Docker容器应用技术

DOCKER RONGQI YINGYONG JISHU

王勇　黄佳　主编

重庆大学电子音像出版社
http//www.cqup.com.cn

内容提要

本书主要讲解了 Docker 简介与安装、Docker 镜像与容器、Docker 仓库、Docker 常用镜像及使用、Dockerfile 构建、Docker 网络与存储、Docker 容器编排集群和 Docker 容器管理平台。通过本书的学习，学生能掌握 Docker 容器的知识和实际应用。

版权信息

书名：Docker 容器应用技术

主编：王勇 黄佳

重庆大学电子音像出版社

责任编辑：田雨

地址：重庆市沙坪坝区大学城西路 21 号

ISBN 978-7-89446-647-1

出版时间：2022 年 9 月

邮编：401331

电话：023-88617016

字数：288 千字

定价：62.00 元

前 言

欢迎学习本系列课程（新形态教材），课程研发团队由教学经验丰富的一线老师、企业中工作经验丰富的行业专家及教育专家组成。

本系列课程作为新形态教材，广泛利用多媒体等新兴技术辅助教学，让学生寓教于乐，提高学习的兴趣和效率；同时，充分考虑了读者的阅读习惯和学习习惯，在编排上做了非常科学的安排：

- 本系列课程为作者团队花费了大量的人力、物力和财力倾力打造的新形态教材，全系采用“二维码链接配套资源”的新形态教材模式，每本教材都拥有视频教学资源、评估试题等配套教学资源，可以通过嵌入到教材中的二维码轻松查看配套资源，让学习变得高效、有趣又轻松。

- 本书包含学习目标、课程内容、总结、作业等，这个编排结构可以让读者更加轻松、高效地学习。一来可以提高理论的应用能力，二来可以巩固所学的理论知识，锻炼读者自己解决问题和举一反三的能力。

- 课程内容中含有实战案例，让读者在提高应用能力的同时获得实战经验，真正体现了学以致用的指导方针。

- 采用图文结合的编排方式，宽松的版式让读者可以轻松阅读。

本系列课程由大量的老师及专家给予支持和帮助，由于参与本系列课程研发的人数太多，在这里没有一一列出他们的名字，在此由衷地感谢他们！本课程中使用的图例和片段仅用于教学示范和讲解，不做其他商业用途。在编写过程中，有一些图例和片段无法确定作者与出处，在此也向他们深表感谢，并请原作者与出版社或主编本人联系。同时希望读者和同行人士多提宝贵意见和建议。

本系列课程适合教学使用，也适合自学使用。

编　者

2022 年 4 月 15 日

评估试题参考答案

案例资源

目录

第1章 Docker 简介与安装

教学资源

学习目标

了解什么是 Docker

了解虚拟机安装 CentOS 7

了解虚拟机安装 Ubuntu

掌握在 Linux 和 Windows 中安装 Docker

掌握基本运行 hello-world 容器

本章单词

请在预习前完成下列单词，并将单词写在横线上。

① docker(['dɒkə(r)] 容器)：_____

② Linux(['lɪnəks] 操作系统)：_____

预习任务

1. Docker 的优势有_____、_____、_____、_____。（参考本章 1.1.2）

2. 在 CentOS 系统中，安装软件的工具是（　　）。（参考本章 1.4）

   A.install

   B.yum

   C.atp

   D.apt

1.1 Docker 简介

1.1.1 什么是 Docker

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（前 dotCloud 公司，PaaS 市场中的老牌提供商）的团队编写，基于 Apache2.0 开源授权协议发行。

Docker 是基于 Go 语言实现的开源容器项目，诞生于 2013 年年初，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（包括 Docker 三剑客、Kubernetes 等），逐渐形成了围绕 Docker 容器的生态体系。由于 Docker 在业界造成的影响力实在太大，dotCloud 公司后来也直接改名为 Docker Inc，并专注于 Docker 相关技术和产品的开发。

Docker 项目已加入了 Linux 基金会，并遵循 Apache2.0 协议。在 Linux 基金会最近一次关于“最受欢迎的云计算开源项目”的调查中，Docker 仅次于 2010 年发起的 OpenStack 项目，并仍处于上升趋势。

