Docker生产环境实践指南

作者：[美]乔?约翰斯顿（Joe Johnston） [西]安东尼?巴彻勒（Antoni Batchelli）[英]贾斯汀?科马克（Justin Cormack）[美] 约翰?菲尔德（John Fiedler）[英] 米洛斯?盖多什（Milos Gajdos）

目　录

版权信息

书名：Docker生产环境实践指南

本书由人民邮电出版社发行数字版。版权所有，侵权必究。

您购买的人民邮电出版社电子书仅供您个人使用，未经授权，不得以任何方式复制和传播本书内容。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

- 著　　［美］Joe Johnston　［西］Antoni Batchelli
  ［英］Justin Cormack　［美］John Fiedler
  ［英］Milos Gajdos

  译　　吴佳兴　梁晓勇

  责任编辑　杨海玲

- 人民邮电出版社出版发行　北京市丰台区成寿寺路11号

  邮编　100164　电子邮件　315@ptpress.com.cn

  网址　http://www.ptpress.com.cn

- 读者服务热线：(010)81055410

  反盗版热线：(010)81055315

版权声明

Copyright © 2015 Bleeding Edge Press. All rights reserved. First published in the English language under the title *Docker in Production* by Joe Johnston, Antoni Batchelli, Justin Cormack, John Fiedler, and Milos Gajdos by Bleeding Edge Press, an imprint of Backstop Media.

本书中文简体版由Backstop Media LLC授权人民邮电出版社出版。未经出版者书面许可，对本书的任何部分不得以任何方式或任何手段复制和传播。

版权所有，侵权必究。

内容提要

本书围绕“Docker该如何应用到生产环境”这一核心问题展开。在本书中，读者将接触到多个IT企业应用Docker到生产环境的成功案例，了解Docker实际投产时将会面临的问题，以及它与现有基础设施存在的矛盾与冲突，了解构建Docker生态系统所需的配套设施，包括安全、构建镜像、持续集成/持续交付、镜像存储、配置管理、网络实现、服务发现、持久化存储以及日志监控等模块的具体选型方案及利弊所在。本书编写时一些案例参考的Docker版本是Docker 1.6或Docker 1.7。

本书要求读者具备一定的容器管理和运维的基础知识，适合在生产环境中使用Docker的相关技术人员阅读，尤其适合具有中高级DevOps和运维背景的读者阅读。

对本书的赞誉

经过2015年Docker项目的飞速发展，国内的互联网企业开始关注容器技术在生产环境下的使用案例。但是，由于与Docker相关的技术资料还没有得到系统性地整理，国内企业一直在各种实际场景中进行实践，所以本书的出现正好填补了当前生产环境实践的实际需要，让国内容器技术的推广迈出坚实的一步。

——肖德时，数人科技CTO

2015年，不管是传统IT企业、互联网巨头，还是初创公司，大大小小的公司都在实践如何在生产环境中应用Docker来解决其实际问题。在这个过程中会遇到各种各样的问题，如网络和存储驱动如何选型，资源限制不够彻底怎么办，镜像仓库如何做权限控制，容器内健康状况如何监测，容器有哪些安全隐患，以及在容器集群化过程中会遇到哪些问题，如服务发现、弹性伸缩等。这些问题都没有标准答案，这也激发了工程师们的探索乐趣，于是就有了各种各样的实战方案。

本书恰好就是对一些实战方案的归纳和总结。相对于市面上其他Docker图书，我认为这本书价值更大：对工程师们来说，将技术应用到“生产环境”中才是最大的价值和乐趣所在！

——陈铁飞，原百度PaaS平台负责人，国内最早大规模应用Docker的实践者

我在Google从事基于容器的基础设施和集群管理研发多年，许多关于容器使用最佳实践的知识都是通过“代代相传、口口相传”的方式获得的。在Docker迅速流行的同时，在开源社区里却缺少如Google（或其他公司）内部“老工程师对新人倾囊相授实践+真理”的这种奢侈。

在众多讲述Docker自身原理、使用方法的书中，这本书从生产角度出发，将作者在实战中积累的一线经验系统地汇总成了基于Docker搭建生产系统的经验，值得我们借鉴。

