互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

第1章 简介

在计算世界中，容器拥有一段漫长且传奇的历史。容器与管理程序虚拟化（hypervisor virtualization，HV）有所不同，管理程序虚拟化通过中间层将一台或多台独立的机器虚拟运行于物理硬件之上，而容器则是直接运行在操作系统内核之上的用户空间。因此，容器虚拟化也被称为“操作系统级虚拟化”，容器技术可以让多个独立的用户空间运行在同一台宿主机上。

由于“客居”于操作系统，容器只能运行与底层宿主机相同或相似的操作系统，这看起来并不是非常灵活。例如，可以在 Ubuntu 服务器中运行 RedHat Enterprise Linux，但却无法在 Ubuntu 服务器上运行 Microsoft Windows。

相对于彻底隔离的管理程序虚拟化，容器被认为是不安全的。而反对这一观点的人则认为，由于虚拟机所虚拟的是一个完整的操作系统，这无疑增大了攻击范围，而且还要考虑管理程序层潜在的暴露风险。

尽管有诸多局限性，容器还是被广泛部署于各种各样的应用场合。在超大规模的多租户服务部署、轻量级沙盒以及对安全要求不太高的隔离环境中，容器技术非常流行。最常见的一个例子就是“权限隔离监牢”（chroot jail），它创建一个隔离的目录环境来运行进程。如果权限隔离监牢中正在运行的进程被入侵者攻破，入侵者便会发现自己“身陷囹圄”，因为权限不足被困在容器创建的目录中，无法对宿主机进行进一步的破坏。

最新的容器技术引入了 OpenVZ、Solaris Zones 以及 Linux 容器（如 lxc）。使用这些新技术，容器不再仅仅是一个单纯的运行环境。在自己的权限范围内，容器更像是一个完整的宿主机。对 Docker 来说，它得益于现代 Linux 内核特性，如控件组（control group）、命名空间（namespace）技术，容器和宿主机之间的隔离更加彻底，容器有独立的网络和存储栈，还拥有自己的资源管理能力，使得同一台宿主机中的多个容器可以友好地共存。

容器经常被认为是精益技术，因为容器需要的开销有限。和传统的虚拟化以及半虚拟化

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

（paravirtualization）相比，容器运行不需要模拟层（emulation layer）和管理层（hypervisor layer），而是使用操作系统的系统调用接口。这降低了运行单个容器所需的开销，也使得宿主机中可以运行更多的容器。

尽管有着光辉的历史，容器仍未得到广泛的认可。一个很重要的原因就是容器技术的复杂性：容器本身就比较复杂，不易安装，管理和自动化也很困难。而 Docker 就是为改变这一切而生。

1.1　Docker 简介

Docker 是一个能够把开发的应用程序自动部署到容器的开源引擎。由 Docker 公司（www.docker.com，前 dotCloud 公司，PaaS 市场中的老牌提供商）的团队编写，基于 Apache 2.0 开源授权协议发行。

> **注意**
>
> 顺便披露一个小新闻：作者本人就曾在 Docker 工作过，目前是 Docker 公司的顾问。

那么 Docker 有什么特别之处呢？Docker 在虚拟化的容器执行环境中增加了一个应用程序部署引擎。该引擎的目标就是提供一个轻量、快速的环境，能够运行开发者的程序，并方便高效地将程序从开发者的笔记本部署到测试环境，然后再部署到生产环境。Docker 极其简洁，它所需的全部环境只是一台仅仅安装了兼容版本的 Linux 内核和二进制文件最小限的宿主机。而 Docker 的目标就是要提供以下这些东西。

1.1.1　提供一个简单、轻量的建模方式

Docker 上手非常快，用户只需要几分钟，就可以把自己的程序“Docker 化（Dockerize）。Docker 依赖于“写时复制”（copy-on-write）模型，使修改应用程序也非常迅速，可以说达到了“随心所至，代码即改”的境界。

随后，就可以创建容器来运行应用程序了。大多数 Docker 容器只需不到 1 秒钟即可启动。由于去除了管理程序的开销，Docker 容器拥有很高的性能，同时同一台宿主机中也可以运行更多的容器，使用户可以尽可能充分地利用系统资源。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

1.1.2 职责的逻辑分离

使用 Docker，开发人员只需要关心容器中运行的应用程序，而运维人员只需要关心如何管理容器。Docker 设计的目的就是要加强开发人员写代码的开发环境与应用程序要部署的生产环境的一致性，从而降低那种“开发时一切都正常，肯定是运维的问题”的风险。

