另外，设立及时、有效的反馈机制可以加快信息传递速率，有助于在初期发现问题、解决问题，并及时修正目标，减少后续返工带来的成本浪费。度量和反馈可以保证整个团队内部信息获取的及时性和一致性，避免信息不同步导致的问题，明确业务价值交付目标和状态，推进端到端价值的快速有效流动。

第 4 部分：技术运维过程

本部分规定了研发运维一体化（DevOps）能力成熟度模型下技术运维管理的能力成熟度要求和评价方法，主要内容如下。

① 技术运维管理目标：指以业务为中心，交付稳定、安全、高效的技术运维服务，构建业界领先的技术运维能力，支撑企业的持续发展和战略成功。技术运维不仅关注“稳定”“安全”和“可靠”，更关注“体验”“效率”和“效益”。

② 技术运维管理过程：分为监控管理、事件与变更管理、配置管理、容量与成本管理、高可用管理、业务连续性管理、用户体验管理等。

③ 监控管理：指对研发运维过程中的对象进行状态数据采集、数据处理分析和存储、异常识别和通知及对象状态可视化呈现的过程，其成熟度决定了技术运维工作的立体性、及时性和有效性。

④ 事件和变更管理：指技术运维和 IT 服务过程的两个重要管理手段，包括事件管理和变更管理两部分。事件管理是对影响生产的事故和问题建立预防、高效处理及度量改进的制度和手段；变更管理是对 IT 基础设施、系统应用、业务产品配置等场景实施变更所进行的审批和控制流程。

⑤ 配置管理：指由识别和确认系统的配置项、记录和报告配置项状态和变更请求、检查配置项的正确性和完整性等活动构成的过程，目的是提供 IT 基础架构的逻辑模型，支持其他服务管理流程特别是变更管理和发布管理的运作。

⑥ 容量和成本管理：指对容量和成本进行评估、规划、分析、调整和优化的过程，结合了业务、服务和资源容量需求，以保证对资源的最优利用，满足与用户之间所约定的性能等级

要求，在公司 IT 规模较大或业务快速增长时，容量和成本管理更重要。

⑦ 高可用管理：指系统无中断地执行其功能的能力，代表系统的可用性程度，包括应用高可用管理和数据库高可用管理两部。

⑧ 业务连续性管理：指对企业识别潜在危机和风险，并制定响应、业务和连续性的恢复计划的过程进行管理，目标是提高企业的风险防范意识，有效响应非计划的业务中断或破坏，并将不良影响降低到最低。

⑨ 用户体验 (User Experience, UE/UX) 管理。用户体验是用户在使用产品过程中建立起来的一种主观感受，一般是有关产品设计方面的，不同的产品对用户体验的追求不同。本节提到的用户体验管理指的是通过技术运维手段来提升用户使用产品直观感受。

第 5 部分：应用架构

本部分规定了研发运维一体化 (DevOps) 能力成熟度模型中应用设计能力的成熟度要求，主要内容包括如下。

① 概念定义：引入软件架构、应用程序、运行时环境、软件包等定义。

② 应用设计：DevOps 技术能力包括开发技术、测试技术、运维技术等能力，其中开发技术中最核心的是应用设计相关技术。

③ 应用接口：指软件系统不同组成部分衔接的约定。

④ 应用性能：指应用实际性能 (Real Performance, 与感知性能 Perceived Performance 相对) 和可用性 (Availability) 的度量，是衡量应用服务水平的重要指标。

⑤ 应用扩展：指应用程序在达到最大负载时支持进行扩展，以保证系统稳定运行的手段和方法，是应对高并发的重要手段。

⑥ 故障处理：指在系统失效、停止响应或出现异常时识别、规划和解决系统问题的过程。在系统运行过程中，运行环境的变化、软件本身的缺陷等可能造成系统运行故障，故障处理技术可以帮助快速修理和恢复系统。

第 6 部分：安全管理

本部分规定 IT 软件或相关服务在采用研发运维一体化 (DevOps) 统一开发模式下，如何保障 IT 软件和相关服务的安全，进行风险管理，主要内容包括如下。

① 概念定义：引入安全基线、安全门限、安全态势感知、安全需求基线、安全需求标准库、暴力破解、分布式拒绝服务攻击、攻击面分析、工作项、黑盒安全测试、红蓝对抗等安全概念。

② 安全及风险管理：相比于传统开发模型，IT 软件或相关服务在采用研发安全运维一体化 (DevSecOps) 的开发模式下，安全需融入每个阶段，开发、安全、运维各部门需紧密合作。

