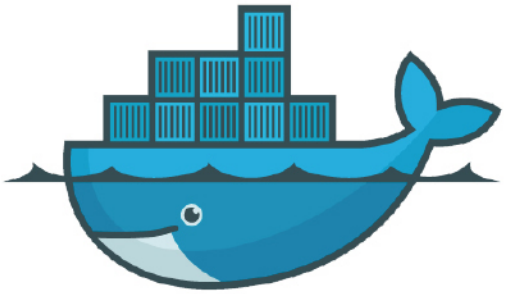
HZ BOOKS
华章科技

中国首部 Docker 著作，一线 Docker 先驱实战经验结晶，来自IBM和新浪等多位技术专家联袂推荐！

结合企业生产环境，深入浅出地剖析 Docker 的核心概念、应用技巧、实现原理以及生态环境，为解决各类问题提供了有价值的参考。

實戰



杨保华 戴王剑 曹亚仑 编著

Docker Primer

Docker
技术入门与实战

机械工业出版社
China Machine Press

实战

Docker 技术入门与实战

杨保华　戴王剑　曹亚仑　编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Docker 技术入门与实战 / 杨保华，戴王剑，曹亚仑编著 . —北京：机械工业出版社，2014.12
（实战）

ISBN 978-7-111-48852-1

I. D…　II. ①杨…　②戴…　③曹…　III. Linux 操作系统　IV. TP316.89

中国版本图书馆 CIP 数据核字（2014）第 287910 号

在云计算时代，开发者将应用转移到云上已经解决了硬件管理的问题，然而软件配置和管理相关的问题依然存在。Docker 的出现正好能帮助软件开发者开阔思路，尝试新的软件管理方法来解决这个问题。通过掌握 Docker，开发人员便可享受先进的自动化运维理念和工具，无需运维人员介入即可顺利运行于各种运行环境。

本书分为三大部分：Docker 入门、实战案例和高级话题。第一部分（第 1 ~ 8 章）介绍 Docker 与虚拟化技术的基本概念，包括安装、镜像、容器、仓库、数据管理等；第二部分（第 9 ~ 17 章）通过案例介绍 Docker 的应用方法，包括与各种操作系统平台、SSH 服务的镜像、Web 服务器与应用、数据库的应用、各类编程语言的接口、私有仓库等；第三部分（第 18 ~ 21 章）是一些高级话题，如 Docker 核心技术、安全、高级网络配置、相关项目等。

本书从基本原理开始入手，深入浅出地讲解 Docker 的构建与操作，内容系统全面，可帮助开发人员、运维人员快速部署应用。

Docker 技术入门与实战

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：吴　怡　　责任校对：董纪丽
印　　刷：　　版　　次：2015 年 1 月第 1 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：19.5
书　　号：ISBN 978-7-111-48852-1　　定　　价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

Foreword 序一

信息技术领域一直试图解决的核心问题之一是提供强大的计算能力。在过去很长一段时间里，我们可以依靠硬件性能的提升来提高物理计算资源的能力，提升处理器的主频或者增加每个处理器里面的处理核心的数量。然而这个时代随着摩尔定律无法胜过物理定律而不得不终结。

在云计算时代，信息技术所面临的难题则截然不同。分布式、虚拟化、大数据……每一项挑战都不是仅仅依靠硬件或软件的局部优化就能得到解决的，更需要通过高效的资源利用来“压榨”计算平台的每一点运算能力。

作为充分发掘计算平台能力的 Linux 容器虚拟化技术，在近些年得到广泛的关注和发展。从早期 IBM 发起的 LXC 项目，到今天如火如荼的 Docker 项目，这些不断涌现的创新项目给计算模式本身带来了巨大的变革。市面上关于传统虚拟机相关技术的书籍有不少，但是探讨容器虚拟化技术的著作寥寥无几。尽管互联网上已经出现了很多关于容器虚拟化、Docker 的文章，但这些文章或过于简略，或仅关注某个技术方面，总体上缺少系统化的从概念、到实现、到如何使用的介绍。这给广大信息产业从业人员了解最新的技术潮流带来了不小的障碍。

值得庆幸的是，能够第一时间拜读这本《 Docker 技术入门与实战》。作为国内开发者撰写的首本探讨 Docker 容器虚拟化技术的书籍，一方面它深入浅出地讲解了 Docker 应用的诸多话题，包括围绕镜像、容器、仓库等核心概念如何来实现“ Build、Ship、Run”的高效流程；另一方面，难能可贵的是书中提供了大量翔实的实战案例，涵盖 DevOps 领域的典型场景。无论是 Docker 技术的初学者还是业内的一线研发人员或资深专家，本书都值得一阅。

作者之一的杨保华博士在加入 IBM 之后，一直从事云计算与软件定义网络领域的相关解决方案和核心技术的研发，热心关注 OpenStack、Docker 等开源社区，热衷使用开源技术，并积极参与开源社区的讨论、积极提交代码。这使得他既能从宏观上准确把握 Docker 技术在整个云计算产业中的定位，又能从微观上清晰理解技术人员所渴望获知的核心之处。

如果你只是 Docker 的初学者，阅读本书，或许并不能让你立刻成为行业内的专业人士，但一定能让你马上体会 Docker 技术所带来的众多优势。如果你已经开始使用 Docker，阅读本书也可以帮助你解答实践中的一些问题，帮助你更恰当地使用 Docker 技术。本书深入浅出，讲解到位，是一本值得常置案头的好书。

刘天成
IBM 中国研究院，云计算运维技术研究组经理
2014 年 11 月

Foreword 序二

最近的几年，云计算是计算机与互联网界的焦点，它的广泛应用离不开虚拟化技术的支持。作为 Linux 下的容器虚拟化工具，Docker 以其轻便易用而受人关注。

这本书向读者清晰地介绍了 Docker 这个虚拟化工具；详细比较了 Docker 和传统虚拟机在组织架构、实现技术和性能上的差异。在此基础上，本书围绕着镜像、容器、仓库三个部分，从实践的角度出发，讲解了 Docker 的安装、配置、使用的方式。在本书的后面几个章节，也介绍了许多 Docker 的实现细节和工作原理。总体而言，本书从实际的案例入手，由浅至深，循序渐进，内容相当丰富。

对于正在寻找虚拟化工具的用户来说，年轻而有活力的 Docker 项目绝对是首选。而如果你正在使用或打算使用 Docker，或者想学习一些新的技术以丰富自己，那就一定不要错过这本书。书中有大量的实践案例、完备的细节讲解，将这本书常备于手边，比起查阅复杂繁琐的文档，能为工作或学习节省更多的时间。

王灿

浙江大学计算机学院副教教授

2014 年 11 月

前言 *Preface*

在一台服务器上同时运行一百个虚拟机，肯定会被认为是痴人说梦。而在一台服务器上同时运行一千个 Docker 容器，这已经成为现实。在计算机技术高速发展的今天，昔日的天方夜谭正在一个个变成现实。

多年的研发和运维（DevOps）经历中，笔者时常会碰到这样一个困境：用户的需求越来越多样，系统的规模越来越庞大，运行的软件越来越复杂，环境配置问题所造成的麻烦层出不穷……为了解决这些问题，开源社区推出过不少优秀的工具。这些方案虽然在某些程度上确能解决部分“燃眉之急”，但是始终没有一种方案能带来“一劳永逸”的效果。

让作为企业最核心资源的工程师们花费大量的时间，去解决各种环境和配置引发的 Bug，这真的正常吗？

回顾计算机的发展历程，最初，程序设计人员需要直接操作各种枯燥的机器指令，编程效率之低可想而知。高级语言的诞生，将机器指令的具体实现成功抽象出来，从此揭开了计算机编程效率突飞猛进的大时代。那么，为什么不能把类似的理念（抽象与分层）也引入到现代的研发和运维领域呢？