现在主流的 Linux 操作系统都已经支持 Docker。例如，红帽公司的 RHEL 6.5/CentOS 6.5 往上的操作系统、Ubuntu 14.04 往上的操作系统，都已经在软件源中默认带有 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 容器。IBM 公司跟 Docker 公司达成了战略合作伙伴关系。微软公司在其云平台 Azure 上加强了对 Docker 的支持。公有云提供商亚马逊也推出了 AWS EC2 Container 服务，提供对 Docker 和容器业务的支持。

Docker 的构想是要实现“Build，Ship and Run Any App，Anywhere”，即通过对应用的封装（Packaging）、分发（Distribution）、部署（Deployment）、运行（Runtime）生命周期进行管理，达到应用组件“一次封装，到处运行”的目的。这里的应用组件，既可以是一个 Web 应用、一个编译环境，也可以是一套数据库平台服务，甚至是一个操作系统或集群。

基于 Linux 平台上的多项开源技术，Docker 提供了高效、敏捷和轻量级的容器方案，并支持部署到本地环境和多种主流云平台。可以说，Docker 首次为应用的开发、运行和部署提供了“一站式”的实用解决方案。

跟大部分新兴技术的诞生一样，Docker 也并非“从石头缝里蹦出来的”，而是站在前人的肩膀上，其中最重要的就是 Linux 容器（Linux Containers，LXC）技术。IBM DeveloperWorks 网站关于容器技术的描述十分准确：“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。”

当然，LXC 也经历了长期的演化。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 和 OpenVZ。

在 LXC 之前，这些相关技术经过多年的演化已经十分成熟和稳定，但是由于种种原因，它们并没有被很好地集成到主流的 Linux 内核中，用户使用起来并不方便。例如，如果用户要使用 OpenVZ 技术，需要先手动给操作系统打上特定的内核补丁方可使用，而且不同版本并不一致。类似的困难造成在很长一段时间内，这些优秀的技术只流传于技术人员的小圈子中。

后来 LXC 项目借鉴了前人成熟的容器设计理念，并基于一系列新引入的内核特性，实现了更具扩展性的虚拟化容器方案。更加关键的是，LXC 终于被集成到了主流 Linux 内核中，进而成为 Linux 系统轻量级容器技术的事实标准。从技术层面来看，LXC 已经蹚过了绝大部分的“坑”，完成了容器技术实用化的大半历程。

在 LXC 的基础上，Docker 进一步优化了容器的使用体验，让它进入了寻常百姓家。首先，Docker 提供了各种容器管理工具（如分发、版本、移植等）让用户无须关注底层的操作，可以更简单明了地管理和使用容器；其次，Docker 通过引入分层文件系统构建和高效的镜像机制，降低了迁移难度，极大地提升了用户体验。用户操作 Docker 容器就像操作应用自身一样简单。

早期的 Docker 代码实现是直接基于 LXC 的。自 0.9 版本开始，Docker 开发了 libcontainer 项目，作为更广泛的容器驱动实现，从而替换掉了 LXC 的实现。目前，Docker 还积极推动成立了 RunC 标准项目，试图让容器支持不再局限于 Linux 操作系统，而是更安全、更具扩展性。简单地讲，可以将 Docker 容器理解为一种轻量级的沙盒（sandbox）。每个容器内运行着一个应用，不同的容器相互隔离，容器之间也可以通过网络互相通信。容器的创建和停止都十分快速，几乎跟创建和终止原生应用一致；另外，容器自身对系统资源的额外需求也十分有限，远远低于传统虚拟机。很多时候，甚至直接把容器当作应用本身也没有任何问题。相信 Docker 技术会进一步成熟，将成为更受欢迎的容器虚拟化技术，并在云计算和 DevOps 等领域得到更广泛的应用。

1.1.2 Docker 的优势

Docker 容器虚拟化的好处：Docker 项目的发起人和 Docker 公司 CTO Solomon Hykes 曾认为，Docker 在正确的地点、正确的时间顺应了正确的趋势——如何正确地构建应用。

在云时代，开发者创建的应用必须要能很方便地在网络上传播，也就是说应用必须脱离底层物理硬件的限制；同时必须是“任何时间、任何地点”可获取的。因此，开发者需要一种新型的创建分布式应用程序的方式，快速分发和部署，这正是 Docker 所能够提供的最大优势。