——张鑫，才云科技CEO

在深入学习Docker时将面对这样的问题：Docker的流行催生出大量与之相关的新技术，这些新技术有哪些，它们是什么关系，又该如何正确选择？Docker也有适用场景，到底哪些场景适合用？把Docker用在生产环境，是否有成功案例或最佳实践，又该如何操作？如果你正面对这些问题，本书就非常适合你。另外，本书不仅以可以直接应用的真实生产环境示例贯穿始终，还讲解了技术的权衡之道，是Docker进阶的难得好书。

——刘凡，好雨云创始人

Docker进入大家视野已经有段时间了，也历经了多个重要的版本升级。有报告表明，在一线互联网公司中，已经有27%的企业在生产环节中使用到了Docker，先行者已开始从中获益。但目前Docker资料仍停留在概念或者实验级别，关于真正在生产环境中使用Docker的内容则少之又少，最佳实践方面的信息与资料也十分稀缺。

Docker技术对于IT架构乃至软件架构的改变是巨大的，对于想在生产环境中使用Docker的企业和团队来讲，只掌握概念和基本原理是不够的，如何能使用Docker解决自身问题，获得由此带来的收益，需要更多的生产实践方面的内容。

本书以生产部署为背景，讲述Docker在真实环境中的使用，能够给读者一个很好的参考，进而达到让读者“举一反三”的效果，使其能让自身的IT架构提升到一个新的技术高度。

——周东波，首都在线总工程师

Docker的出现不仅是单一技术的诞生，而是整个开发、测试、运维体系的一次变革，是从传统的底层主机化真正向资源服务化转变。Docker也同时改变了广大技术人员的思维模式，让原先枯燥重复的工作变得有趣。

本书非常系统地讲解了Docker在生产环境的搭建和应用，同时详细地讲解了其镜像、网络、存储、安全等方面的知识，是有志于研究Docker的读者必备的书籍。

——孙斌，去哪儿网运维总监

Docker是互联网领域最近两年最火热的技术词语，没有之一。以Docker为代表的容器技术，通过务实的工程创新，正在影响无数企业的IT基础设施以及主流的公有云服务。国内越来越多的互联网企业开始将Docker应用到生产环境中。可以预见，掌握Docker相关知识即将成为软件工程师或运维工程师的必备技能。DockOne.io社区组织翻译的这本书恰逢其时，值得每名IT从业者阅读。

——刘海锋，京东云平台总架构师

容器技术已经是2015年最热的技术。容器技术已经存在数年，为什么Docker作为容器技术的布道者能够有翻天覆地的影响？为什么容器技术能够给传统应用带来好处？如何把传统应用容器化并通过DevOps方式简洁化？如何通过具体参数和方法解决现实问题？如何使用Docker以及Docker实践？

其实还有很多问题需要解决，如应用的复杂性和难维护性、开发与运维的脱离性，以及成本不断攀升的现实性。本书从简入繁，通过具体案例解释具体问题，通过实践帮助读者理解具体问题，对正在或者即将使用基于容器的DevOps的读者有很多益处。

——陈冉，Linker Networks CTO & VP

Docker的出现使得IaaS和PaaS的界限进一步模糊化，引领了云计算技术变革。时下诸多公司正如火如荼地探索在生产环境中如何玩转Docker。本书的特点在于不仅介绍“是什么”，更进一步探讨“如何用”和“为什么用”；不限于某种特定的框架，而是对比各种不同的方案。书中涵盖容器监控、配置管理、安全、调度、持续交付等生产实践经验，想在生产环境中玩转Docker的技术人员定能从中获益。

——吴毅挺，携程CIS-系统研发部总监

译者介绍

吴佳兴，毕业于华东理工大学计算机系，目前是携程网系统研发团队的一名DevOps工程师，主要研究方向有Python开发、运维自动化、配置管理及PaaS平台的构建等。2014年年底有幸加入DockOne社区，作为译者，利用闲暇时间为社区贡献一些微薄的力量。个人博客devopstarter.info。欢迎邮件联系（wjx_colstu@hotmail.com）。