Docker 无疑在这一方向上迈出了具有革新意义的一步。笔者在刚接触 Docker 时，就为它所能带来的敏捷工作流程而深深吸引，也为它能充分挖掘云计算资源的效能而兴奋不已。我们深信，Docker 的出现，必将给 DevOps 技术，甚至整个信息技术产业的发展带来深远的影响。

笔者曾尝试编写了介绍 Docker 技术的中文开源文档。短短一个月的时间，竟收到了来自全球各个地区超过 20 万次的阅读量和全五星的好评。这让我们看到国内技术界对于新兴开源技术的敏锐嗅觉和迫切需求，同时也倍感压力，生怕其中有不妥之处，影响了大家学习和推广 Docker 技术的热情。在开源文档撰写过程中，我们一直在不断思考，在生产实践中到底怎么用 Docker 才是合理的？在“华章图书”的帮助下，终于有了现在读者手中的这本书。

与很多技术类书籍不同，本书中避免一上来就讲述冗长的故事，而是试图深入浅出、直奔主题，在最短时间内让读者理解和掌握最关键的技术点，并且配合实际操作案例和精炼的

点评，给读者提供真正可以上手的实战指南。

本书在结构上分为三大部分。第一部分是 Docker 技术的基础知识介绍，这部分将让读者对 Docker 技术能做什么有个全局的认识；第二部分将具体讲解各种典型场景的应用案例，供读者体会 Docker 在实际应用中的高效秘诀；第三部分将讨论一些偏技术环节的高级话题，试图让读者理解 Docker 在设计上的工程美学。最后的附录归纳了应用 Docker 的常见问题和一些常用的参考资料。读者可根据自身需求选择阅读重点。全书主要由杨保华和戴王剑主笔，曹亚仑写作了编程开发和实践之道章节。

本书在写作过程中参考了官方网站上的部分文档，并得到了 DockerPool 技术社区网友们的积极反馈和支持，在此一并感谢！

成稿之际，Docker 已经发布了增强安全特性的 1.32 版本。衷心祝愿 Docker 及相关技术能够快速成长和成熟，让众多 IT 从业人员的工作和生活都更加健康、更加美好！

作者于 2014 年 11 月

作者简介 *About the Authors*

杨保华 清华大学博士毕业，现为 IBM 中国研究院研究员。主要从事数据中心网络解决方案的研发与部署，技术方向包括云计算、软件定义网络（SDN）、网络安全等，是国内较早从事 SDN 和网络虚拟化相关技术的推广者，同时也是 DockerPool 开源社区的发起人之一。他的个人主页为 yeasy.github.io。

在本书的写作期间，得到了我的父母亲和妻子吴俞萱女士的关怀与支持，得到了公司领导和同事们的信任与鼓励，特别是刘天成帮忙审阅了部分内容。在此表示最深厚的感谢！

戴王剑 资深架构师，从事计算机网络、服务器架构设计多年，负责过多个省级项目的架构设计。热衷开源事业，是 DockerPool 开源社区的发起人之一。

写作期间，我的女儿戴子萱刚刚出生，感谢我的父母亲以及我的妻子刘乃华对我的大力支持，没有你们的辛勤付出，我不可能安心写完这本书，我爱你们！在这里一并感谢公司领导和同事对我的信任，感谢郭捷给予的硬件支持，在经过半年左右的测试之后，Docker 在公司的项目中正式上线，并给我们的开发、测试、生产带来了实实在在的效率。特别感谢我的导师：浙江师范大学杨传斌教授，没有杨老师，我可能在毕业前就放弃从事 IT 行业了。本书的出版能得到杨老师的肯定，是最让我开心的事。

曹亚仑 85 后，全栈 Web 开发者，擅长并专注于 SaaS 系统架构设计与研发，兴趣方向为 PaaS 和智能可穿戴设备。译著有《 Arduino 无线传感器网络实践指南》，开源图书有《程序员禅修指南》。个人主页为 allengaller.com。

我要感谢我的父母和妻子丁小芬，感谢你们在我写书过程中所给予的帮助和支持，我爱你们。

Contents 目录

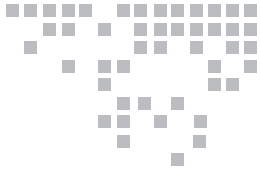
第一部分 Part 1

Docker 入门

欢迎来到 Docker 的世界！

在这一部分里，笔者将介绍 Docker 的基础知识，本部分分为 8 章。

第 1 章介绍 Docker 开源项目以及它与现有的虚拟化技术，特别是 Linux 容器技术的关系；第 2 章介绍 Docker 的三大核心概念，以及如何在常见的操作系统环境中安装 Docker；第 3 章第 5 章通过具体的示例操作讲解 Docker 的常用命令；第 6 章剖析如何在 Docker 中使用数据卷来保存数据；第 7 章介绍如何使用容器网络，特别是使容器访问外网和其他容器；第 8 章介绍如何编写 Dockerfile，以及使用 Dockerfile 配置文件来创建镜像的具体方法和注意事项。



Chapter 1

第 1 章 初识 Docker

如果说个人主机时代大家比拼的关键是 CPU 主频的高低和内存的大小，那么在云计算时代，虚拟化技术无疑是整座信息技术大厦最核心的一块基石。

伴随着信息技术产业的发展，虚拟化技术已经应用到各种关键场景中。从最早上世纪 60 年代 IBM 推出的大型主机虚拟化到后来 X86 平台上的虚拟化，虚拟化技术自身也在不断丰富和创新。

虚拟化既可以通过硬件模拟来实现，也可以通过操作系统来实现。而近些年出现的容器虚拟化方案，更是充分利用了操作系统本身已有的机制和特性，可以实现轻量级的虚拟化，甚至有人把它称为新一代的虚拟化技术。Docker 毫无疑问就是其中的佼佼者。

那么，什么是 Docker？它会带来什么好处？它跟现有虚拟化技术又有何关系呢？

本章在介绍 Docker 项目的起源和发展之后，会剖析 Docker 和 Linux 容器技术的密切联系，以及在开发和运维中使用 Docker 的突出优势。最后，还将阐述 Docker 在整个虚拟化领域中的定位。

1.1 什么是 Docker

Docker 开源项目

Docker 是基于 Go 语言实现的云开源项目，诞生于 2013 年初，最初发起者是 dotCloud 公司。Docker 自开源后受到广泛的关注和讨论，目前已有多个相关项目，逐渐形成了围绕 Docker 的生态体系。dotCloud 公司后来也改名为 Docker Inc，专注于 Docker 相关技术和产

品的开发。

Docker 项目目前已加入了 Linux 基金会，遵循 Apache 2.0 协议，全部开源代码均在 https://github.com/docker/docker 上进行维护。在最近一次 Linux 基金会的调查中，Docker 是仅次于 OpenStack 的最受欢迎的云计算开源项目。

现在主流的 Linux 操作系统都已经支持 Docker。例如，Redhat RHEL 6.5/ CentOS 6.5 往上的操作系统、Ubuntu 14.04 操作系统，都已经默认带有 Docker 软件包。Google 公司宣称在其 PaaS（Platform as a Service）平台及服务产品中广泛应用了 Docker。微软公司宣布和 Docker 公司合作，以加强其云平台 Azure 对 Docker 的支持。公有云提供商亚马逊近期也推出了 AWS EC2 Container，提供对 Docker 的支持。

