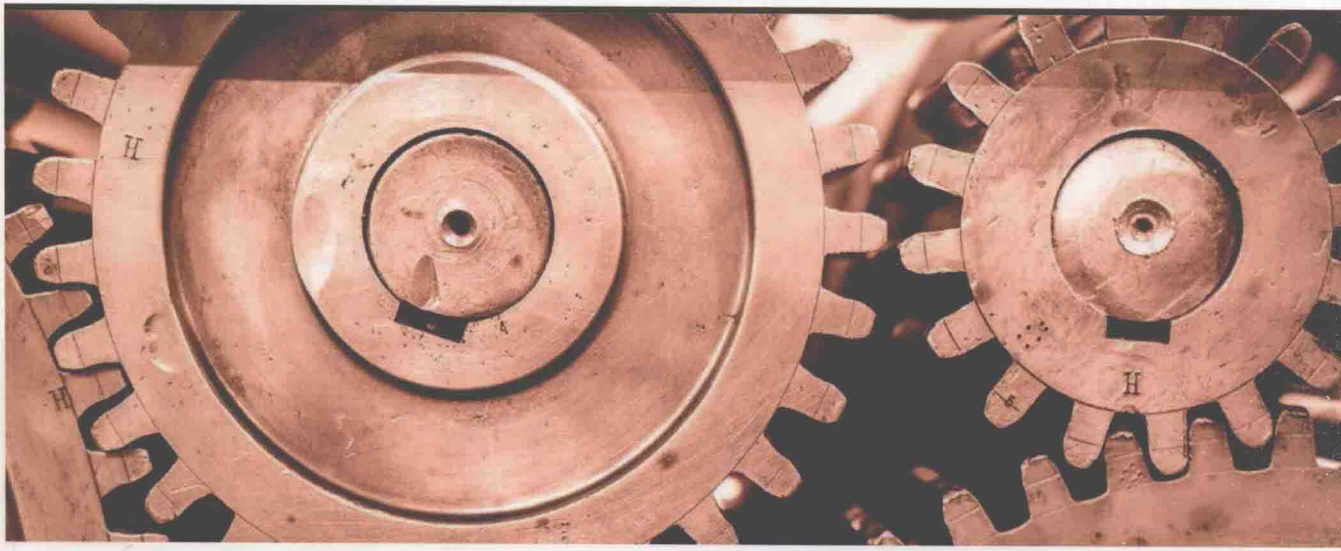
Packt>

微服务运维实战（第一卷）

The DevOps 2.0 Toolkit



Automating the Continuous Deployment Pipeline with Containerized Microservices

[西] Viktor Farcic 著
任发科 何腾欢 汪欣 袁诗瑶 译

華中科技大學出版社
http://www.hustp.com

微服务运维实战

（第一卷）

[西] Viktor Farcic 著

任发科 何腾欢 汪 欣 袁诗瑶 译

华中科技大学出版社

中国·武汉

内容简介

本书详细介绍了微服务和容器在软件持续集成和部署中的应用。将微服务打包成不可变的容器，通过配置管理工具实现自动化测试和持续部署，同时保证零停机且随时能回滚。采用集中日志对集群进行记录和监控，轻松实现服务器扩展。作者通过介绍相关工具（Docker、Kubernetes、Ansible、Consul 等）的用法，分享自己的工作经验，帮助读者构建高效、可靠、可快速恢复的软件系统。

湖北省版权局著作权合同登记　图字：17-2018-119 号

图书在版编目(CIP)数据

微服务运维实战. 第一卷 /（西）维克托·法西克著；任发科等译. —武汉：华中科技大学出版社，2018.6

ISBN 978-7-5680-4161-4

Ⅰ.①微… Ⅱ.①维… ②任… Ⅲ.①互联网络-网络服务器-程序设计 Ⅳ.①TP368.5

中国版本图书馆 CIP 数据核字(2018)第 101100 号

微服务运维实战(第一卷)

Wei Fuwu Yun-wei Shizhan

[西]Viktor Farcic 著

任发科 何腾欢 汪欣 袁诗瑶 译

策划编辑：徐定翔

责任编辑：陈元玉

责任监印：周治超

出版发行：华中科技大学出版社(中国·武汉)　电话：(027)81321913

武汉市东湖新技术开发区华工科技园　邮编：430223

录　排：华中科技大学惠友文印中心

印　刷：湖北新华印务有限公司

开　本：787mm×960mm　1/16

印　张：27.25

字　数：655 千字

版　次：2018 年 6 月第 1 版第 1 次印刷

定　价：115.00 元

推荐序

马克思的辩证唯物主义观点告诉我们，为了解决某个社会问题而诞生的新生事物，在流行并占据统治地位后，必然会出现它的反面，也就是负面的影响。然后又会出现新生事物来消除前者造成的负面影响。这样循环往复，推动人类社会向着更高级的方向发展。