梁晓勇，毕业于厦门大学，现任某互联网金融公司架构师，DockOne社区编外人员，非著名互联网从业者。长期奋战在技术研发第一线，在网络管理、技术开发、架构设计等方面略有心得。热爱互联网技术，积极投身开源社区，对Docker等容器技术具有浓厚兴趣。欢迎邮件联系（sunlxy@yahoo.com），交流指教。

前言

Docker是基础设施的新成员。很少有新兴技术能像它这样，在DevOps和基础设施领域中快速风靡起来。在不到两年的时间内，Google、亚马逊、微软、IBM以及几乎所有云供应商都宣布支持运行Docker容器。大量与Docker相关的创业公司在2014年和2015年年初都获得了风险资本的投资。Docker开源技术背后的同名公司——Docker公司，在2015年第一季度的D轮融资中估值为10亿美元左右。

大大小小的公司都在转换其应用，使之运行于容器内，以此实现面向服务架构（SOA）和微服务。不论是参加从旧金山到柏林的任何DevOps聚会，还是阅读最热门的公司工程博客，都可以看出全世界的运维领导者们如今都在云上运行Docker。

毫无疑问，容器已经成为应用程序打包和基础设施自动化的重要组成部分。但有一个棘手的问题，促使本书作者和同僚们创作了另一本Docker图书。

本书面向的读者

具有中高级DevOps和运维背景的读者将从本书获益最多。因而，强烈建议读者应具备在生产环境中运行服务器以及创建和管理容器这两方面的基本经验。

很多图书和博客文章已经涵盖了与Docker安装及运行相关的话题，但能把在生产环境中运行Docker时产生的大量甚至是令人挠头的关注点结合在一起的材料则少之又少。不用担心，如果你很喜欢《盗梦空间》（Inception）这部电影，在云服务器的虚拟机中运行容器会让你感觉很自然。

本书将带读者深入理解生产环境中架构的组成部分、关注点，以及如何运行基于Docker的基础设施。

谁真的在生产环境中使用Docker

换个更深刻的说法，对于在真实生产环境中使用Docker遇到的问题，如何找到解决之道？本书综合了访谈、真实公司端到端的生产环境实例，以及来自DevOps杰出专家的参考文献，以此来解答这些问题。虽然本书包含了一些有用的示例，但它并不是一本复制粘贴的“教程式”参考书。相反，本书侧重于生产环境中对前沿技术进行评估、风险抵御及运维所需的实践理论和经验。

作为作者，我们希望这本书所包含的内容能够为那些正在评估如何及何时将Docker相关技术引入其DevOps栈的团队提供一个可靠的决策指南，这远比代码片段要来得长久。

生产环境中运行的Docker为企业提供了多个新的运行和管理服务器端软件的方式。很多现成的用例讲解了如何使用Docker，但很少有公司公开分享过他们的全栈生产环境经验。本书汇集了作者在生产环境中运行Docker的多个实例和一组选定的友好公司分享的使用经验。

为什么使用Docker

Docker所使用的底层容器技术已经存在了很多年，甚至早于dotCloud这家平台即服务（PaaS）创业公司，即后来我们所熟知的Docker。在dotCloud之前，许多知名的公司（如Heroku和Iron.io）已经在生产环境中运行大型容器集群，以获取额外的超越虚拟机的性能优势。与虚拟机相比，在容器中运行软件赋予了这些公司秒级而非分钟级的实例启动与停止的能力，同时能使用更少的机器运行更多实例。

既然这项技术并不新鲜，为什么Docker能获得如此巨大的成功呢？主要是因为它的易用性。Docker创造了一种统一的方式，通过简便的命令行及HTTP API工具来打包、运行和维护容器。这种简化降低了将应用程序及其运行时环境打包成一个自包含镜像的入门门槛，使之变得可行且有趣，而不需要类似Chef、Puppet及Capistrano之类的配置管理和发布系统。

Docker提供了一种统一手段，将应用程序及其运行时环境打包到一个简单的Dockerfile里，这从根本上改变了开发人员与DevOps团队之间的交互界面。从而极大简化了开发团队与DevOps之间的沟通需求与责任边界。