Docker 的主要目标是“Build, Ship and Run Any App, Anywhere”，即通过对应用组件的封装（Packaging）、分发（Distribution）、部署（Deployment）、运行（Runtime）等生命周期的管理，达到应用组件级别的“一次封装，到处运行”。这里的应用组件，既可以是一个 Web 应用，也可以是一套数据库服务，甚至是一个操作系统或编译器。

Docker 基于 Linux 的多项开源技术提供了高效、敏捷和轻量级的容器方案，并且支持在多种主流云平台（PaaS）和本地系统上部署。可以说 Docker 为应用的开发和部署提供了“一站式”的解决方案。

Linux 容器技术

Docker 引擎的基础是 Linux 容器（Linux Containers，LXC）技术。IBM DeveloperWorks 上给出了关于容器技术的准确描述：

容器有效地将由单个操作系统管理的资源划分到孤立的组中，以便更好地在孤立的组之间平衡有冲突的资源使用需求。与虚拟化相比，这样既不需要指令级模拟，也不需要即时编译。容器可以在核心 CPU 本地运行指令，而不需要任何专门的解释机制。此外，也避免了准虚拟化（paravirtualization）和系统调用替换中的复杂性。

Linux 容器其实不是一个全新的概念。最早的容器技术可以追溯到 1982 年 Unix 系列操作系统上的 chroot 工具（直到今天，主流的 Unix、Linux 操作系统仍然支持和带有该工具）。早期的容器实现技术包括 Sun Solaris 操作系统上的 Solaris Containers（2004 年发布），FreeBSD 操作系统上的 FreeBSD jail（2000 年左右出现），以及 GNU/Linux 上的 Linux-VServer (http://linux-vserver.org/)（2001 年 10 月）和 OpenVZ (http://openvz.org)（2005 年）。

虽然这些技术经过多年的演化已经十分成熟，但是由于种种原因，这些容器技术并没有被集成到主流的 Linux 内核中，使用起来并不方便。例如，如果用户要使用 OpenVZ 技术，就需要先给操作系统打上特定的内核补丁方可使用。

后来 LXC 项目借鉴了前人成熟的容器设计理念，并基于一系列新的内核特性实现了更具扩展性的虚拟化容器方案。更加关键的是，LXC 被集成到了主流 Linux 内核中，进而成为 Linux 系统轻量级容器技术的事实标准。

从 Linux 容器到 Docker

在 LXC 的基础上，Docker 进一步优化了容器的使用体验。Docker 提供了各种容器管理工具（如分发、版本、移植等）让用户无需关注底层的操作，可以简单明了地管理和使用容器。用户操作 Docker 容器就像操作一个轻量级的虚拟机那样简单。

读者可以简单地将 Docker 容器理解为一种沙盒（Sandbox）。每个容器内运行一个应用，不同的容器相互隔离，容器之间也可以建立通信机制。容器的创建和停止都十分快速，容器自身对资源的需求也十分有限，远远低于虚拟机。很多时候，甚至直接把容器当作应用本身也没有任何问题。

有理由相信，随着 Docker 技术的进一步成熟，它将成为更受欢迎的容器虚拟化技术实现，得到更广泛的应用。

1.2　为什么要使用 Docker

Docker 容器虚拟化的好处

Docker 项目的发起人和 Docker Inc. 的 CTO Solomon Hykes 认为，Docker 在正确的地点、正确的时间顺应了正确的趋势——即高效地构建应用。现在开发者需要能方便地创建运行在云平台上的应用，也就是说应用必须能够脱离底层机器，而且同时必须是“任何时间任何地点”可获取的。因此，开发者们需要一种创建分布式应用程序的方式，这也是 Docker 所能够提供的。

举个简单的应用场景的例子。假设用户试图基于最常见的 LAMP（Linux + Apache + MySQL + PHP）组合来运维一个网站。按照传统的做法，首先，需要安装 Apache、MySQL 和 PHP 以及它们各自运行所依赖的环境；之后分别对它们进行配置（包括创建合适的用户、配置参数等）；经过大量的操作后，还需要进行功能测试，看是否工作正常；如果不正常，则意味着更多的时间代价和不可控的风险。可以想象，如果再加上更多的应用，事情会变得更加难以处理。

更为可怕的是，一旦需要服务器迁移（例如从阿里云迁移到腾讯云），往往需要重新部署和调试。这些琐碎而无趣的“体力活”，极大地降低了工作效率。

而 Docker 提供了一种更为聪明的方式，通过容器来打包应用，意味着迁移只需要在新的服务器上启动需要的容器就可以了。这无疑将节约大量的宝贵时间，并降低部署过程出现问题的风险。

Docker 在开发和运维中的优势

对开发和运维（DevOps）人员来说，可能最梦寐以求的就是一次性地创建或配置，可以

在任意环境、任意时间让应用正常地运行。而 Docker 恰恰是可以实现这一终极目标的瑞士军刀。

具体说来，Docker 在开发和运维过程中，具有如下几个方面的优势。

- 更快速的交付和部署。使用 Docker，开发人员可以使用镜像来快速构建一套标准的开发环境；开发完成之后，测试和运维人员可以直接使用相同环境来部署代码。Docker 可以快速创建和删除容器，实现快速迭代，大量节约开发、测试、部署的时间。并且，各个步骤都有明确的配置和操作，整个过程全程可见，使团队更容易理解应用的创建和工作过程。
- 更高效的资源利用。Docker 容器的运行不需要额外的虚拟化管理程序（Virtual Machine Manager，VMM，以及 Hypervisor）支持，它是内核级的虚拟化，可以实现更高的性能，同时对资源的额外需求很低。
- 更轻松的迁移和扩展。Docker 容器几乎可以在任意的平台上运行，包括物理机、虚拟机、公有云、私有云、个人电脑、服务器等。 这种兼容性让用户可以在不同平台之间轻松地迁移应用。
- 更简单的更新管理。使用 Dockerfile，只需要小小的配置修改，就可以替代以往大量的更新工作。并且所有修改都以增量的方式进行分发和更新，从而实现自动化并且高效的容器管理。

Docker 与虚拟机比较

作为一种轻量级的虚拟化方式，Docker 在运行应用上跟传统的虚拟机方式相比具有显著优势：

- Docker 容器很快，启动和停止可以在秒级实现，这相比传统的虚拟机方式要快得多。
- Docker 容器对系统资源需求很少，一台主机上可以同时运行数千个 Docker 容器。
- Docker 通过类似 Git 的操作来方便用户获取、分发和更新应用镜像，指令简明，学习成本较低。
- Docker 通过 Dockerfile 配置文件来支持灵活的自动化创建和部署机制，提高工作效率。

Docker 容器除了运行其中的应用之外，基本不消耗额外的系统资源，保证应用性能的同时，尽量减小系统开销。传统虚拟机方式运行 N 个不同的应用就要启动 N 个虚拟机（每个虚拟机需要单独分配独占的内存、磁盘等资源），而 Docker 只需要启动 N 个隔离的容器，并将应用放到容器内即可。

当然，在隔离性方面，传统的虚拟机方式多了一层额外的隔离。但这并不意味着 Docker 就不安全。Docker 利用 Linux 系统上的多种防护机制实现了严格可靠的隔离。从 1.3 版本开始，Docker 引入了安全选项和镜像签名机制，极大地提高了使用 Docker 的安全性。

下表总结了使用 Docker 容器技术与传统虚拟机技术的特性比较。

特性	容器	虚拟机
启动速度	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个
隔离性	安全隔离	完全隔离