③ 研发运维一体化控制通用风险。在 DevOps 模式下，安全内建于开发、交付、运维过程中，通用风险覆盖三个过程中的共性安全要求，包括组织建设和人员管理、安全工具链、基础设施管理、第三方管理、数据管理、度量与反馈改进。

④ 研发运维一体化控制开发过程风险管理：为保障进入交付过程的代码是安全的，降低后续交付、运维中的安全风险，保障研发运维一体化的整体安全，定义了从应用的开发过程开始需实施安全风险管理工作的管理过程，包括需求管理、设计管理和开发过程管理。

⑤ 研发运维一体化控制交付过程风险管理。交付过程是指从代码提交到应用发布给用户使用，定义了将安全内建到交付过程中的安全交付管理，包括配置管理、构建管理、测试管理、部署和发布管理。

⑥ 研发运维一体化控制技术运维过程的安全风险管理。技术运维过程是指应用发布给用户后的过程，定义了通过监控、运维、响应、反馈等实现技术运维的安全风险闭环管理方式将安全内建于运维过程中，包括安全监控、运维安全、应急响应、运维反馈。

第 7 部分：组织结构

本部分规定了研发运维一体化（DevOps）能力成熟度模型中组织结构的能力成熟度要求，主要内容包括如下。

① 概念定义：引入平台型组织、多功能团队等定义。

② 组织结构：分别从组织形态、文化塑造、人员技能、创新管理、变革管理五个维度描述研发运维一体化（DevOps）能力成熟度模型在组织结构上的不同级别。

③ 组织形态：在 DevOps 场景中，采用适当组织形态，可以让团队各类角色更好地分工协作，降低组织内不同部门或角色之间的交接成本和等待浪费，对于达成企业的绩效目标非常关键。

④ 文化塑造：在 DevOps 场景中，文化塑造是基于组织发展的不同阶段而实现动态调整和升级的过程，是一个组织是否有能力适应快速变化环境和持续改进的关键要素。组织能力持续的改进依赖于组织内部是否能够形成高度信任、相互协作和持续学习的文化。

⑤ 人员技能：在 DevOps 场景中，人员专业化的能力并掌握多项技能的综合能力是在复杂环境中解决问题、提升绩效的关键要素；鼓励员工在专精专业领域技能的基础上，理解软件生命周期上下游的多种技能，成为多面手，能够促进整体价值流在公司内部更顺畅地流动。

⑥ 创新管理：在 DevOps 场景中，主动迎接需求变更，快速响应市场变化，需要组织激发出富有创造力的团队，培养团队成员形成一种创新的习惯，从而不断发现解决问题的新方法，或者将已有的技术进行应用创新，改善或改造产品，解决用户需求，并不断为组织创造新的机会和价值。

⑦ 变革管理：DevOps 提倡变革，通过变革更好地强大自己，使得企业快速成长，从而更好地适应社会的发展。组织的 DevOps 转型成功与否，变革管理至关重要，需要对企业战略、组织结构、工作流程、工程工艺、技术方法和企业文化定期进行分析评估，不断改进，降低成本和减少浪费，达到最佳产出和效率最大化。

第 8 部分：系统和工具技术要求

本部分规定了研发运维一体化（DevOps）过程涉及的系统和工具的能力技术要求，主要内容包括如下。

① 概念定义：引入负载均衡、网络钩子、生产环境、中间件等定义。

② 总体架构：指将端到端软件交付生命周期全流程用工具链进行连接，包括项目与开发管理、应用设计与开发、持续交付（部署）、测试管理、自动化测试、技术运维。

③ 项目与开发管理：指根据用户要求建造出软件系统或者系统中软件部分的过程的管理工具，包括项目管理、工作项管理、计划管理、文档与知识管理、团队协同、统计度量、项目

集管理。

④ 应用设计与开发：指应用架构设计与代码开发过程，包括应用框架、集成开发环境、威胁建模。

⑤ 持续交付（部署）：指持续地将各类变更（包括新功能、缺陷修复、配置变化、实验等）安全、快速、高质量地交付到生产环境或用户手中的能力的管理工具。相关工具包括：版本控制系统、构建与持续集成、流水线、制品管理、部署管理、发布管理、环境管理、应用配置管理、数据变更管理、移动应用安全加固。