1.1.3 快速、高效的开发生命周期

Docker 的目标之一就是缩短代码从开发、测试到部署、上线运行的周期，让你的应用程序具备可移植性，易于构建，并易于协作。

1.1.4 鼓励使用面向服务的架构

Docker还鼓励面向服务的架构和微服务架构[①]。Docker推荐单个容器只运行一个应用程序或进程，这样就形成了一个分布式的应用程序模型，在这种模型下，应用程序或服务都可以表示为一系列内部互联的容器，从而使分布式部署应用程序，扩展或调试应用程序都变得非常简单，同时也提高了程序的内省性。

注意

如果你愿意，当然不必拘泥于这种模式，你可以轻松地在一个容器内运行多个进程的应用程序。

1.2 Docker 组件

我们来看看 Docker 的核心组件：

- Docker 客户端和服务器；
- Docker 镜像；
- Registry；
- Docker 容器。

① http://martinfowler.com/articles/microservices.html

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

1.2.1 Docker 客户端和服务器

Docker是一个客户-服务器（C/S）架构的程序。Docker客户端只需向Docker服务器或守护进程发出请求，服务器或守护进程将完成所有工作并返回结果。Docker提供了一个命令行工具docker以及一整套RESTful API[①]。你可以在同一台宿主机上运行Docker守护进程和客户端，也可以从本地的Docker客户端连接到运行在另一台宿主机上的远程Docker守护进程。图 1-1 描绘了Docker的架构。

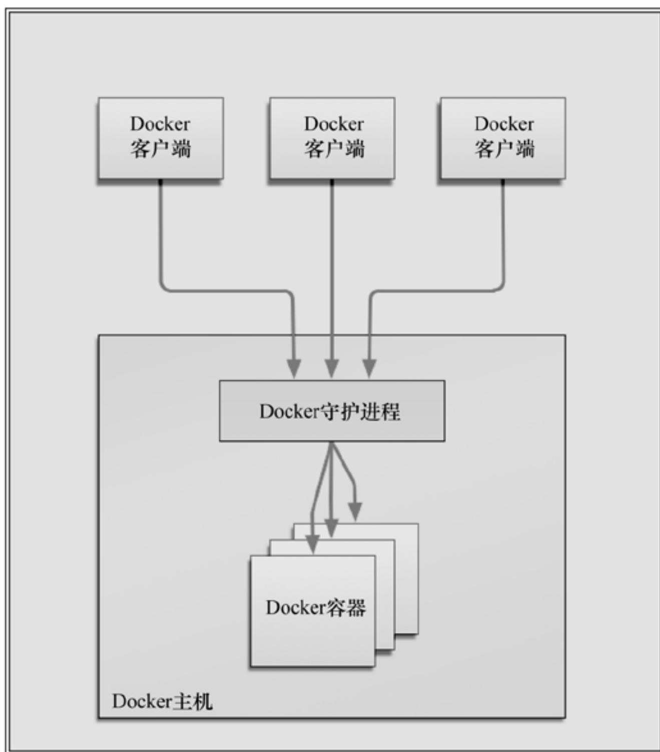

图 1-1 Docker 架构

1.2.2 Docker 镜像

镜像是构建 Docker 世界的基石。用户基于镜像来运行自己的容器。镜像也是 Docker 生

① http://docs.docker.com/reference/api/docker_remote_api/

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

命周期中的“构建”部分。镜像是基于联合（Union）文件系统的一种层式的结构，由一系列指令一步一步构建出来。例如：

- 添加一个文件；
- 执行一个命令；
- 打开一个端口。

也可以把镜像当作容器的“源代码”。镜像体积很小，非常“便携”，易于分享、存储和更新。在本书中，我们将会学习如何使用已有的镜像，同时也会尝试构建我们自己的镜像。

1.2.3 Registry

Docker用Registry来保存用户构建的镜像。Registry分为公共和私有两种。Docker公司运营的公共Registry叫做Docker Hub。用户可以在Docker Hub[①]注册账号[②]，分享并保存自己的镜像。

根据最新统计，Docker Hub上有超过 10 000 注册用户构建和分享的镜像。需要Nginx Web服务器[③]的Docker镜像，或者Asterix开源PABX系统[④]的镜像，抑或是MySQL数据库[⑤]的镜像？这些镜像在Docker Hub上都有，而且具有多种版本。

你也可以在 Docker Hub 上保存自己的私有镜像。例如，包含源代码或专利信息等需要保密的镜像，或者只在团队或组织内部可见的镜像。

你甚至可以架设自己的私有 Registry。具体方法我们会在第 4 章中讨论。私有 Registry 可以受到防火墙的保护，将镜像保存在防火墙后面，以满足一些组织的特殊需求。

1.2.4 容器

Docker 可以帮你构建和部署容器，你只需要把自己的应用程序或服务打包放进容器即可。我们刚刚提到，容器是基于镜像启动起来的，容器中可以运行一个或多个进程。我们可以认为，镜像是 Docker 生命周期中的构建或打包阶段，而容器则是启动或执行阶段。