1.3 虚拟化与 Docker

虚拟化技术是一个通用的概念，在不同领域有不同的理解。在计算领域，一般指的是计算虚拟化（Computing Virtualization），或通常说的服务器虚拟化。维基百科上的定义如下：

在计算机技术中，虚拟化（Virtualization）是一种资源管理技术，是将计算机的各种实体资源，如服务器、网络、内存及存储等，予以抽象、转换后呈现出来，打破实体结构间的不可切割的障碍，使用户可以用比原本的组态更好的方式来应用这些资源。

可见，虚拟化的核心是对资源进行抽象，目标往往是为了在同一个主机上运行多个系统或应用，从而提高系统资源的利用率，同时带来降低成本、方便管理和容错容灾等好处。

从大类上分，虚拟化技术可分为基于硬件的虚拟化和基于软件的虚拟化。其中，真正意义上的基于硬件的虚拟化技术不多见，少数如网卡中的单根多 IO 虚拟化（Single Root I/O Virtualization and Sharing Specification，SR-IOV）等技术，也超出了本书的讨论范畴。

基于软件的虚拟化从对象所在的层次，又可以分为应用虚拟化和平台虚拟化（通常说的虚拟机技术即属于这个范畴）。其中，前者一般指的是一些模拟设备或 Wine 这样的软件。后者又可以细分为如下几个子类：

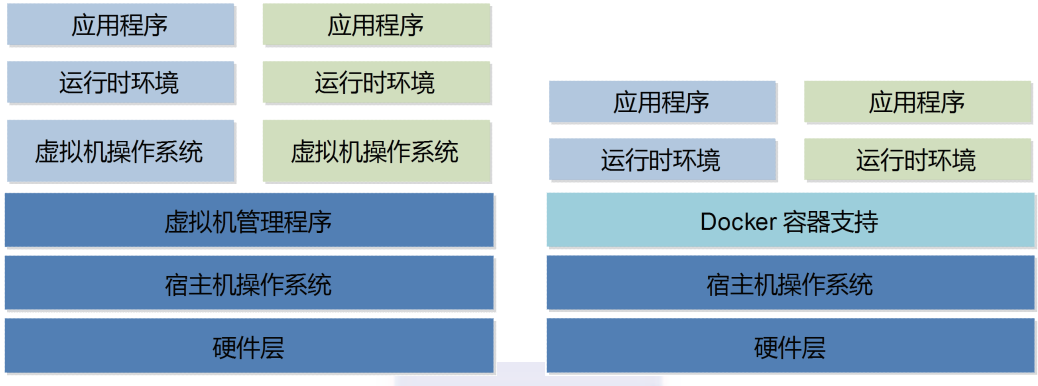
- 完全虚拟化。虚拟机模拟完整的底层硬件环境和特权指令的执行过程，客户操作系统无需进行修改。例如 VMware Workstation、VirtualBox、QEMU 等。
- 硬件辅助虚拟化。利用硬件（主要是 CPU）辅助支持（目前 x86 体系结构上可用的硬件辅助虚拟化技术包括 Intel-VT 和 AMD-V）处理敏感指令来实现完全虚拟化的功能，客户操作系统无需修改，例如 VMware Workstation、Xen、KVM。
- 部分虚拟化。只针对部分硬件资源进行虚拟化，客户操作系统需要进行修改。现在有些虚拟化技术的早期版本仅支持部分虚拟化。
- 超虚拟化（Paravirtualization）。部分硬件接口以软件的形式提供给客户机操作系统，客户操作系统需要进行修改，例如早期的 Xen。
- 操作系统级虚拟化。内核通过创建多个虚拟的操作系统实例（内核和库）来隔离不同的进程。容器相关技术即在这个范畴。

可见，Docker 以及其他容器技术都属于操作系统的虚拟化这个范畴。

Docker 虚拟化方式之所以拥有众多优势，这跟操作系统的虚拟化自身的特点是分不开

的。下面图 1-1 比较了 Docker 和常见的虚拟机方式的不同之处。

传统方式是在硬件层面实现虚拟化，需要有额外的虚拟机管理应用和虚拟机操作系统层。



a) 传统的虚拟化方式　　b) Docker 虚拟化方式

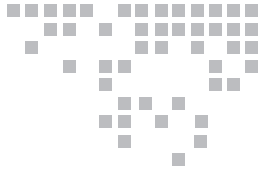
图 1-1　Docker 和传统的虚拟机方式的不同之处

Docker 容器是在操作系统层面上实现虚拟化，直接复用本地主机的操作系统，因此更加轻量级。

1.4　本章小结

通过本章内容的叙述，相信读者已经对于 Docker 技术不再陌生，并为它带来的众多优势所深深吸引。通过为 Linux 容器技术提供更简便的使用和管理方案、更高效的版本控制机制，Docker 让容器技术一下子变得前所未有的方便易用。

笔者相信，随着云计算技术的进一步发展，以 Docker 技术为代表的容器技术必将在整个虚拟化领域占有越来越重要的地位。



Chapter 2

第 2 章 Docker 的核心概念和安装

本章首先介绍 Docker 的三大核心概念：

- ❑ 镜像（Image）
- ❑ 容器（Container）
- ❑ 仓库（Repository）

读者理解了这三个核心概念，就能顺利地理解 Docker 的整个生命周期。社区讨论很激烈的一个话题，就是 Docker 和 Linux 容器技术到底有何区别？相信读者在阅读完本章后，会得到更清晰的答案。

随后，笔者将介绍如何在常见的操作系统上安装 Docker，包括 Ubuntu、CentOS、Windows 和 MacOS 等。

2.1 核心概念

Docker 镜像

Docker 镜像（Image）类似于虚拟机镜像，可以将它理解为一个面向 Docker 引擎的只读模板，包含了文件系统。

例如：一个镜像可以只包含一个完整的 Ubuntu 操作系统环境，可以把它称为一个 Ubuntu 镜像。镜像也可以安装了 Apache 应用程序（或用户需要的其他软件），可以把它称为一个 Apache 镜像。

镜像是创建 Docker 容器的基础。通过版本管理和增量的文件系统，Docker 提供了一套十分简单的机制来创建和更新现有的镜像，用户甚至可以从网上下载一个已经做好的应用镜

像，并通过简单的命令就可以直接使用。

Docker 容器

Docker 容器（Container）类似于一个轻量级的沙箱，Docker 利用容器来运行和隔离应用。

容器是从镜像创建的应用运行实例，可以将其启动、开始、停止、删除，而这些容器都是相互隔离、互不可见的。

读者可以把容器看做一个简易版的 Linux 系统环境（这包括 root 用户权限、进程空间、用户空间和网络空间等），以及运行在其中的应用程序打包而成的应用盒子。

镜像自身是只读的。容器从镜像启动的时候，Docker 会在镜像的最上层创建一个可写层，镜像本身将保持不变。

Docker 仓库

Docker 仓库（Repository）类似于代码仓库，是 Docker 集中存放镜像文件的场所。

有时候会看到有资料将 Docker 仓库和注册服务器（Registry）混为一谈，并不严格区分。实际上，注册服务器是存放仓库的地方，其上往往存放着多个仓库。每个仓库集中存放某一类镜像，往往包括多个镜像文件，通过不同的标签（tag）来进行区分。例如存放 Ubuntu 操作系统镜像的仓库，称为 Ubuntu 仓库，其中可能包括 14.04、12.04 等不同版本的镜像。仓库注册服务器的示例如图 2-1 所示。