举个简单的例子，假设用户试图基于最常见的 LAMP（Linux+Apache+MySQL+PHP）组合来构建一个网站。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，则进行调试追踪，意味着更多的时间代价和不可控的风险。可以想象，如果应用数目变多，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（例如从亚马逊云迁移到其他云），往往需要对每个应用都进行重新部署和调试。这些琐碎而无趣的“体力活”，极大地降低了工作效率。究其根源，是这些应用直接运行在底层操作系统上，无法保证同一份应用在不同的环境中行为一致。而 Docker 提供了一种更为聪明的方式，通过容器来打包应用，解耦应用和运行平台。意味着迁移的时候，只需要在新的服务器上启动需要的容器就可以了，无论新旧服务器是否是同一类型的平台。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，可能最梦寐以求的效果就是一次创建或配置，之后可以在任意地方、任意时间让应用正常运行。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：

- 更快速地交付和部署。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完全相同环境来部署代码。只要开发测试过的代码，就可以确保在生产环境无缝运行。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，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。跟传统虚拟机方式相比，要提高一到两个数量级。

- 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，同时支持主流的操作系统发行版本。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被分发和更新，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 在运行应用上与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，而传统的虚拟机方式需要数分钟。

- 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（在 IBM 服务器上已经实现了同时运行 10K 量级的容器实例）。

- Docker 通过类似 Git 设计理念的操作来方便用户获取、分发和更新应用镜像，存储复用，增量更新。
- Docker 通过 Dockerfile 支持灵活的自动化创建和部署机制，提高工作效率，使流程标准化。

Docker 容器除了运行其中应用外，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 个不同的应用就要起 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 个隔离得“很薄的”容器，并将应用放进容器内即可。应用获得的是接近原生的运行性能，如图 1.1 所示。

特　性	容　器	虚拟机
启动速度	秒级	分钟级
性能	接近原生	较弱
内存代价	很小	较多
硬盘使用	一般为 MB	一般为 GB
运行密度	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
迁移性	优秀	一般

图 1.1

当然，在隔离性方面，传统的虚拟机方式提供的是相对封闭的隔离。但这并不意味着 Docker 就不安全，Docker 利用 Linux 系统上的多种防护技术实现了严格的隔离可靠性，并且可以整合众多安全工具。从 1.3. 0 版本开始，Docker 重点改善了容器的安全控制和镜像的安全机制，极大提高了使用 Docker 的安全性。在已知的大规模应用中，目前尚未出现值得担忧的安全隐患。

图 1.1 总结了使用 Docker 容器技术与传统虚拟机技术的特性比较，可见容器技术在很多应用场景下都具有巨大的优势。

1.1.3 Docker 与虚拟化

虚拟化（Virtualization）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：“虚拟化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的抽象，目标往往是为了在同一个主机上同时运行多个系统或应用，从而提高系统资源的利用率，并且带来降低成本、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诸如 Wine 这样的软件。后者又可以细分为如下几个子类：

(1) 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，客户操作系统无须进行修改。例如 IBM p 和 z 系列的虚拟化、VMware Workstation、VirtualBox、QEMU 等。

(2) 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，客户操作系统无须修改，例如 VMware Workstation、Xen、KVM。

(3) 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。

(4) 准虚拟化（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。

(5) 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，都属于操作系统虚拟化这个范畴，操作系统虚拟化最大的特点就是不需要额外的 supervisor 支持。

Docker 虚拟化方式之所以有众多优势，这与操作系统虚拟化技术自身的设计和实现是分不开的，如图 1.2 所示。

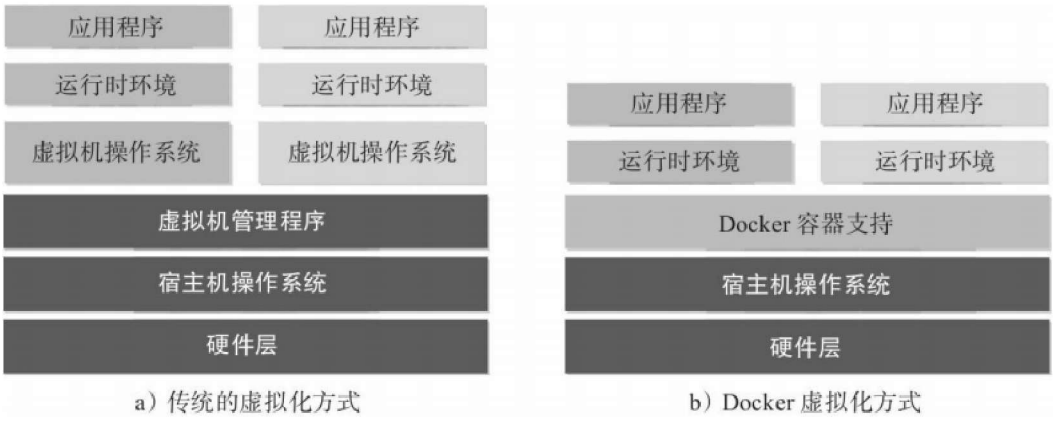

图 1.2 Docker 和传统的虚拟化方式的区别

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。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。