软件开发也遵循辩证唯物主义的规律。早期的软件应用都是单片应用，随着流量的增大，单片应用无法支持，而且复杂的单片应用也难以维护和测试，最终开发团队只好将单片应用化整为零，变成分布式应用。分布式应用的设计和开发很复杂，所以出现了一些新的开发方法，如面向服务架构（SOA）和微服务架构（MSA）。千万不要以为 MSA 就是软件开发最终的理想国。MSA 仍然有很多令人头疼的地方，其中最主要的一个方面是运维。

在 MSA 流行之前，一个软件应用即使是分布式的，服务数量通常也不多（不超过 10 个），运维工程师的工作量不算很大。MSA 流行之后，分布式应用常常会有二三十个服务甚至上百个服务。运维工程师的工作量不是随服务数量线性增加，而是按照服务数量的平方增加。可想而知，如果不想办法尽量降低运维工作的成本，建造理想的 MSA 就是不切实际的空中楼阁。

聪明的运维工程师和聪明的程序员都懂得 DRY（don't repeat yourself）原则。解决方案只写一次，尽量重用，能自动化完成的工作尽量自动化完成。这个思维

和工作方法叫 DevOps，它已经在运维领域流行了很多年。近 5 年来涌现出了大量的DevOps工具，以及以Docker为代表的轻量级容器，这些新生事物极大地提高了运维工作的自动化程度，使得运维工作的效率有了 10 倍以上的提升。

《微服务运维实战》这套书探讨如何把设计开发 MSA 和 DevOps 两方面的最佳实践结合在一起，它的出版可以说是恰逢其时，因为有很多想要尝试 MSA 的软件开发团队，由于不知道如何做好运维工作，而最终无奈放弃。实施好 MSA 项目不可能一蹴而就，它需要长期的演化迭代。有了《微服务运维实战》这套书的帮助，开发团队可以少踩很多坑，更加顺利地实施 MSA，少走回头路。我向大家强烈推荐这本书，它非常实用，也应该成为正在实施 MSA 项目的所有技术人员的案头书。

李　锟

上海霓风网络科技有限公司 CEO

前言

Preface

我的职业生涯是从程序员开始的。那段日子，我所知道的只是编写代码。我以为出色的软件设计师就是精通编码的人，而精通就是对所选的一种编程语言做到了如指掌。后来，我的想法变了，我开始对不同的编程语言产生兴趣：从 Pascal 换到 Basic，而后换到 ASP。Java 和.NET 让我了解到面向对象编程的好处。Python、Perl、Bash、HTML、JavaScript、Scala，每种编程语言都带来了一些新东西，并教给了我如何以不同的方式思考。我学会了为手头的任务挑选正确的工具。每学会一种新语言，我就感觉距离成为专家又近了一点。我只想成为一名资深程序员，这个想法随着时间的推移而发生了变化。我认识到，如果要把自己的工作做好，我得成为一名**软件艺匠**（software craftsman）。我学习的东西远不止输入代码。有一段时间我痴迷于测试，现在我认为测试是开发不可或缺的一部分。除非有特殊原因，否则我编写的每行代码都是通过测试驱动开发（test-driven development，TDD）来完成的。测试驱动开发已成为我手上必不可少的工具。另外我还认识到，在确定应该做什么时，我必须接近客户并与他们肩并肩地工作。所有这些事情都将我引向软件架构领域。

我在软件行业工作的这些年，没有哪个工具、框架或者实践能像持续集成（continuous integration，CI）以及之后的持续交付（continuous delivery，CD）那样

让我着迷。起初，我以为学会 CI/CD 意味着了解 Jenkins 并且能够书写脚本。随着时间的推移，我认识到 CI/CD 几乎涉及软件开发的方方面面。而我是在付出一定代价后才有了这样的认识。

我不止一次尝试为我开发的应用创建 CI 管道，但都失败了，因为我采用的方法是错误的。不考虑架构问题，CI/CD 是无法实现的。类似的道理亦适用于测试、配置、环境、容错等方面。要成功实施 CI/CD，我们需要做出很多改变，这些改变第一眼看上去似乎没有直接的关联。我们需要从一开始就应用一些模式和实践。我们还得考虑架构、测试、耦合、打包、容错，以及其他许多事情。通过实践 CI/CD，我们正在影响和改善软件开发生命周期的方方面面。

要真正精通 CI/CD，我们需要对运维更加熟悉。DevOps 运动将开发所能带来的优势与传统运维相结合，这是一个显著的改善。但我认为这还不够。如果想要获得 CI/CD 所能带来的全部好处，还需要深入理解架构、测试、开发、运维，甚至客户洽谈，并做出相应的改变。使用 DevOps 这个名字来概括 CI/CD 其实是不合适的，因为 DevOps 不仅关系到开发和运维，还关系到软件开发的所有方面，需要架构师、测试人员，甚至管理者的共同参与。DevOps 将传统运维与开发相结合是一个巨大的进步。就当前的业务需求而言，手工运维几乎是行不通的，而自动化需要开发工作。因此，我认为应该扩展 DevOps 的定义。我本打算将它重新命名为 DevOpsArchTestManageAndEverythingElse，但这个名字过于烦琐而且几乎不可能读出来，因此我使用 DevOps 2.0 来代替。DevOps 2.0 不但要实现运维自动化，而且要让整个系统变得自动化、快速、可扩展、容错、零停机、易于监控。这无法通过某个单一的工具实现，只能通过深入技术层面和流程层面重构整个系统来实现。