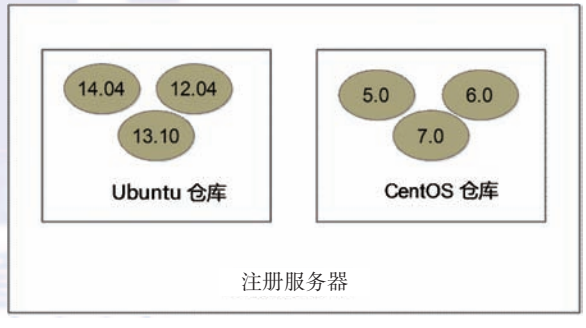


图 2-1 仓库和注册服务器

根据所存储的镜像公开分享与否，Docker 仓库可以分为公开仓库（Public）和私有仓库（Private）两种形式。

目前，最大的公开仓库是 Docker Hub，存放了数量庞大的镜像供用户下载。国内的公开仓库包括 Docker Pool 等，可以提供稳定的国内访问。

当然，用户如果不希望公开分享自己的镜像文件，Docker 也支持用户在本地网络内创建一个只能自己访问的私有仓库。

当用户创建了自己的镜像之后就可以使用 push 命令将它上传到指定的公有或者私有仓库。这样用户下次在另外一台机器上使用该镜像时，只需将其从仓库上 pull 下来就可以了。

可以看出，Docker 利用仓库管理镜像的设计理念与 Git 非常相似。

2.2　安装 Docker

Docker 支持在主流的操作系统平台上使用，包括 Ubuntu、CentOS、Windows 以及 MacOS 系统等。当然，在 Linux 系列平台上是原生支持，使用体验也最好。

Ubuntu

1. Ubuntu 14.04 及以上版本

Ubuntu 14.04 版本官方软件源中已经自带了 Docker 包，可以直接安装：

```
$ sudo apt-get update
$ sudo apt-get install -y docker.io
$ sudo ln -sf /usr/bin/docker.io /usr/local/bin/docker
$ sudo sed -i '$acomplete -F _docker docker' /etc/bash_completion.d/docker.io
```

以上流程使用 Ubuntu 14.04 系统默认自带 docker.io 安装包安装 Docker，这样安装的 Docker 版本相对较旧。

读者也可通过下面的方法从 Docker 官方源安装最新版本。首先需要安装 apt-transport-https，并添加 Docker 官方源：

```
$ sudo apt-get install apt-transport-https
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 36A1D7
869245C8950F966E92D8576A8BA88D21E9
$ sudo bash -c "echo deb https://get.docker.io/ubuntu docker main > /etc/apt/
sources.list.d/docker.list"
$ sudo apt-get update
```

之后，可以通过下面的命令来安装最新版本的 Docker：

```
$ sudo apt-get install -y lxc-docker
```

在安装了 Docker 官方软件源后，若需要更新 Docker 软件版本，只需要执行以下命令即可升级：

```
$ sudo apt-get update -y lxc-docker
```

后文中使用 $ 作为终端引导符时，表示非 root 权限用户；# 代表是 root 用户。

2. ubuntu 14.04 以下的版本

如果使用的是较低版本的 Ubuntu 系统，则需要先进行内核更新并重启系统后再进行安装：

```
$ sudo apt-get update
```

```
$ sudo apt-get install -y linux-image-generic-lts-raring linux-headers-generic-
lts-raring
$ sudo reboot
```

重启后，重复在 Ubuntu 14.04 系统的安装步骤即可。

CentOS

Docker 支持 CentOS 6 及以后的版本。

对于 CentOS 6 系统可使用 EPEL 库安装 Docker，命令如下：

```
$ sudo yum install -y http://mirrors.yun-idc.com/epel/6/i386/epel-release-6-8.
noarch.rpm
$ sudo yum install -y docker-io
```

对于 CentOS 7 系统，由于 CentOS-Extras 源中已内置 Docker，读者可以直接使用 yum 命令进行安装：

```
$ sudo yum install -y docker
```

目前在 Centos 系统中更新 Docker 软件有两种方法，一是自行通过源码编译安装，二是下载二进制文件进行更新。

Windows

目前 Docker 官方已经宣布 Docker 通过虚拟机方式支持 Windows 7.1 和 8，前提是主机的 CPU 支持硬件虚拟化。由于近几年发布的 Intel 和 AMD CPU 基本上都已支持了硬件虚拟化特性，因此在 Windows 中使用 Docker 通常不会有硬件支持的问题。

由于 Docker 引擎使用了 Linux 内核特性，所以在 Windows 上运行的话，需要额外使用一个虚拟机来提供 Linux 支持。这里推荐使用 Boot2Docker 工具，它会首先安装一个经过加工与配置的 VirtualBox 轻量级虚拟机，然后在其中运行 Docker。主要步骤如下：

1）从 https://docs.docker.com/installation/windows/ 下载最新官方 Docker for Windows Installer。

2）运行 Installer。这个过程将安装 VirtualBox，MSYS-git，boot2docker Linux ISO 镜像，以及 Boot2Docker 管理工具。如图 2-2 所示。

图 2-2 Windows 下安装 Docker

3）打开桌面的 Boot2Docker Start 程序，或者用以下命令：Program Files > Boot2Docker for Windows。此初始化脚本在第一次运行时需要输入一个 SSH Key Passphrase（用于 SSH 密钥生成的口令）。读者可以自行设定，也可以直接按回车键，跳过此设定，如图 2-3 所示。

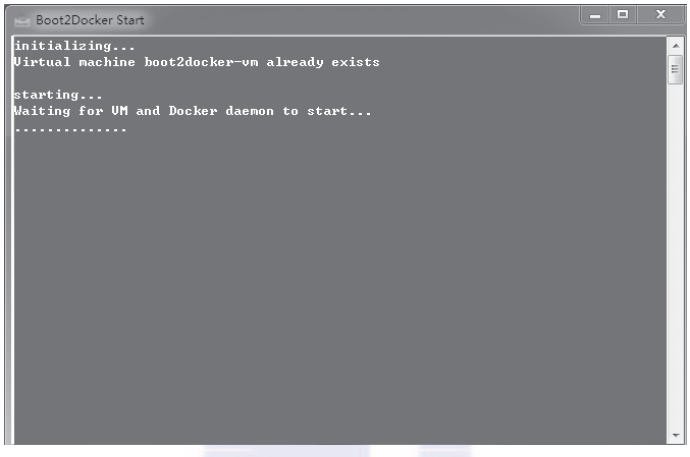


图 2-3　Boot2Docker 启动后界面

此时 Boot2Docker Start 程序将连接至虚拟机中的 Shell 会话，Docker 已经运行起来了！

Mac OS

目前 Docker 已经支持 Mac OS X 10.6 Snow Leopard 及以上版本的 Mac OS。

在 Mac OS 上使用 Docker，同样需要 Boot2Docker 工具的支持。主要步骤如下：

1）下载最新官方 Docker for OS X Installer。读者可以从 https://docs.docker.com/installation/mac/ 下载。

2）双击运行安装包。这个过程将安装一个 VirtualBox 虚拟机、Docker 本身以及 Boot2Docker 管理工具，如图 2-4 所示。

3）安装成功后，找到 Boot2Docker（Mac 系统的 Application 或“应用”文件夹中）并运行它。现在进行 Boot2Docker 的初始化：

```
$ boot2docker init
$ boot2docker start
$ $(boot2docker shellinit)
```

读者将看到虚拟机在命令行窗口中启动运行，并显示 Docker 的启动信息，则说明 Docker 安装成功。当虚拟机初始化完毕后，可以使用 `boot2docker stop` 和 `boot2docker start` 来控制它。

注意：如果在命令行中看到如下提示信息：

```
To connect the Docker client to the Docker daemon, please set: export  DOCKER_
HOST=tcp://192.168.59.103:2375
```