1.2 虚拟机的使用

1.2.1 在虚拟机中安装 CentOS 7

在虚拟机软件（虚拟机软件使用的是 VMware Workstation 15.5）中点击“文件”，选择“新建虚拟机”，如图 1.3 所示。

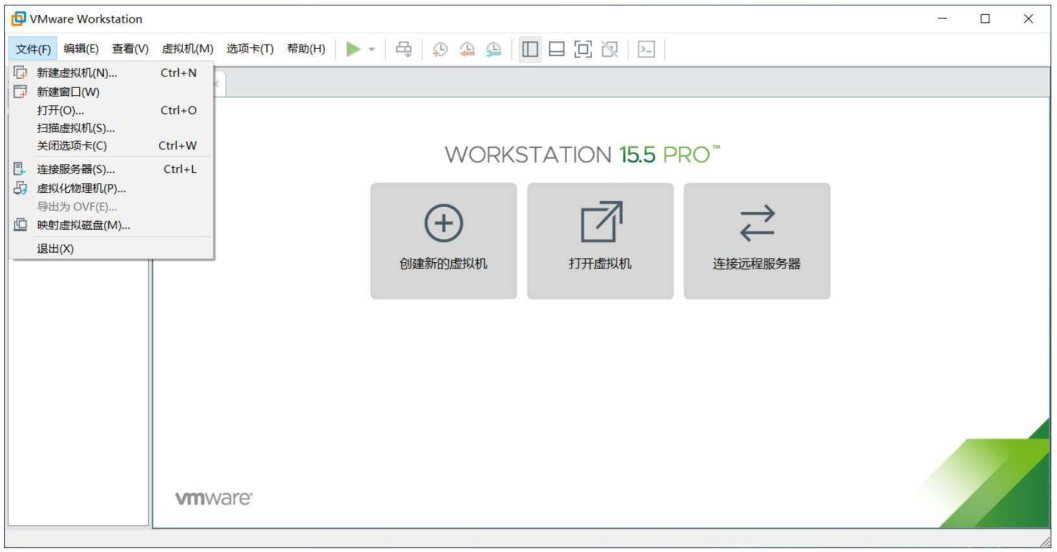

图 1.3 新建虚拟机

在弹出的菜单中选择“自定义(高级)”，点击下一步，选择“稍后安装操作系统”，如图 1.4 所示。

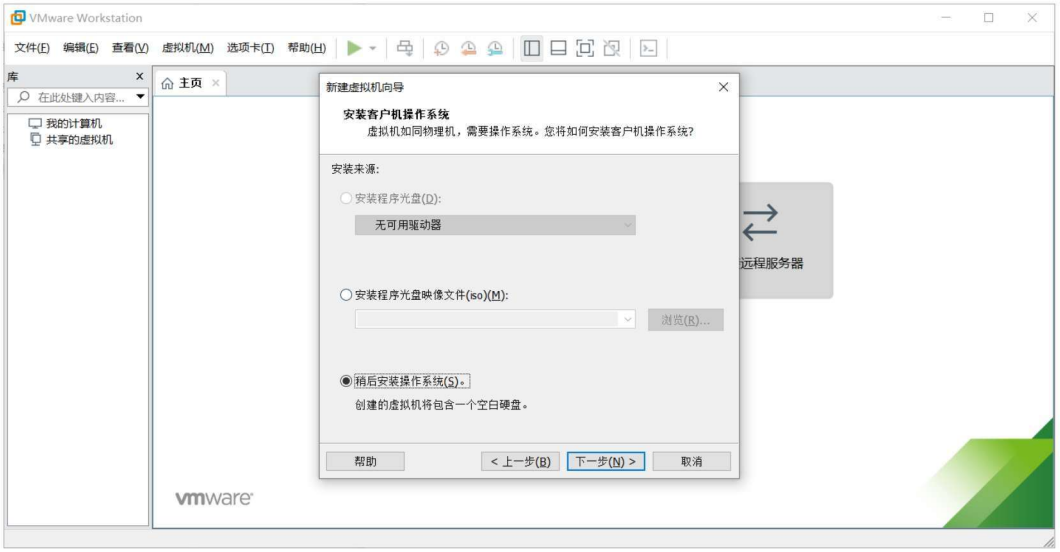