在Docker出现之前，各个公司的开发与运维团队之间经常会爆发史诗般的战争。开发团队想要快速前进，整合最新版的软件及依赖，以及持续部署。运维团队则以保证稳定为己任，他们负责把关可以运行于生产环境中的内容。如果运维团队对新的依赖或需求感到不适，他们通常会站在保守的立场上，要求开发人员使用旧版软件以确保糟糕的代码不会搞垮整台服务器。

Docker一下子改变了DevOps的决策思维，从“基本上说不”变成了“好的，只要运行在Docker中就可以”，因为糟糕的代码只会让容器崩溃，而不会影响到同一服务器上的其他服务。在这种泛型中，DevOps有效地负责为开发人员提供PaaS，而开发人员负责保证其代码能正常运行。如今，很多团队将开发人员加入到PagerDuty中，以监控他们在生产环境中的代码，让DevOps和运维人员专注于平台的稳定运行及安全。

开发环境与生产环境

对大多数团队而言，采用Docker是受开发人员更快的迭代和发布周期需求推动的。这对于开发环境是非常有益的，但对于生产环境，在单台宿主机上运行多个Docker容器可能会导致安全漏洞，这一点我们将在第6章“安全”中讲述。事实上，几乎所有关于在生产环境中运行Docker的话题都是围绕着将开发环境与生产环境区分开的两个关注点进行的：一是编排，二是安全。

有些团队试图让开发环境和生产环境尽可能保持一致。这种方法看起来很好，但是限于开发环境这样做所需定制工具的数量又或者说模拟云服务（如AWS）的复杂度，这种方法并不实际。

为了简化这本书的范畴，我们将介绍一些部署代码的用例，但判定最佳开发环境设置的实践机会将留给读者。作为基本原则之一，尽量保持生产环境和开发环境的相似性，并使用一个持续集成/持续交付（CI/CD）系统以获取最佳结果。

我们所说的“生产环境”

对于不同的团队，生产环境意味着不同的东西。在本书中，我们所说的生产环境是指真实客户用于运行代码的环境。这是相对于开发环境、预演环境及测试环境而言的，后者的停机时间不会被客户感知到。

在生产环境中，Docker有时是用于接收公共网络流量的容器，有时则是用于处理来自队列负荷的异步的后台作业。不管哪种用途，在生产环境中运行Docker与在其他环境中运行相比，最主要的差异都是需要在其安全性与稳定性上投入较多的注意力。

编写本书的动力之一是，与Docker相关的文档和博客文章中缺乏对实际生产环境与其他环境的明确区分。我们认为，80%的Docker博客文章中的建议在尝试在生产环境中运行6个月之后会被放弃（或至少修改）。为什么？因为大多数博客文章中举的都是理想化的例子，使用了最新、最好用的工具，一旦某个极端的情况变成了致命缺陷，这些工具将被遗弃（或延期），被更简单的方法所取代。这是Docker技术生态系统现状的一个反映，而非技术博客的缺陷。

总的来说，生产环境很难管理。Docker简化了从开发到生产的工作流程，但同时增加了安全和编排的复杂度（更多关于编排的内容参见第4章）。

为了节省时间，下面给出本书的重点综述。

所有在生产环境中运行Docker的团队，都会在传统的安全最佳实践上做出一项或多项妥协。如果无法完全信任容器内运行的代码，那么就只得选用容器与虚拟机一对一的拓扑方式。对于很多团队而言，在生产环境中运行Docker的优势远远大于其带来的安全与编排问题。如果遇到工具方面的问题，请等待一到两个月，以便Docker社区对其进行修复，不要浪费时间去修补其他人的工具。保持Docker设置最小化。让一切自动化。最后，对成熟的编排工具（如Mesos、Kubernates等）的需求远比想象的要少得多。

功能内置与组合工具

Docker社区一个常见的口头禅是“电池内置但可移除”，指的是将很多功能捆绑在一起的单体二进制文件，这有别于传统Unix哲学下相对较小、功能单一、管道化的二进制文件。