可以执行提示信息中的语句：`export DOCKER_HOST=tcp://192.168.59.103:2375`。此语句的作用是在系统环境变量中设置 Docker 的主机地址。

图 2-4 Mac OS 上安装 Boot2Docker

2.3 本书环境介绍

本书的实践环境是一台装有 Linux Mint 17 的笔记本电脑，并使用虚拟机软件 VirturBox 虚拟了一套 Ubuntu 14.04 系统，两套系统上都安装了 Docker 的 1.3 版本，虚拟机通过 VirturBox 网络的 NAT 方式连接到外部，如图 2-5 所示。

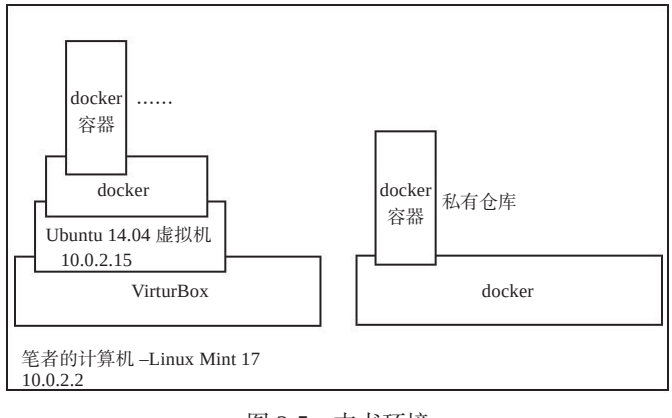


图 2-5 本书环境

其中，Ubuntu 14.04 虚拟机将是主要的操作环境（自动获取的 IP 地址为 10.0.2.15/24），而笔记本上装的 Linux Mint 环境（内网地址为 10.0.2.2/24，外网地址为 192.168.1.0/24 段地址）将作

为本地私有仓库的服务器，演示跟仓库相关的操作。

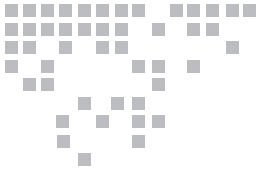
读者可根据自己本地环境，选择搭建类似的环境。

2.4　本章小结

本章介绍了 Docker 的三大核心概念：镜像、容器和仓库。

通过这三大核心概念所构建的高效工作流程，毫无疑问，正是 Docker 得以从众多容器虚拟化方案中脱颖而出的重要原因。

熟悉 Git 和 GitHub 的读者，会理解这一工作流程为文件分发和合作所带来的众多优势。在后续章节，笔者将进一步地介绍围绕这三大核心概念的 Docker 常见操作命令。



Chapter 3

第 3 章 镜　像

镜像是 Docker 的三大核心概念之一。

Docker 运行容器前需要本地存在对应的镜像，如果镜像不存在本地，Docker 会尝试先从默认镜像仓库下载（默认使用 Docker Hub 公共注册服务器中的仓库），用户也可以通过配置，使用自定义的镜像仓库。

本章将介绍围绕镜像这一核心概念的具体操作，包括如何使用 pull 命令从 Docker Hub 仓库中下载镜像到本地；如何查看本地已有的镜像信息；如何在远端仓库使用 search 命令进行搜索和过滤；如何删除镜像标签和镜像文件；如何创建用户定制的镜像并且保存为外部文件。最后，还将介绍如何向 Docker Hub 仓库中推送自己的镜像。

3.1　获取镜像

镜像是 Docker 运行容器的前提。

读者可以使用 `docker pull` 命令从网络上下载镜像。该命令的格式为 `docker pull NAME[:TAG]`。对于 Docker 镜像来说，如果不显式地指定 `TAG`，则默认会选择 `latest` 标签，即下载仓库中最新版本的镜像。

下面，笔者将演示如何从 Docker Hub 的 Ubuntu 仓库下载一个最新的 Ubuntu 操作系统的镜像。

```
$ sudo docker pull ubuntu
ubuntu:latest: The image you are pulling has been verified

d497ad3926c8: Downloading [======>                    ] 25.41 MB/201.6 MB 51m14s
```

```
ccb62158e970: Download complete
e791be0477f2: Download complete
3680052c0f5c: Download complete
22093c35d77b: Download complete
5506de2b643b: Download complete
511136ea3c5a: Download complete
```

该命令实际上下载的就是 ubuntu:latest 镜像，目前最新的 14.04 版本的镜像。

下载过程中可以看出，镜像文件一般由若干层组成，行首的 2185fd50e2ca 这样的字串代表了各层的 ID。下载过程中会获取并输出镜像的各层信息。层（Layer）其实是 AUFS（Advanced Union File System，一种联合文件系统）中的重要概念，是实现增量保存与更新的基础。

读者还可以通过指定标签来下载特定版本的某一个镜像，例如 14.04 标签的镜像。

```
$ sudo docker pull ubuntu:14.04
```

上面两条命令实际上都相当于 $ sudo docker pull registry.hub.docker.com/ubuntu:latest 命令，即从默认的注册服务器 registry.hub.docker.com 中的 ubuntu 仓库来下载标记为 latest 的镜像。

用户也可以选择从其他注册服务器的仓库下载。此时，需要在仓库名称前指定完整的仓库注册服务器地址。例如从 DockerPool 社区的镜像源 dl.dockerpool.com 下载最新的 Ubuntu 镜像。

```
$ sudo docker pull dl.dockerpool.com:5000/ubuntu
```

下载镜像到本地后，即可随时使用该镜像了，例如利用该镜像创建一个容器，在其中运行 bash 应用。

```
$ sudo docker run -t -i ubuntu /bin/bash
root@fe7fc4bd8fc9:/#
```

3.2 查看镜像信息

使用 docker images 命令可以列出本地主机上已有的镜像。

例如，下面的命令列出了本地刚从官方下载的 ubuntu:14.04 镜像，以及从 DockerPool 镜像源下载的 ubuntu:latest 镜像。

```
$ sudo docker images
REPOSITORY           TAG        IMAGE ID     CREATED       VIRTUAL SIZE
ubuntu             14.04         5506de2b643b      1 weeks ago        197.8 MB
dl.dockerpool.com:5000/ubuntu  latest   5506de2b643b   1 weeks ago    197.8 MB
```

在列出信息中，可以看到几个字段信息：

- 来自于哪个仓库，比如 ubuntu 仓库。
- 镜像的标签信息，比如 14.04。
- 镜像的 ID 号（唯一）。
- 创建时间。
- 镜像大小。

其中镜像的 ID 信息十分重要，它唯一标识了镜像。

TAG 信息用于标记来自同一个仓库的不同镜像。例如 ubuntu 仓库中有多个镜像，通过 TAG 信息来区分发行版本，包括 10.04、12.04、12.10、13.04、14.04 等标签。

为了方便在后续工作中使用这个镜像，还可以使用 docker tag 命令为本地镜像添加新的标签。例如添加一个新的 ubuntu:latest 镜像标签如下：

```
$ sudo docker tag dl.dockerpool.com:5000/ubuntu:latest ubuntu:latest
```

再次使用 docker images 列出本地主机上镜像信息，可以看到多了一个 ubuntu:latest 标签的镜像。

```
$ sudo docker images
REPOSITORY           TAG         IMAGE ID       CREATED          VIRTUAL SIZE
ubuntu               14.04        5506de2b643b     1 weeks ago       197.8 MB
dl.dockerpool.com:5000/ubuntu latest   5506de2b643b    1 weeks ago    192.8 MB
ubuntu               latest         5506de2b643b      1 weeks ago       192.8 MB
```