图 1.4

再点击下一步，在选择客户机操作系统菜单中选择“Linux”选项，并在下拉菜单中选中“CentOS 7 64 位”，如图 1.5 所示。

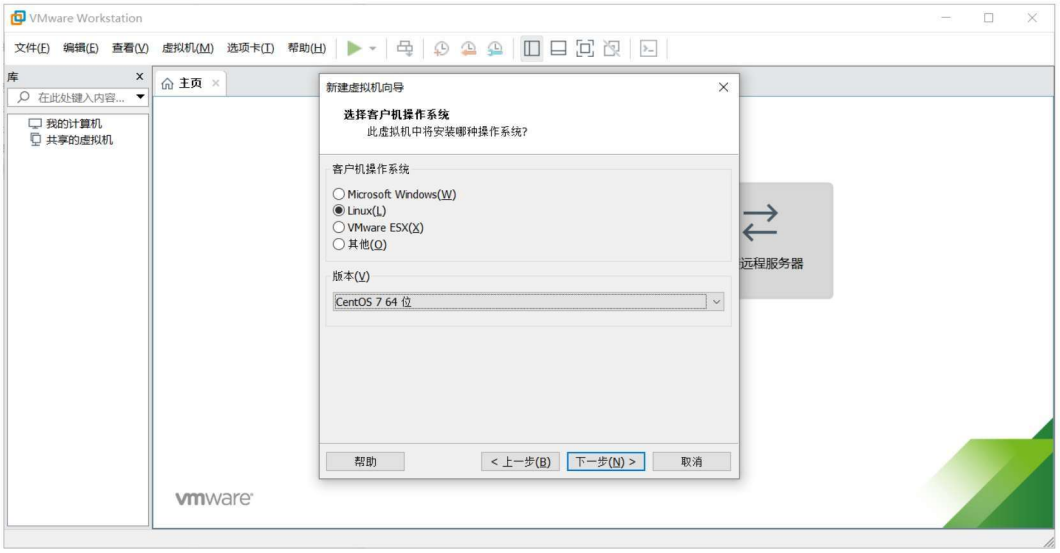

图 1.5

点击下一步继续，为客户机虚拟机命名并选择文件存放的位置进入下一步，为客户机分配处理器数量(注意数量不要超过自己电脑的处理器数量)，如图 1.6 所示。

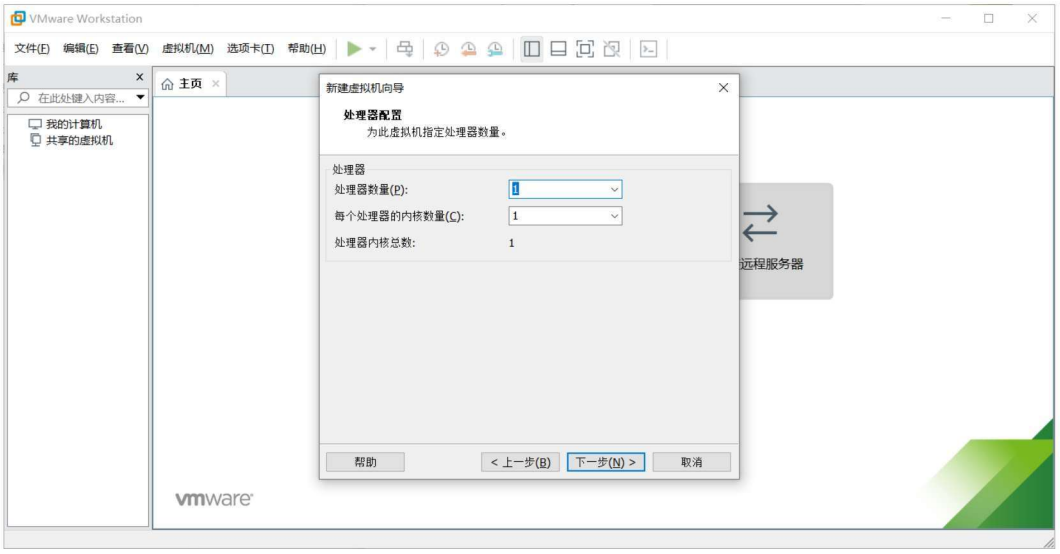

图 1.6