---

① http://hub.docker.com/
② https://hub.docker.com/account/signup/
③ https://hub.docker.com/search?q=nginx
④ https://hub.docker.com/search?q=Asterisk
⑤ https://hub.docker.com/search?q=mysql

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

总结起来，Docker 容器就是：

- 一个镜像格式；
- 一系列标准的操作；
- 一个执行环境。

Docker 借鉴了标准集装箱的概念。标准集装箱将货物运往世界各地，Docker 将这个模型运用到自己的设计哲学中，唯一不同的是：集装箱运输货物，而 Docker 运输软件。

每个容器都包含一个软件镜像，也就是容器的“货物”，而且与真正的货物一样，容器里的软件镜像可以进行一些操作。例如，镜像可以被创建、启动、关闭、重启以及销毁。

和集装箱一样，Docker 在执行上述操作时，并不关心容器中到底塞进了什么，它不管里面是 Web 服务器，还是数据库，或者是应用程序服务器什么的。所有容器都按照相同的方式将内容“装载”进去。

Docker 也不关心你要把容器运到何方：你可以在自己的笔记本中构建容器，上传到 Registry，然后下载到一个物理的或者虚拟的服务器来测试，再把容器部署到 Amazon EC2 主机的集群中去。像标准集装箱一样，Docker 容器方便替换，可以叠加，易于分发，并且尽量通用。

使用 Docker，我们可以快速构建一个应用程序服务器、一个消息总线、一套实用工具、一个持续集成（continuous integration，CI）测试环境或者任意一种应用程序、服务或工具。我们可以在本地构建一个完整的测试环境，也可以为生产或开发快速复制一套复杂的应用程序栈。可以说，Docker 的应用场景相当广泛。

1.3 我们能用 Docker 做什么

那么，我们为什么要关注 Docker 或容器技术呢？我们前面已经简单地讨论了容器提供的隔离性，结论是，容器可以为各种测试提供很好的沙盒环境。并且，容器本身就具有“标准性”的特征，非常适合为服务创建构建块。Docker 的一些应用场景如下。

- 加速本地开发和构建流程，使其更加高效、更加轻量化。本地开发人员可以构建、运行并分享 Docker 容器。容器可以在开发环境中构建，然后轻松地提交到测试环境中，并最终进入生产环境。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

- 能够让独立服务或应用程序在不同的环境中，得到相同的运行结果。这一点在面向服务的架构和重度依赖微型服务的部署中尤其实用。
- 用 Docker 创建隔离的环境来进行测试。例如，用 Jenkins CI 这样的持续集成工具启动一个用于测试的容器。
- Docker 可以让开发者先在本机上构建一个复杂的程序或架构来进行测试，而不是一开始就在生产环境部署、测试。
- 构建一个多用户的平台即服务（PaaS）基础设施。
- 为开发、测试提供一个轻量级的独立沙盒环境，或者将独立的沙盒环境用于技术教学，如 Unix shell 的使用、编程语言教学。
- 提供软件即服务（SaaS）应用程序，如Memcached即服务[①]。
- 高性能、超大规模的宿主机部署。

本书为读者提供了一个基于和围绕 Docker 生态环境构建的早期项目列表，详情请查看 http://blog.docker.com/2013/07/docker-projects-from-the-docker-community/。

1.4 Docker 与配置管理

从Docker项目公布以来，已经有大量关于"哪些配置管理工具适用于Docker"的讨论，如Puppet、Chef。Docker包含一套镜像构建和镜像管理的解决方案。现代配置管理工具的原动力之一就是"黄金镜像"模型[②]。然而，使用黄金镜像的结果就是充斥了大量、无管理状态的镜像：已部署或未部署的复杂镜像数量庞大，版本状态混乱不堪。随着镜像的使用，不确定性飞速增长，环境中的混乱程度急剧膨胀。镜像本身也变得越来越笨重。最终不得不手动修正镜像中不符合设计和难以管理的配置层，因为底层的镜像缺乏适当的灵活性。

与传统的镜像模型相比，Docker 就显得轻量多了：镜像是分层的，你可以对其进行迅速的迭代。数据表明，Docker 的这些特性确实能够减轻许多传统镜像管理中的麻烦。现在还难以确定 Docker 是否可以完全取代配置管理工具，但是从幂等性和内省性来看，Docker 确实能够获得非常好的效果。Docker 本身还是需要在主机上进行安装、管理和部署的。而

---

① http://www.memcachedasaservice.com/
② https://web.archive.org/web/20090207105003/http://madstop.com/2009/02/04/golden-image-or-foil-ball

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

主机也需要被管理起来。这样，Docker 容器需要编配、管理和部署，也经常需要与外部服务和工具进行通信，而这些恰恰是配置管理工具所擅长的。

Docker 一个显著的特点就是，对不同的宿主机、应用程序和服务，可能会表现出不同的特性与架构（或者确切地说，Docker 本就是被设计成这样的）：Docker 可以是短生命周期的，但也可以用于恒定的环境，可以用一次即销毁，也可以提供持久的服务。这些行为并不会给 Docker 增加复杂性，也不会和配置管理工具的需求产生重合。基于这些行为，我们基本不需要担心管理状态的持久性，也不必太担心状态的复杂性，因为容器的生命周期往往比较短，而且重建容器状态的代价通常也比传统的状态修复要低。

然而，并非所有的基础设施都具备这样的“特性”。在未来的一段时间内，Docker 这种理想化的工作负载可能会与传统的基础设备部署共存一段时间。长期运行的主机和物理设备上运行的主机在很多组织中仍具有不可替代的地位。由于多样化的管理需求，以及管理 Docker 自身的需求，在绝大多数组织中，Docker 和配置管理工具可能都需要部署。

1.5 Docker 的技术组件

Docker 可以运行于任何安装了现代 Linux 内核的 x64 主机上。我们推荐的内核版本是 3.8 或者更高。Docker 的开销比较低，可以用于服务器、台式机或笔记本。它包括以下几个部分。

一个原生的Linux容器格式，Docker中称为`libcontainer`，或者很流行的容器平台lxc[①]。`libcontainer`格式现在是Docker容器的默认格式。

Linxu内核的命名空间（namespace）[②]，用于隔离文件系统、进程和网络。

- 文件系统隔离：每个容器都有自己的 root 文件系统。
- 进程隔离：每个容器都运行在自己的进程环境中。
- 网络隔离：容器间的虚拟网络接口和 IP 地址都是分开的。
- 资源隔离和分组：使用cgroups[③]（即control group，Linux的内核特性之一）将CPU和内存之类的资源独立分配给每个Docker容器。

---

① http://lxc.sourceforge.net/
② http://lwn.net/Articles/531114/
③ http://en.wikipedia.org/wiki/Cgroups

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

- 写时复制[1]：文件系统都是通过写时复制创建的，这就意味着文件系统是分层的、快速的，而且占用的磁盘空间更小。
- 日志：容器产生的 STDOUT、STDERR 和 STDIN 这些 IO 流都会被收集并记入日志，用来进行日志分析和故障排错。
- 交互式 shell：用户可以创建一个伪 tty 终端，将其连接到 STDIN，为容器提供一个交互式的 shell。

1.6　本书的内容

在本书中，我们将讲述如何安装、部署、管理 Docker，并对其进行功能扩展。我们首先会向大家介绍 Docker 的基础知识及其组件，然后用 Docker 构建容器和服务，来完成各种的任务。

我们还会体验从测试到生产环境的完整开发生命周期，并会探讨 Docker 适用于哪些领域，Docker 是如何让我们的生活更加简单的。我们使用 Docker 为新项目构建测试环境，演示如何将 Docker 集成到持续集成工作流，如何构建程序应用的服务和平台。最后，我们会向大家介绍如何使用 Docker 的 API，以及如何对 Docker 进行扩展。

我们将会教大家如何：

- 安装 Docker；
- 尝试使用 Docker 容器；
- 构建 Docker 镜像；
- 管理并共享 Docker 镜像；
- 运行、管理更复杂的 Docker 容器；
- 将 Docker 容器的部署纳入测试流程；
- 构建多容器的应用程序和环境；
- 介绍使用 Fig 进行 Docker 编配的基础；

① http://en.wikipedia.org/wiki/Copy-on-write

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

- 探索 Docker 的 API；
- 获取帮助文档并扩展 Docker。

推荐读者按顺序阅读本书。每一章都会以前面章节的 Docker 知识为基础，并引入新的特性和功能。读完本书后，读者应该会对如何使用 Docker 构建标准容器，部署应用程序、测试环境和独立的服务有比较深刻的理解。

1.7　Docker 资源

- Docker 官方主页（http://www.docker.com/）。
- Docker Hub（http://hub.docker.com）。
- Docker 官方博客（http://blog.docker.com/）。
- Docker 官方文档（http://docs.docker.com/）。
- Docker 快速入门指南（http://www.docker.com/tryit/）。
- Docker 的 GitHub 源代码（https://github.com/docker/docker）。
- Docker Forge（https://github.com/dockerforge）：收集了各种 Docker 工具、组件和服务。
- Docker 邮件列表（https://groups.google.com/forum/#!forum/docker-user）。
- Docker 的 IRC 频道（irc.freenode.net）。
- Docker 的 Twitter 主页（http://twitter.com/docker）。
- Docker 的 StackOverflow 问答主页（http://stackoverflow.com/search?q=docker）。
- Docker 官网（http://www.docker.com/）。

除这些资源之外，我们在第 9 章中会详细介绍去哪里以及如何获得 Docker 的帮助信息。