这种单体式的做法是由两个主要因素决定的：（1）使Docker易于开箱即用；（2）Golang缺少动态链接。Docker及多数相关工具都是用Google的Go编程语言编写的，该语言可以简化高并发代码的编写与部署。虽然Go是一门出色的编程语言，但用它来构建的Docker生态系统中也因此迟迟无法实现一个可插拔的架构，在这种架构中可以很容易用替代品对工具进行更换。

如果读者有Unix系统背景，最好是编译自己的精简版Docker守护进程，以符合生产环境的需求。如果读者有开发背景，预计到2015年下半年，Docker插件将成为现实[1]。在此期间，估计Docker生态系统中的工具将会出现明显的重叠现象，某些情况下甚至是相互排斥的。

换句话说，要让Docker运行于生产环境中，用户的一半工作将是决定哪些工具对自己的技术栈最有意义。与DevOps所有事情一样，先从最简单的解决方案入手，然后在必要时增加其复杂性。

2015年5月，Docker公司发布了Compose、Machine及Swarm，与Docker生态系统内的同类工具进行竞争。所有这些工具都是可选的，请根据实际情况对其进行评估，而不要认为Docker公司提供的工具就一定是最佳解决方案。

探索Docker生态系统时的另一项关键建议是：评估每个开源工具的资金来源及其商业目标。目前，Docker公司和CoreOS经常发布工具，以争夺关注度和市场份额。一个新工具发布后，最好等上几个月，看看社区的反应，不要因为它看起来很酷就切换到最新、最好用的工具上。

哪些东西不要Docker化

最后一个关键点是，不要期望能在Docker容器中运行所有东西。Heroku风格的“十二要素”（12 factor）应用是最容易Docker化的，因为它们不维护状态。在理想的微服务环境中，容器能在几毫秒内启动、停止而不影响集群的健康或应用程序的状态。

类似ClusterHQ这样的创业公司正着手实现Docker化数据库和有状态的应用程序，但眼下，由于编排和性能方面的原因，可能需要继续直接在虚拟机或裸机上运行数据库。

Docker还不适用于任何需要动态调整CPU和内存要求的应用[2]。允许动态调整的代码已经完成，但尚不清楚何时才能在一般的生产环境中投入使用。目前，若对容器的CPU和内存的限制进行调整，需要停止并重新启动容器。

另外，对网络吞吐量有高要求的应用进行最佳优化时不要使用Docker，因为Docker使用iptables来完成宿主机IP到容器IP的NAT转换。通过禁用Docker的NAT来提升网络性能是可行的，但这是一个高级的使用场景，很少有团队会在生产环境中这么做。

技术审稿人

衷心感谢以下技术审稿人提供的早期反馈及细致的评论：Mika Turunen、Xavier Bruhiere和Felix Rabe。

---

[1] Docker 1.7版中正式引入了插件系统。——译者注

[2] Docker 1.10版中新增的docker update命令可实现CPU和内存的动态调整。——译者注

第1章 入门

建立Docker生产环境系统的首要任务，是以一个有助于想象各组件如何相互配合的方式来理解其术语。与其他快速发展的技术生态系统一样，我们可以预见，Docker野心勃勃的市场营销、不完善的文档以及过时的博客文章将造成使用者对各个工具职责理解上的混乱。

我们将在本章中定义贯穿全书的术语和概念，而非提供一份统一的Docker百科全书。通常情况下，我们的定义与生态系统中的大体一致，但如果你所阅读的博客文章中使用了不同的术语也不用太过惊讶。

在本章中，我们将介绍在生产环境中运行Docker的核心概念以及不涉及具体技术的容器常识。在随后的章节中，我们将讨论真实世界的生产环境用例，并详细说明其组件和供应商信息。

1.1 术语

下面让我们来看一下本书所采用的Docker术语。

1.1.1 镜像与容器

- 镜像是指文件系统快照或tar包。
- 容器是指镜像的运行态。

1.1.2 容器与虚拟机

- 虚拟机持有整个操作系统和应用程序的快照。
- 虚拟机运行着自己的内核。
- 虚拟机可以运行Linux之外的其他操作系统。
- 容器只持有应用程序，不过应用程序的概念可以延伸到整个Linux发行版。
- 容器共享宿主机的内核。
- 容器只能运行Linux，不过在同一宿主机上运行的每个容器都可包含不同的发行版。