概述
Overview

本书介绍快速构建现代软件系统的方法，我们将微服务打包成不可变的容器，通过配置管理工具实现自动化测试和持续部署，同时保证零停机且随时能回滚。设计能够从硬件和软件故障中恢复的自愈系统，采用集中日志对集群进行记录和监控，轻松实现服务器扩展。

换言之，本书采用业界最新的工具和方法开展微服务开发与部署。我们将用到 Docker、Kubernetes、Ansible、Ubuntu、Docker Swarm、Docker Compose、Consul、etcd、Registrator、confd、Jenkins。

最后，本书虽然介绍了很多理论，但仍然需要上手实践。仅仅在上班的地铁上阅读本书是不够的，还得在计算机上亲自实践。如果你需要帮助，或者想就书中内容发表评论，请把你的想法发到 Disqus 的 DevOps 2.0 Toolkit 频道。如果你喜欢一对一地讨论，请发邮件给我（viktor@farcic.com），或者在 HangOuts 上联系我，我会尽全力帮助你。

读者对象
Audience

本书是写给那些对持续部署和微服务感兴趣的专业人士看的，涉及的内容非常宽泛，目标读者包括想了解如何围绕微服务设计系统的架构师、想了解如何应用现代配置管理实践和持续部署容器化应用的 DevOps 人员、希望将整个流程掌控在自己手中的开发人员，以及想要更好地理解软件交付流程的管理人员。我们会谈

及系统的扩展和监控，甚至会设计（并实现）能够从（硬件或软件）故障中自愈的系统。

本书内容涉及从设计、开发、测试、部署到运维的所有阶段。我们介绍的流程是业界最新的最佳实践。

目录

Contents

第1章

DevOps的理想
The DevOps Ideal

能参与小而新的项目确实很棒。我上一次参与这样的项目是在2015年的夏天，尽管这个项目有它自己的问题，但它真的充满乐趣。开发小而相对新的产品让我们可以选择自己喜欢的技术、实践和框架。我们可以使用微服务吗？当然，为什么不用呢？我们可以尝试Polymer和GoLang吗？当然！没有负担拖累，这感觉非常美妙。即使我们做出错误的决策，也最多白干一周而已，绝不会将别人数年的工作置于险境。简言之，没有遗留系统需要考虑和担心。

可惜我大部分的职业生涯并非如此。我有幸，或者说不幸，一直在处理大型遗留系统。我就职的企业在我加入前已存在很长时间，而且不论好坏，它们的系统早已就位。我不得不在创新和改进间进行权衡，同时确保当前业务持续不断地运转。我这些年一直尝试发现改进这类系统的新方法。我也必须承认，多数尝试都以失败告终。

我要谈谈我是如何失败的，以便更好地理解动机，也正是这种动机催生了本书。

1.1 持续集成、交付和部署 Continuous Integration, Delivery, and Deployment

发现 CI 和 CD 时我非常兴奋。以前的集成要花数天到数周不等，甚至是几个月，这是我们都害怕的阶段。不同的团队在不同的服务或应用上开发数月之后，集成的第一天简直就是人间地狱。要不是我知道真相，没准我会认为但丁是一个开发人员，而他写《地狱》时正在做集成的工作。

集成的第一天，我们愁眉苦脸地来到办公室。当集成工程师宣布整个系统已经搭建起来时，底下都在窃窃私语。系统开启后，很可能显示的是白屏——几个月来各自为政的工作被证明是一场灾难。服务与应用无法集成，修复问题的慢慢长路开始了，我们又得再干几周。大家对提前定义的需求一如既往地提出了不同的意见，这种争论在集成阶段变得尤为明显。

后来，极限编程（eXtreme Programming，XP）的各种实践应运而生，持续集成（CI）就是其中之一。集成应当持续不断地进行，这个想法今天听起来似乎显而易见。当然，你不能等到最后时刻才集成。但那时还是瀑布开发的时代，有些事情还不像现在这么明了。我们实现了持续集成流水线，开始检查每次提交，运行静态分析、单元测试和功能测试，打包，部署，开展集成测试。任何一个阶段失败，我们都会放下正在做的事情马上去修复流水线上出现的问题。流水线本身非常快，在某人提交到代码库后的几分钟内，我们就能获得是否失败的通知。后来，持续交付（CD）又出现了，我们以为这下每次提交都能轻易部署到生产环境了。不仅如此，每次构建完毕，再也无需等待任何人的（手工）确认就能完成部署。最妙的是，所有这一切是完全自动化的。

这真是美梦。真的！它就是梦。我们没能现实。为什么会这样？因为我们犯了错。我们以为 CI/CD 是运维部门的任务（今天，我们称之为 DevOps）。我们以为自己能创建一个将应用和服务封装在内的流程，我们以为 CI 工具和框架已经准