Python Docker 实战

[美] 萨蒂亚吉斯·巴哈(Sathyajith Bhat) 著

蒲 成 译

清华大学出版社

北 京

Practical Docker with Python: Build, Release and Distribute your Python App with Docker
By Sathyajith Bhat
EISBN：978-1-4842-3783-0

北京市版权局著作权合同登记号　图字：01-2019-2123

图书在版编目(CIP)数据

Python Docker 实战 / (美)萨蒂亚吉斯 • 巴哈(Sathyajith Bhat) 著；蒲成 译. —北京：清华大学出版社，2019

书名原文: Practical Docker with Python: Build, Release and Distribute your Python App with Docker

ISBN 978-7-302-52761-9

Ⅰ. ①P… Ⅱ. ①萨… ②蒲… Ⅲ. ①Linux 操作系统－程序设计 Ⅳ. ①TP316.85

中国版本图书馆 CIP 数据核字(2019)第 070971 号

责任编辑：王　军　韩宏志
封面设计：孔祥峰
版式设计：思创景点
责任校对：牛艳敏
责任印制：丛怀宇

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**：5.25　　**字　　数**：146 千字
版　　次：2019 年 5 月第 1 版　　**印　　次**：2019 年 5 月第 1 次印刷
定　　价：59.80 元

产品编号：082796-01

译 者 序

目前，最流行的容器化工具可能就是 Docker 了，不管是 DevOps 还是微服务，你都离不开 Docker 这一开源的应用容器引擎。它使得开发者可将应用以及依赖包打包到一个可移植的容器中，然后发布到任何流行的 Linux、Windows、macOS 机器上，也可以实现虚拟化，Docker 完全使用沙箱机制，相互之间不会有任何接口。

Docker 技术将应用以集装箱的方式打包交付，使应用在不同的团队中共享，让应用通过镜像的方式可以部署于任何环境中。这样就避免了各团队之间的协作问题的出现，成为企业实现 DevOps 目标的重要工具。以容器方式交付的 Docker 技术支持不断地开发迭代，大大提升了产品的开发和交付速度。

本书以一个 Python 聊天机器人作为贯穿全书的示例，通过对这一聊天机器人构建过程的讲解串联起 Docker 容器化的基础内容。通过阅读本书，读者将了解 Docker 的相关基础知识，理解其专用术语，并亲手实践如何将一个传统应用程序打包成 Docker 镜像。对于刚开始接触 Docker 并且想要系统地理解 Docker 基础知识结构的读者而言，本书将是最好的入门指南。

本书从 Docker 容器化基本概念开始讲解，逐步介绍 Docker 技术架构中的要点，为读者描绘了一个清晰的 Docker 容器化结构环境，为

读者进一步深入学习 Docker 应用技术打下了坚实基础。就其技术原理而言，Docker 并非具有很陡峭的学习曲线，只要读者遵循本书的节奏，彻底理解本书中所讲解的概念并对本书内容中的每个练习都亲自动手进行实践，就能完全掌握 Docker 这一容器化应用技术。

在此要特别感谢清华大学出版社的编辑们，在本书翻译过程中他们提供了颇有助益的帮助，没有其热情付出，本书将难以付梓。

本书内容由蒲成翻译，参与翻译活动的还有何东武、李鹏、李文强、林超。

虽然译者将大量心血倾注到本书的翻译中，但难免会出现一些错误或翻译不准确的地方，如果读者能够指出并勘正，译者将不胜感激。

译　者

作者简介

Sathyajith Bhat 是一位经验丰富的 DevOps/SRE 专家，目前担任 Adobe I/O 的 DevOps 工程师。Sathyajith 此前担任 Styletag.com 的首席 Ops/SRE；还曾担任 CGI 的首席分析师，使用 Oracle Fusion 栈(Oracle DB/PL/SQL/Oracle Forms 及相关中间件)为北欧的一家大型保险公司设计、构建和实现完整的端到端解决方案。

Sathyajith 是 Barcamp Bangalore 规划小组的成员，负责处理 DevOps 和社交媒体事务。他还是 AWS Users Group Bangalore 的组织者之一，是 Super User and Web Apps Stack Exchange 的社区志愿版主。Sathyajith 也曾担任 Chip-India 与 Tech 2 论坛的版主。

读者可通过以下方式联系 Sathyajith。

电子邮件：sathya@sathyasays.com

博客：https://sathyasays.com

Twitter：https://twitter.com/sathyabhat

LinkedIn：https://linkedin.com/in/sathyabhat

技术编辑简介

Swapnil Kulkarni 是一位云端架构师和开源技术的狂热支持者，他的经验很丰富，涉及的领域包括区块链、原生云解决方案、容器，以及企业软件产品架构。在各种不同的私有化、混合云架构方面，他具有多样化的经验，涉猎的平台包括 Amazon Web Services、Azure、OpenStack、CloudStack 和 IBM Cloud 等。他是 OpenStack 的活跃技术贡献者，参与了多个项目。他也是 OpenStack Kolla 和 OpenStack Requirements Project 的核心审查者。他对各个开源社区都做出了贡献，其中包括 OpenStack、Docker 和 Kubernetes，并且打造了多个即将发布的容器化开源平台。Swapnil 还多次出席 OpenStack 技术峰会——LinuxCon 以及 ContainerCon 等。Swapnil 在其博客上分享他的技术观点和技术尝试，后面会提供其博客地址。他也担任多家出版社的技术编辑，对于学习和实现不同的理念充满了热情。读者可通过电子邮件来联系他，或在其社交媒体主页上与他进行交流。

电子邮件：toswapnilkulkarni@gmail.com

博客：https://cloudnativetech.wordpress.com

Twitter：https://twitter.com/coolsvap

LinkedIn：https://www.linkedin.com/in/coolsvap

致　　谢

感谢我的妻子 Jyothsna，谢谢她在我的职业生涯以及编写本书期间给予我足够的耐心并且毫无保留地支持我。

我要感谢 Apress 出版社的 Nikhil Karkal、Siddhi Chavan 和 Divya Modi，在本书的整个编写过程中他们为我提供了极大的帮助。

谢谢技术编辑 Swapnil Kulkarni，他给我提供了极具参考性的意见。

我还要感谢 Saurabh Minni、Ninad Pundalik、Prashanth H. N.、Mrityunjay Iyer 和 Abhijith Gopal 在过去数年中对我的鼎力支持。

前　言

Docker 的人气一路飙升，并且已经成为容器化镜像格式以及容器化运行时的实际标准。现代应用程序正变得越来越复杂，你对于微服务的日益重视促进了 Docker 的广泛应用，因为 Docker 允许将应用程序及其依赖项打包到一个文件中，作为可以运行在任何系统上的容器。这就使得应用程序部署的周期更短并且复杂性更低，也避免了出现“南橘北枳”的问题。

本书涵盖了容器化的基础内容，可以让读者熟悉 Docker 的相关知识，剖析了像 Dockerfile 和 Docker 卷这样的专用术语，并将指导读者使用 Python 构建一个聊天机器人。本书讲解如何将一个传统应用程序打包成一个 Docker 镜像。

本书内容结构

本书被划分成 7 章——第 1 章简要介绍 Docker 和容器化。然后，第 2 章讲解 Docker 的入门知识，其中包括安装、配置和一些 Docker 术语。第 3 章讲解项目以及如何配置聊天机器人。

第 4～6 章深入探讨 Docker 的主要知识点，重点介绍 Dockerfile、Docker 网络及 Docker 卷。这几章提供了关于如何将这些知识点纳入

到项目之中的实践练习。最后，将介绍 Docker Compose 并讲解如何才能运行多容器应用程序。

源代码下载

通过 GitHub 获得本书的源代码以及其他补充材料，网址为 https://github.com/Apress/practical-docker-with-python。另外，也可扫描封底的二维码直接下载。

目　录

第 1 章

容器化简介

本章将讲解什么是 Docker，还将介绍容器化的概念及其与虚拟化的区别。

1.1 什么是 Docker

在我们回答这个问题时，需要澄清一下“Docker”这个词，因为 Docker 已经变成容器的代名词了。

1.1.1 Docker 公司

Docker 公司运营着 Docker，它的前身是 Solomon Hykes 于 2010 年创建的 dotCloud 公司。dotCloud 的工程师为 Linux Containers 构建抽象概念和工具，并使用 Linux 内核的 cgroups 特性和命名空间来降低使用 Linux 容器的复杂性。dotCloud 将其工具开源并将业务重心从 PaaS 转移到容器化上。后来，Docker 公司将 dotCloud 出售给 cloudControl 公司(该公司最终申请了破产)。

1.1.2 软件技术层面的 Docker

Docker 是一种技术，用于提供操作系统级别的虚拟化，也就是你所熟知的概念——容器。值得一提的是，这与硬件虚拟化并不一样。稍后将探讨这一点。Docker 使用了 Linux 内核的资源隔离特性，比如 cgroups、内核命名空间(kernel namespace)及 OverlayFS，这些全都位于同一台物理机或虚拟机中。OverlayFS 是一种可联合的文件系统，它将一些文件和目录合并成一个，以便运行多个相互隔离和相互包含的应用程序，而这全都在同一台物理机或虚拟机中完成。

1.1.3 理解 Docker 所解决的问题

一直以来，安装开发人员的工作站对于系统管理员而言都是一项非常麻烦的任务。即便有了开发工具的完全自动化安装，在面对具有多个不同的操作系统、操作系统的多个不同版本，以及不同版本的库和编程语言的情况时，设置能够持续提供一致体验的工作环境几乎也是不可能的。通过减少活动部件数量的方式，Docker 在很大程度上解决了这个问题。相对于应对操作系统和程序版本的问题，现在我们只要关注 Docker 引擎和运行时即可。Docker 引擎提供了底层系统的统一抽象，这使得开发人员可以非常容易地测试自己的代码。

在生产环境中，事情会变得更复杂。假设我们有一个用 Python 编写的 Web 应用程序，它运行在 Python 2.7 环境中的 Amazon Web Services EC2 实例上。为让代码库变得现代化，应用程序已经进行了一些重大升级，其中包括 Python 版本从 2.7 升级为 3.5。假设当前运行现有代码库的 Linux 发行版，其所提供的包中并没有这个版本的 Python；那么现在就要部署这个新的 Python 版本，可以采用以下任一途径来部署。

- 替换现有实例。

- 通过以下方式设置 Python 解释器：
  - 将 Linux 发行版的版本变更为包含较新 Python 包的版本。
  - 添加提供了较新 Python 版本的打包版的第三方通道。
  - 执行就地升级，保留现有 Linux 发行版的版本。
  - 基于包含额外依赖项的源码来编译 Python 3.5。
  - 使用诸如 virtualenv 的工具，不过它们也都有自身的优缺点。

无论从哪个角度看，应用程序代码的新版本部署都会带来大量的不确定性。作为运维工程师，将变更限定在配置层面是至关重要的。源于操作系统、Python 版本以及应用程序代码的变更，会导致大量的不确定性。

Docker 通过显著减少不确定性的影响面解决了这个问题。想让我们的应用程序变得现代化？没问题。在保持现有基础设施不变的情况下，我们可以构建一个装有新应用程序代码和依赖项的新容器并最终交付它。如果该应用程序并未如预期般运行，那么只要简单地重新部署旧容器即可实现回滚——在一个 Docker 注册服务器中存储所有生成的 Docker 镜像的做法并不少见。这种在回滚时不干扰当前基础设施的简单做法会显著缩短故障响应所需的时间。

1.2　容器化历程

在过去几年中，尽管容器化技术得到了稳步发展并且持续火爆，但其实容器化的概念早在 20 世纪 70 年代就已被提出。

1.2.1　1979 年：chroot

1979 年，Version 7 UNIX 中引入了 chroot 这一系统调用。chroot 最初用于改变当前运行进程及其子进程的显式根目录。在 chroot 中初始化的进程无法访问所分配目录树之外的文件。这一环境被称为

chroot jail。

1.2.2 2000 年：FreeBSD jail

FreeBSD 基于 chroot 概念扩展而来，它增加了对一个新特性的支持，该特性允许将 FreeBSD 系统划分成数个被称为 jail 的独立的、隔离的系统。每个 jail 都是宿主机系统上的一个虚拟环境，有它自己的一套文件、进程和用户账户。chroot 仅将进程活动范围限制在文件系统的一个视图中，而 FreeBSD jail 则限制受限进程的活动范围为整个系统，包括为其绑定的 IP 地址范围。对于测试联网软件的新配置而言，FreeBSD jail 就成了一种理想方法，因为这样一来，我们就能轻易地测试不同的配置而不会让来自 jail 的变更影响到外部的主系统。

1.2.3 2005 年：OpenVZ

在为低端虚拟专用服务器(Virtual Private Server，VPS)提供商提供操作系统虚拟化方面，OpenVZ 曾经非常流行。OpenVZ 允许一台物理服务器运行多个隔离的操作系统实例(即容器)。OpenVZ 使用一个打过补丁的 Linux 内核，从而将该内核与所有容器共享。每个容器都充当一个独立实体，并且有自己的一套文件、用户、分组、进程树及虚拟网络设备。

1.2.4 2006 年：cgroups

cgroups(control groups，控制群组)最初被称为进程容器，它是由 Google 工程师发起构建的。cgroups 是一个 Linux 内核特性，它将资源(如 CPU、内存、磁盘 I/O 和网络)的使用限制并隔离到一个进程集合中。cgroups 已被重新设计过多次，每次重新设计都归因于其不断增长的用例数量和所需特性。

1.2.5　2008 年：LXC

通过整合 Linux 内核的 cgroups 以及对隔离命名空间的支持，LXC 就能为应用程序提供一个隔离环境，从而提供操作系统级别的虚拟化。Docker 最初使用 LXC 来提供隔离特性，但随后切换到它自己的库。

1.3　理解容器和虚拟机之间的区别

许多人都认为，既然容器隔离了应用程序，那么与虚拟机没区别。乍一看似乎是这样，但根本性的区别在于，容器与宿主机共享相同的内核。

Docker仅会隔离单一进程(或者一组进程，这取决于构建镜像的方式)，并且所有容器都运行在相同的宿主机系统上。由于其隔离性应用在内核级别，因此相对于虚拟机而言，容器的运行并不会导致宿主机上的较大开销。当一个容器运行时，所选的一个或一组进程仍旧会运行在相同的宿主机上，而无须虚拟化或模拟任何东西。图 1-1 显示了运行在单台物理宿主机上三个不同容器中的三个应用。

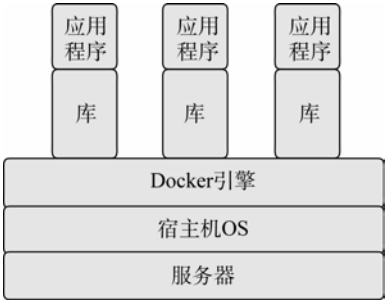

图 1-1　运行在三个不同容器中的三个应用程序的图示

相比之下，当虚拟机运行时，虚拟机管理程序会虚拟化整个系统——从 CPU 到 RAM，再到存储。为了支撑这个虚拟化系统，需要

安装整个操作系统。针对所有的实际用途而言，虚拟化系统就是运行在一台物理计算机中的完整虚拟化计算机。如果能够想象出运行单个操作系统需要多大的资源开销，那么想想运行一个嵌套的操作系统时的资源开销会是多大！图 1-2 显示了运行在单台物理宿主机上三个不同虚拟机中的三个应用程序。

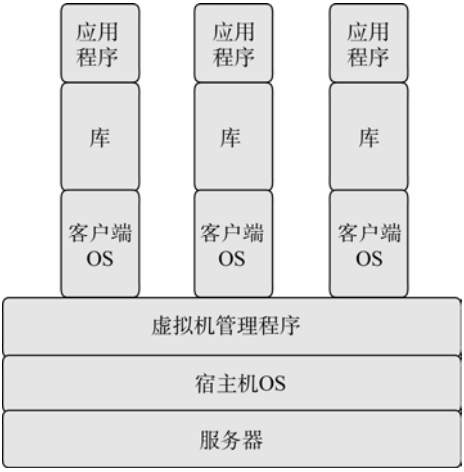

图 1-2　运行在三个不同虚拟机中的三个应用程序的图示

图 1-1 和图 1-2 表明了运行在单一宿主机上的三个不同应用程序。在 VM 的示例中，我们不仅需要应用程序的依赖库，还需要一个操作系统来运行该应用程序。相比之下，使用容器的话，对应用程序共享宿主机 OS 内核则意味着免除了额外的 OS 开销。这不仅显著提升了性能，还可以提高资源利用率并将计算能力的消耗浪费降至最低。

1.4　本章小结

本章介绍了一些关于 Docker 公司、Docker 容器及容器(与虚拟机做对比)的内容，还介绍了一些与容器试图解决的现实环境问题有关的内容。下一章将讲解 Docker 并完成两个关于构建和运行容器的练习。

第 2 章

Docker入门

既然你已经理解了什么是 Docker以及它受欢迎的原因，那么这一章我们就来介绍一些与 Docker 相关的基本术语知识。在这一章中，你将了解如何安装 Docker 并将学习一些 Docker 术语，比如镜像、容器、Dockerfile 以及 Docker Compose。此外，还会使用一些简单的 Docker 命令来创建、运行和停止运行 Docker 容器。

2.1 安装 Docker

Docker 支持 Linux、macOS 和 Windows 平台。在大多数平台上，Docker 的安装都很简单，稍后将介绍。Docker 公司提供了 Docker 平台的社区版和企业版。企业版具有与社区版相同的特性，但企业版提供了额外的支持以及经过认证的容器、插件和基础设施。对于本书的目标以及大多数通用开发和生产用途而言，社区版就够用了，因此我们将在本书中使用这一版本。