细心的读者可能会注意到，这些不同标签的镜像的 ID 是完全一致的，说明它们实际上指向了同一个镜像文件，只是别名不同而已。标签在这里起到了引用或快捷方式的作用。

使用 docker inspect 命令可以获取该镜像的详细信息。

```
$ sudo docker inspect 5506de2b643b
[{
    "Architecture": "amd64",
    "Author": "",
    "Comment": "",
    "Config": {
        "AttachStderr": false,
        "AttachStdin": false,
        "AttachStdout": false,
        "Cmd": [
            "/bin/bash"
        ],
        "CpuShares": 0,
        "Cpuset": "",
        "Domainname": "",
        "Entrypoint": null,
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
        ],
        "ExposedPorts": null,
```

```
    "Hostname": "065262ce3c91",
    "Image":"964692831e07f7362f5c3fedf0c4b81a622f2c6e3ec5f19d0eddff21afd64c12",
    "Memory": 0,
    "MemorySwap": 0,
    "NetworkDisabled": false,
    "OnBuild": [],
    "OpenStdin": false,
    "PortSpecs": null,
    "StdinOnce": false,
    "Tty": false,
    "User": "",
    "Volumes": null,
    "WorkingDir": ""
},
"Container":"f26bc14cc07412402bdab911b8a935fead0322649cf042cee8515c02ebdfa53a",
"ContainerConfig": {
    "AttachStderr": false,
    "AttachStdin": false,
    "AttachStdout": false,
    "Cmd": [
        "/bin/sh",
        "-c",
        "#(nop) CMD [/bin/bash]"
    ],
    "CpuShares": 0,
    "Cpuset": "",
    "Domainname": "",
    "Entrypoint": null,
    "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ],
    "ExposedPorts": null,
    "Hostname": "065262ce3c91",
    "Image":"964692831e07f7362f5c3fedf0c4b81a622f2c6e3ec5f19d0eddff21afd64c12",
    "Memory": 0,
    "MemorySwap": 0,
    "NetworkDisabled": false,
    "OnBuild": [],
    "OpenStdin": false,
    "PortSpecs": null,
    "StdinOnce": false,
    "Tty": false,
    "User": "",
    "Volumes": null,
    "WorkingDir": ""
},
"Created": "2014-09-23T22:37:05.812213629Z",
"DockerVersion": "1.2.0",
"Id": "53bf7a53e8903fce40d24663901aac6211373a8d8b4effe08bc884e63e181805",
"Os": "linux",
"Parent":"964692831e07f7362f5c3fedf0c4b81a622f2c6e3ec5f19d0eddff21afd64c12",
```

```
    "Size": 0
}
]
```

`docker inspect` 命令返回的是一个 JSON 格式的消息，如果我们只要其中一项内容时，可以使用 `-f` 参数来指定，例如，获取镜像的 `Architecture` 信息：

```
$ sudo docker inspect -f {{".Architecture"}} 550
amd64
```

在指定镜像 ID 的时候，通常使用该 ID 的前若干个字符组成的可区分字串来替代完整的 ID。

3.3 搜寻镜像

使用 `docker search` 命令可以搜索远端仓库中共享的镜像，默认搜索 Docker Hub 官方仓库中的镜像。用法为 `docker search TERM`，支持的参数包括：

- `--automated=false` 仅显示自动创建的镜像。
- `--no-trunc=false` 输出信息不截断显示。
- `-s, --stars=0` 指定仅显示评价为指定星级以上的镜像。

例如，搜索带 mysql 关键字的镜像如下所示：

```
$ sudo docker search mysql
NAME             DESCRIPTION          STARS     OFFICIAL   AUTOMATED
NAME             DESCRIPTION          STARS     OFFICIAL   AUTOMATED
mysql            MySQL is a widely used, open-source relati...   213       [OK]
tutum/mysql      MySQL Server image - listens in port 3306....   74        [OK]
orchardup/mysql                                                  36        [OK]
tutum/lamp       LAMP image - Apache listens in port 80, an...   32        [OK]
tutum/wordpress  Wordpress Docker image - listens in port 8...   26        [OK]
paintedfox/mariadb A docker image for running MariaDB 5.5, a ... 21        [OK]
dockerfile/mysql  Trusted automated MySQL (http://dev.mysql....   14        [OK]
google/mysql     MySQL server for Google Compute Engine          13        [OK]
anapsix/gitlab-ci  GitLab-CI Continuous Integration in Docker... 12        [OK]
centurylink/drupal Drupal docker image without a DB included ... 11        [OK]
stenote/docker-lemp  MySQL 5.6、PHP 5.5、Nginx、Memcache          10        [OK]
...
```

可以看到返回了很多包含关键字的镜像，其中包括镜像名字、描述、星级（表示该镜像的受欢迎程度）、是否官方创建、是否自动创建等。

默认的输出结果将按照星级评价进行排序。官方的镜像说明是官方项目组创建和维护的，`automated` 资源则允许用户验证镜像的来源和内容。

3.4 删除镜像

使用镜像的标签删除镜像

使用docker rmi命令可以删除镜像，命令格式为docker rmi IMAGE [IMAGE...]，其中IMAGE可以为标签或ID。

例如，要删除掉dl.dockerpool.com:5000/ubuntu:latest镜像，可以使用如下命令：

```
$ sudo docker rmi dl.dockerpool.com:5000/ubuntu
Untagged: dl.dockerpool.com:5000/ubuntu:latest
```

读者可能会担心，本地的ubuntu:latest镜像是否会受到此命令的影响。无需担心，当同一个镜像拥有多个标签的时候，docker rmi命令只是删除了该镜像多个标签中的指定标签而已，并不影响镜像文件。因此上述操作相当于只是删除了镜像5506de2b643b的一个标签而已。

为保险起见，再次查看本地的镜像，发现ubuntu:latest镜像（准确地说，是5506de2b643b镜像）仍然存在：

```
$ sudo docker images
REPOSITORY          TAG          IMAGE ID          CREATED          VIRTUAL SIZE
ubuntu              14.04        5506de2b643b      1 weeks ago      197.8 MB
ubuntu              latest       5506de2b643b      1 weeks ago      192.8 MB
```

但当镜像只剩下一个标签的时候就要小心了，此时再使用docker rmi命令会彻底删除该镜像。

假设本地存在一个标签为mysql:latest的镜像，且没有额外的标签指向它，执行docker rmi命令，可以看出它会删除这个镜像文件的所有AUFS层：

```
$ sudo docker rmi mysql:latest
Untagged: mysql:latest
Deleted: 9a09222edf600a03ea48bd23cfa363841e45a8715237e3a58cb0167f0e8bad54
Deleted: 4daeda4ad839a152a3b649672bd5135977d7f81866d3bc0e16d0af3f65cc8af6
Deleted: cf07a411bf0883bd632940e8108dac49c64456a47f7390507de5685bbd6daf85
Deleted: 4f513746df18b222a07bb8d76d4b6d29752ce5dcb69bfad0ce92e6c1449a3821
Deleted: 228ecd435c8a29d25b77999036701a27f2d67874c915bb8eb9fb175b1f98aa60
Deleted: 37e4b3932afa186924a09eab332bc8ebec3aac8bac074314ed9a2d1e94547f50
Deleted: 898883ccfcee705e440547e30e240cb025c12410d7c9e4d2bcb11973ba075975
Deleted: 0a09ddcf99b7fd8fcb3525c41b54696038ecf13677f4459f1c98c742ffa60ab2
Deleted: 35bc8591e39be5089265a093e234d13a4b155a01d2ab9e8904eafa81664fb597
Deleted: 857e856e4481d59ee88a4cdedd9aaf855666bd494fa38506e6788361c0af4cda
```

使用镜像ID删除镜像

当使用docker rmi命令后面跟上镜像的ID（也可以是ID能进行区分的部分前缀串）

时，会先尝试删除所有指向该镜像的标签，然后删除该镜像文件本身。

注意，当有该镜像创建的容器存在时，镜像文件默认是无法被删除的，例如：

先利用 ubuntu 镜像创建一个简单的容器，输出一句话“hello! I am here!”：

```
$ sudo docker run ubuntu echo 'hello! I am here!'
hello! I am here!
```

使用 docker ps -a 命令可以看到本机上存在的所有容器：

```
$ sudo docker ps -a
CONTAINER ID     IMAGE     COMMAND     CREATED     STATUS      PORTS      NAMES
e812617b41f6      ubuntu:latest      "echo 'hello! I am h      13 seconds ago
Exited (0) 12 seconds ago     silly_leakey
```

可以看到，后台存在一个退出状态的容器，是刚基于 ubuntu:latest 镜像创建的。

试图删除该镜像，Docker 会提示有容器正在运行，无法删除：

```
$ sudo docker rmi ubuntu
Error response from daemon: Conflict, cannot delete 5506de2b643b because the
container e812617b41f6 is using it, use -f to force
2014/10/16 18:10:31 Error: failed to remove one or more images
```

如果要想强行删除镜像，可以使用 -f 参数：

```
$ sudo docker rmi -f ubuntu
```

笔者不推荐使用 -f 参数来强制删除一个存在容器依赖的镜像，因为这样往往会造成一些遗留问题。

再次使用 docker images 查看本地的镜像列表，读者会发现一个标签为 <none> 的临时镜像，原来被强制删除的镜像换了新的 ID 继续存在系统中。

```
$ sudo docker images
REPOSITORY       TAG          IMAGE ID          CREATED           VIRTUAL SIZE
<none>           <none>       2318d26665ef      3 months ago      198.7 MB
```

因此，正确的做法是，先删除依赖该镜像的所有容器，再来删除镜像。首先删除容器 e812617b41f6：

```
$ sudo docker rm e81
e81
```

此时再使用临时的 ID 来删除镜像，此时会正常打印出删除的各层信息：

```
core@localhost ~ $   docker rmi -f 2318d26665ef
Deleted: 2318d26665eff33e9f91c4c99036751afb40eb58f944a585372bec1407828ad3
Deleted: ebc34468f71dca9cb9937bf4c33062540bcacae148df8a70053bfd1acbecaa20
Deleted: 25f11f5fb0cb9e41531d1da8dc56351286427e070c536f7015fe76e4dae0a4bc
Deleted: 9bad880da3d219b10423804147d6982da1a7bb1e285777a4d746afca6215bebb
Deleted: 511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158
```

此时查看本地镜像，读者会发现临时镜像已经被删除：

```
$ sudo docker images
REPOSITORY       TAG       IMAGE ID       CREATED       VIRTUAL SIZE
```

3.5 创建镜像

创建镜像的方法有三种：基于已有镜像的容器创建、基于本地模板导入、基于 Dockerfile 创建。

本节将重点介绍前两种方法。最后一种基于 Dockerfile 创建的方法将在后续章节专门予以详细介绍。

基于已有镜像的容器创建

该方法主要是使用 `docker commit` 命令，其命令格式为 `docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]`，主要选项包括：

- `-a, --author=""` 作者信息。
- `-m, --message=""` 提交消息。
- `-p, --pause=true` 提交时暂停容器运行。

下面将演示如何使用该命令创建一个新镜像。首先，启动一个镜像，并在其中进行修改操作，例如创建一个 test 文件，之后退出：

```
$ sudo docker run -ti ubuntu:14.04 /bin/bash
root@a925cb40b3f0:/# touch test
root@a925cb40b3f0:/# exit
```

记住容器的 ID 为 `a925cb40b3f0`。

此时该容器跟原 `ubuntu:14.04` 镜像相比，已经发生了改变，可以使用 `docker commit` 命令来提交为一个新的镜像。提交时可以使用 ID 或名称来指定容器：

```
$ sudo docker commit -m "Added a new file" -a "Docker Newbee" a925cb40b3f0 test
9e9c814023bcffc3e67e892a235afe61b02f66a947d2747f724bd317dda02f27
```

顺利的话，命令会返回新创建的镜像的 ID 信息，例如：

```
9e9c814023bcffc3e67e892a235afe61b02f66a947d2747f724bd317dda02f27
```

此时查看本地镜像列表，即可看到新创建的镜像：

```
$ sudo docker images
REPOSITORY      TAG         IMAGE ID          CREATED            VIRTUAL SIZE
test            latest      9e9c814023bc      4 seconds ago      225.4 MB
```

基于本地模板导入

也可以直接从一个操作系统模板文件导入一个镜像。在这里，推荐使用 OpenVZ 提供的模板来创建。OPENVZ 模板的下载地址为 http://openvz.org/Download/templates/precreated。

比如，笔者下载了一个 ubuntu-14.04 的模板压缩包后，可以使用以下命令导入：

```
$ sudo cat ubuntu-14.04-x86_64-minimal.tar.gz  |docker import - ubuntu:14.04
```

然后查看新导入的镜像，已经在本地存在了：

```
$ sudo docker images
REPOSITORY      TAG      IMAGE ID          CREATED             VIRTUAL SIZE
ubuntu          14.04    05ac7c0b9383      17 seconds ago      215.5 MB
```

3.6 存出和载入镜像

可以使用 `docker save` 和 `docker load` 命令来存出和载入镜像。

存出镜像

如果要存出镜像到本地文件，可以使用 `docker save` 命令。例如，存出本地的 ubuntu:14.04 镜像为文件 ubuntu_14.04.tar：

```
$ sudo docker images
REPOSITORY      TAG        IMAGE ID          CREATED          VIRTUAL SIZE
ubuntu          14.04      c4ff7513909d      5 weeks ago      225.4 MB
...
$ sudo docker save -o ubuntu_14.04.tar ubuntu:14.04
```

载入镜像

可以使用 `docker load` 从存出的本地文件中再导入到本地镜像库，例如从文件 ubuntu_14.04.tar 导入镜像到本地镜像列表，如下所示：

```
$ sudo docker load --input ubuntu_14.04.tar
```

或

```
$ sudo docker load < ubuntu_14.04.tar
```

这将导入镜像以及其相关的元数据信息（包括标签等），可以使用 `docker images` 命令进行查看。

3.7 上传镜像

可以使用 `docker push` 命令上传镜像到仓库，默认上传到 DockerHub 官方仓库（需要登录），命令格式为 `docker push NAME[:TAG]`。

用户在 DockerHub 网站注册后，即可上传自制的镜像。例如用户 `user` 上传本地的 `test:latest` 镜像，可以先添加新的标签 `user/test:latest`，然后用 `docker push` 命令上传镜像：

```
$ sudo docker tag test:latest user/test:latest
$ sudo docker push user/test:latest
The push refers to a repository [base/163] (len: 1)
Sending image list

Please login prior to push:
Username:
Password:
Email: xxx@xxx.com
```

第一次使用时，会提示输入登录信息或进行注册。

3.8 本章小结

本章具体介绍了围绕 Docker 镜像的一系列重要命令操作，包括获取、查看、搜索、删除、创建等。

读者可能已经发现，镜像是使用 Docker 的前提，也是最重要的资源。所以，在平时的 Docker 使用中，推荐大家注意积累定制的镜像文件，并将自己创建的高质量镜像分享到社区中。

在后续章节，笔者将会介绍更多对镜像进行操作的场景。