⑥ 测试管理：指对测试过程进行管理，从需求确定后进行测试用例编写到测试完成的管理工具。相关工具包括：用例与测试计划管理、测试数据管理。

⑦ 自动化测试：指以自动化的方式对应用进行功能与非功能测试。相关工具包括：代码质量管理、单元测试、接口/服务测试、UI 测试、移动应用测试、性能测试、安全性测试。

⑧ 技术运维：指从技术方面支撑与完善 IT 系统日常运维保障、运维工具建设、运维决策辅助，从技术方面支撑业务运营，提升 IT 的业务价值，获得可持续竞争优势的管理方式与工具。技术运维管理过程分为：监控管理、事件管理与变更管理、配置管理、容量与成本管理、高可用管理、业务连续性管理、用户体验管理等。本节将从上述运营管理过程涉及工具的维度分别阐述。相关工具包括：配置管理、运维数据分析、应用性能监控、基础监控管理、日志监控管理、自动化作业平台、容量管理、成本管理、资产安全风险管理

DevOps 标准的运行和实施、推广、应用情况

DevOps 标准已经得到国内各行业广泛接受和普遍认同，截至 2022 年第一季度，参考标准进行的企业包括银行、证券、保险、通信行业等领域的企业。

2.3 站点可靠性工程 SRE

2.3.1 SRE 概述

DevOps 文化在全球持续推广和得到认可的同时，也有一些开发运维相关的知名管理框架和最佳实践在世界范围得到推广，比较突出的是谷歌公司的站点可靠性工程（Site Reliability Engineering，SRE）。

SRE 是一种结合软件工程各方面并将其应用于基础架构和运维问题的技术，于 2003 年左右由谷歌公司创建，并通过 SRE books 进行宣传。SRE 的主要目标是创建超大规模和高度可靠的分布式软件系统，因为在很多方面均给出了有效的实践创新，所以成为运维研发领域被广泛认同的 IT 管理框架。SRE 被称为运维领域的 DevOps 最佳实践框架。

- ① SRE 将 50% 的时间花在 Ops（运维）相关工作上，如问题解决、电话值班和手工干预。
- ② SRE 将 50% 的时间花在 Dev（开发）任务上，如新功能、扩展或自动化。
- ③ 监控、警报和自动化是 SRE 的主要部分。

SRE 与 DevOps 的关系如图 2-2 所示。

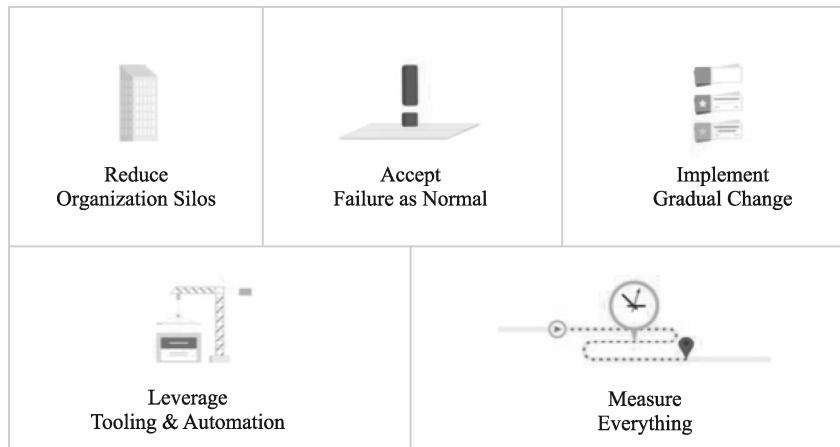


图 2-2 SRE 与 DevOps 的关系

谷歌公司定义了 DevOps 成功的 5 个关键支柱：减少组织的竖井，接受失败即常态，实现渐进的变更，利用工具和自动化，测量一切。

从上述支柱来看，SRE 与 DevOps 的核心理念是完全契合的。谷歌公司认为，SRE 是“带有一些扩展的 DevOps 的特定实现”，如图 2-3 所示。

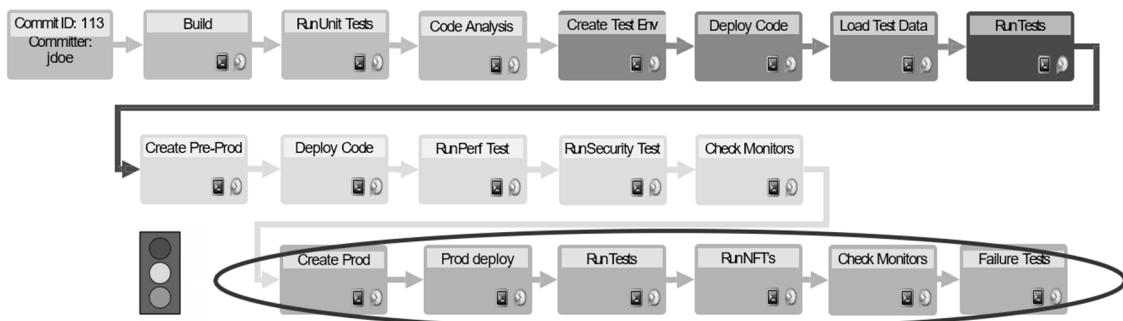


图 2-3 SRE 对 DevOps 的拓展

SRE 是谷歌公司在开发运维的落地过程实现的优化和拓展，包含如下。

1. 运维侧的拓展

SRE 在 DevOps (开发、测试、运维) 的基础上将整个流程进行了拓展：创建生产环境 (原有) → 生产部署 (原有) → 运行环境测试 (新增) → 运行环境非功能性测试 (新增) → 检查监控 (新增) → 错误测试 (混沌工程，新增)。

2. 安全生产

在上线前，SRE 增加了“安灯绳”(Andon，起源于日本丰田汽车公司，用来实现“立即暂停制度”，即时解决质量问题，达到持续高品质地生产)环节，也就是说，SRE 可以针对任何影响安全生产的流水线直接喊停 (Say No)，可以避免高度自动化条件下的研发错误、无序发布、生产。

总体来说，SRE 更关注生产环境 (PROD)——使 DevOps 获得了“运维的智慧”，SRE 可

以说“不”。

2.3.2 SRE 的核心原则

第一条：运维是软件问题

SRE 的基本原则是，做好运维最重要的是软件问题，应该使用软件工程方法来解决运维问题。SRE 是将运维从传统的操作和维护转成关注，使用软件工程对整个运维流程进行设计和构建的技术。

据估计，软件总成本 40%~90%是在产品发布后产生的，因此必须从源头做好软件。

第二条：一切围绕服务水平

服务水平目标(Service Level Objective, SLO)即产品或服务的可用性目标(不必是 100%)。SRE 强调使用 SLO 来管理 SRE 服务，一切围绕 SLO。

第三条：减少琐事 (Toil)

谷歌公司认为，任何琐事（手动的、强制的运维任务）都是很糟糕的（长期执行成本、增加人为错误），如果一个任务可以自动化，它就应该自动化，以持续减少琐事。

第四条：自动化

SRE 强调，将当前手工完成的工作都实现自动化，并建议采用一种基于工程的方法来解决问题，而不是一遍一遍地苦干。

自动化是 SRE 的主要工作。

第五条：减少失败的成本

后期问题（缺陷）的发现是昂贵的，所以 SRE 寻找方法来避免，通过提升软件系统健壮性来改善平均修复时间（ Mean Time To Recovery, MTTR ）。

第六条：共享所有权

SRE 与产品开发团队共享技能集，以避免竖井。“应用程序开发”与“生产”（开发和运维）之间的界限应该被消除。SRE 的“左移”为开发团队提供“生产智慧”。

本章小结

本章介绍了国内外 DevOps 标准和最佳实践，DevOps 文化和理念在全球持续推广和得到认可。读者可以了解到目前业界对 DevOps 的范围定义及落地过程中的实践注意事项和方法。

参考文献

- [1] 中国信通院官网.

- [2] CDF 官网.
- [3] 敏捷宣言官网.
- [4] 谷歌 SRE 官网.

习题 2

- 2-1 DevOps 标准中不包括（ ）。
- A. 敏捷开发管理
 - B. 持续交付
 - C. 技术运维
 - D. 快速迭代
- 2-2 在 DevOps 标准中，（ ）领域是默认要求。
- A. 敏捷开发管理
 - B. 应用架构
 - C. 组织结构
 - D. 安全管理
- 2-3 Google 定义了 DevOps 成功的关键支柱，包括（ ）。(多选)
- A. 减少组织的竖井
 - B. 正常接受失败
 - C. 实现渐变
 - D. 安全左移
- 2-4 SRE 对 DevOps 的主要拓展包括（ ）。
- A. 创建生产环境
 - B. 测试环境搭建
 - C. 生产环境测试（混沌工程）
 - D. 错误测试
- 2-5 DevOps 的具体落实需要从组织、管理、技术角度进行变革，您有哪些建议？