1.1.3 持续集成/持续交付

在应用程序新代码提交或触发其他条件时，系统自动构建新镜像并进行部署。

1.1.4 宿主机管理

设置/配备一台物理服务器或虚拟机以便用于运行Docker容器的过程。

1.1.5 编排

编排（orchestration，也称编配）这个术语在Docker生态系统中有多种含义。通常情况下，它包括调度和集群管理，不过有时也包括了宿主机管理。

在本书中，我们将编排作为一个松散的总称，包括容器调度的过程、集群的管理、容器的链接（发现），以及网络流量路由。或者换句话说，编排是个控制器进程，用于决定在哪里运行容器，以及如何让集群知道可用的服务。

1.1.6 调度

用于决定哪些容器可以以给定的资源约束（如CPU、内存和IO）运行在哪些宿主机上。

1.1.7 发现

容器如何公开服务给集群，以及发现如何查找其他服务并与之通信的过程。举个简单的用例：一个网站应用容器发现如何连接到数据库服务。

Docker文档中的发现是指将容器链接在一起，不过在生产级系统中，通常使用的是更复杂的发现机制。

1.1.8 配置管理

配置管理过去常常指的是Docker出现之前的自动化工具，如Chef和Puppet。大多数的DevOps团队正在转移到Docker上，以消除这类配置管理系统的复杂度。

在本书的示例中，配置管理工具只用于配备具有Docker和少量其他东西的宿主机。

1.2 从开发环境到生产环境

本书着重于生产环境或非开发环境中的Docker，这意味着我们不会花太多的篇幅在开发环境中Docker的配置和运行上。但由于所有服务器都在运行代码，如何看待在Docker和非Docker系统中的应用程序代码还是值得简单讨论一下的。

与Chef、Puppet和Ansible这类传统配置系统不同，Docker最好的使用方式是将应用程序代码预先打包成一个Docker镜像。镜像通常包含所有的应用程序代码、运行时的依赖以及系统的需求。而包含数据库凭证和其他敏感信息的配置文件通常在运行时添加，而非内建到镜像中。

有些团队会在开发机上手工构建Docker镜像，然后推送到镜像仓库，之后再从仓库中拉取镜像到生产环境宿主机中。这是个很简单的用例。虽然行得通，但从工作流和安全角度考虑并不理想。

一个更常见的生产环境示例是，使用持续集成/持续交付系统在应用程序代码或Dockerfile文件发生变更时自动构建新镜像。

1.3 使用Docker的多种方式

过去的几年时间，科技发生了巨大变化，从物理服务器到虚拟服务器，再到拥有PaaS环境的云计算。不论是否采用了全新架构，Docker镜像都可以在当前环境中很容易地被使用。要使用Docker，并不需要立即从单体应用程序迁移到面向服务架构。有很多用例允许在不同层次上集成Docker。

Docker常用于以下场景。

- 使用以镜像为基础的部署方式取代类似Capistrano的代码部署系统。
- 安全地在同一台服务器中运行遗留应用和新应用。
- 使用一个工具链循序渐进地迁移到面向服务架构。
- 管理云端或裸机上的水平扩展性和弹性。
- 确保从开发环境到预演环境到生产环境跨环境的一致性。
- 简化开发人员的机器设置和一致性。

将应用的后台程序迁移到Docker集群中，同时保持网页服务器和数据库服务器不变是开始使用Docker的常见示例。另一示例是将应用的部分REST API迁移到Docker中运行，前端使用Nginx代理在遗留服务和Docker集群之间路由通信。通过使用此类技术，团队可以渐进式地从单体应用无缝地迁移到面向服务架构。

欢迎访问：电子书学习和下载网站（https://www.shgis.cn）
文档名称：《Docker生产环境实践指南》[美]乔_约翰斯顿（Joe Johnston） [西]安东尼_巴彻勒
请登录 https://shgis.cn/post/271.html 下载完整文档。
手机端请扫码查看: