第3章 Docker 容器化引擎

本章学习目标

- 了解 Docker 的常用命令
- 理解 Docker 与传统虚拟化方式的区别
- 掌握 Docker 的安装
- 掌握 Docker 中镜像、容器的使用
- 掌握在 Docker 容器中部署项目

如今 Docker 在一线互联网公司的应用已经非常普遍，使用 Docker 技术可以给企业带来极大的好处，使企业的业务水平扩展更快速，从而到达弹性部署业务的能力。在云服务概念兴起之后，Docker 的使用场景和范围得到了进一步发展，如今在微服务架构越来越流行的情况下，微服务＋Docker 的组合方式，更加方便微服务架构运维部署落地，使应用部署、测试和请求都变得前所未有的便捷和轻松。

3.1 Docker 与传统虚拟化方式的区别

百度百科中对 Docker 的解释是“Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。”它是谷歌公司基于 Linux 内核，通过 GO 语言开发的。Docker 技术与虚拟机技术相比，前者更加轻便、快捷。之前在安装 Ubuntu 时就已经使用过虚拟机技术 VMware，接下来，把 Docker 技术与传统虚拟化技术进行比较，了解 Docker 的优点。

传统的基础服务设施是由硬件和操作系统组成的，然后在操作系统上再安装 QQ、微信、浏览器等软件，如图 3.1 所示。

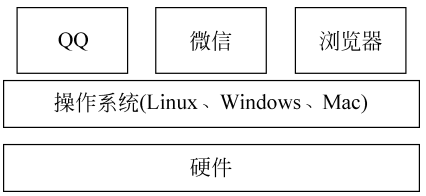

图 3.1 基础设施

传统的虚拟化技术可以将众多的计算机结合在一起统一管理，在这个过程中需要用到虚拟化技术(Hypervisor)。基于硬件，通过虚拟化技术将多台主机连接在一起，共同进行管

理，如图 3.2 所示。

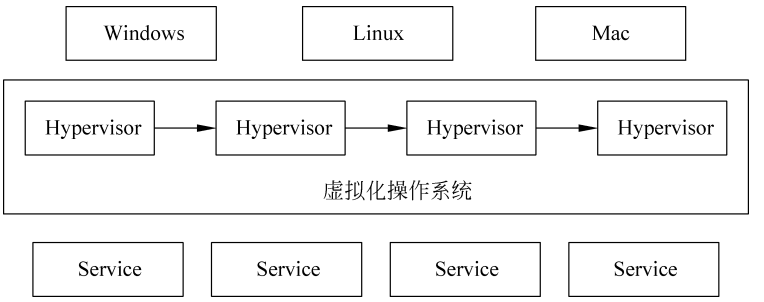

图 3.2　虚拟化操作系统

图 3.2 类似一个云服务平台，假设该平台有四台 Service（计算机），这四台机器通过 Hypervisor（虚拟化技术），整体虚拟化操作系统，最终组成一个超级计算机。该超级计算机能够统一调度资源，统一进行管理，能够分配给用户一定量的空间供用户使用，然后每个用户可以操作自己所分配的那块空间，且与其他用户得到的空间中存储的信息和分配的资源互不相干。

虚拟机也是同样的道理，先是在硬件之上安装宿主机，在宿主机之上使用 Hypervisor 实现虚拟化技术，然后再安装客户机。以通常使用的计算机为例，在最底层的硬件之上安装 Windows 操作系统，然后在操作系统之上使用虚拟化技术（安装 VMware），最后再安装 Ubuntu，用户可以根据自己的需求，在 Ubuntu 之上安装应用程序，如图 3.3(a)所示。

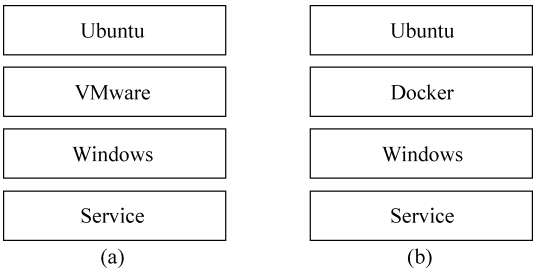

图 3.3　虚拟化技术

然而，从图 3.3(b)中可以发现，有了 Docker 技术后将不再使用虚拟机，Ubuntu 直接安装在 Docker 之上即可。虽然图 3.3(a)和图 3.3(b)两部分看似差别不大，只是用 Docker 取代虚拟机而已，但是，在使用 Docker 时可以发现，Docker 直接利用的是宿主机（图 3.3 中安装了 Windows 的机器）的内核与资源。安装在 Docker 之上的 Linux 的所有资源都来自于底层设备，这个时候它的资源是共享状态的，即 Windows 中的资源与 Docker 之上安装的 Ubuntu 的资源之间可以共享。而虚拟机安装后则会从宿主机中占用部分资源，而且这部分资源与主机之间是不能够共享的。如有一台宿主机，CPU 参数为 3.0GHz 4 核，内存 16GB，当在该宿主机中安装虚拟机（VMware）后，一个虚拟机开启后将被分配 2GB 内存，此时宿主机能够使用的内存只剩 14GB，如果在虚拟机上安装一个 App，当 App 的使用资源超出 2GB 时，还会发生内存溢出，当 App 的使用资源只有 1GB 时，宿主机分配给它的资源将会浪费掉一半。假如不安装虚拟机，使用 Docker 的情况时，宿主机不需要单独为 Docker 分

配任何资源，Docker 可以直接使用宿主机的内存，届时不必担心为 Docker 分配的资源大小是否合适、资源是否因为分配不均而导致浪费问题，只要是宿主机的资源即可全部拿来使用。

3.2 Docker 容器化引擎

简单来说，Docker 就是使用客户端-服务器(C/S)架构模式，通过远程 API 管理和创建容器的。Docker 在容器的基础之上，会进一步地封装，把文件系统、网络互联、进程隔离等都封装在不同的层面之上，这些都极大地简化了 Docker 容器的创建与维护。Docker 容器化引擎也是一种服务器，这种服务器可以长时间地运行，可以通过 REST API 和守护进程完成通信，如图 3.4 所示。

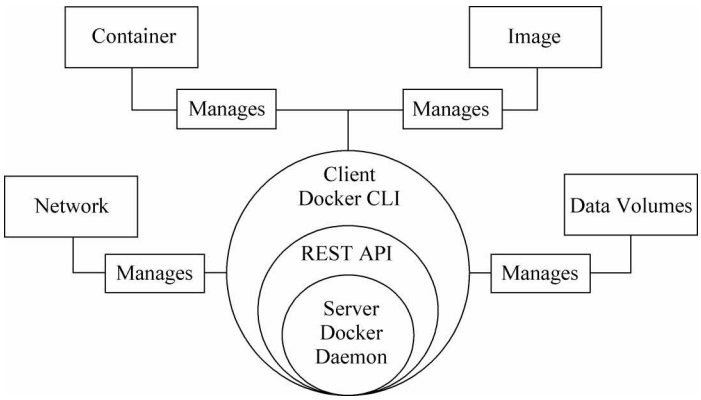

图 3.4 Docker 容器化引擎

图 3.4 中 Server Docker Daemon 即为 Docker 的守护进程，Client Docker CLI 是客户端命令行工具。请求会通过命令行工具去调用 Docker 提供的 REST API，将命令发送给守护进程。通过 API 可以对镜像(Image)、容器(Container)、网络(Network)、数据卷(Data Volumes)进行管理，守护进程最后会把命令的执行结果返回给客户端。

3.3 Docker 镜像、容器、仓库

Docker 镜像是 Docker 容器运行的前提，容器是镜像运行的实例，有了镜像才能够启动容器。Docker 镜像是一个特殊的文件系统，它的内部除了包含提供容器运行时所需要的程序、库、资源、配置等文件外，还包含了容器在运行时需要准备的一些配置参数信息(如匿名卷、环境变量、用户等)。Docker 镜像里面不会包含任何动态数据，它的内容在构建之后也不会被改变，而且，Docker 镜像是分层构建的，前一层是后一层的基础，每一层在构建完成后都不会再发生改变，后一层上的任何改变都只发生在当前这一层上，并不会影响到之前已经构建好的层。例如，在当前层中删除前一层的文件时，实际上并不是真删除了前一层的文件，只是在当前层标记该文件被删除，最终容器运行的时候，虽然看不到该文件，但是实际上该文件会一直跟随镜像。因此，在构建 Docker 镜像时需要特别注意，每一层都要尽量只包

含该层所必须添加的东西，其他额外的东西都应该在该层构建结束前清理掉。通常在构建 Docker 镜像时会把之前构建好的镜像用来作为基础层，然后进一步添加新的层，在新层中定制自己所需要的内容，构建新的镜像，如图 3.5 所示。

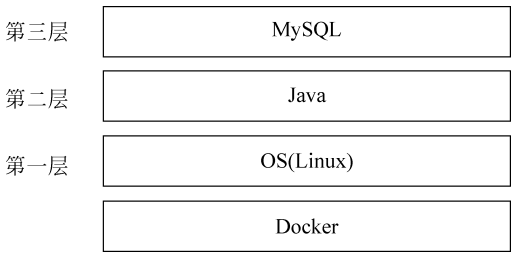

图 3.5　Docker 镜像结构

从图 3.5 可以看出，Docker 引擎之上，安装了操作系统（OS），这时就相当于封装了一层镜像，该层将不再发生变化。在 OS 的基础上再添加一层，第二层当中装载了 Java，由于该层是在第一层 OS 的基础上新加的，因此，该层不仅包含了 Java 也继承了第一层 OS 的 Linux，这时的镜像又再一次进行了封装，当添加到第三层时，可以发现，在构建的新镜像中，不仅包含新安装的 MySQL，还包含了前两层中的 Linux 和 Java。

镜像（Image）和容器（Container）的关系，就像是面向对象编程思想中的类和实例一样，容器是镜像运行时的实体。容器同样也能够被创建、启动、停止、删除、暂停等。每一个容器都是一个进程，但它与宿主机中的进程之间的区别就是容器拥有自己的命名空间。

前面讲过镜像使用的是分层存储，容器也是如此，每一个容器在运行时，都是以镜像为基础层，在基础层的上面会再创建一个存储层用来为容器运行时的读和写做准备。该存储层的生命周期会伴随着容器的消亡而消亡，所以保存在存储层中的信息都会有丢失的风险。因此，所有的文件在写入时，都应该使用数据卷（数据卷是一个可以供一个或多个容器使用的特殊目录）或者绑定宿主目录，这样文件的读写操作会跳过容器存储层，直接对宿主（或网络存储）发生读写，其性能和稳定性更高。

数据卷的生存周期并不会因为容器的消亡而消亡。因此，使用数据卷后，容器无论是被删除还是被重新运行，数据都不会丢失。

镜像构建完成后，可以很容易地在当前宿主机上运行，但是，如果需要在其他服务器上使用这个镜像，就需要一个集中的存储、分发镜像的服务——Docker Registry 服务，如图 3.6 所示。

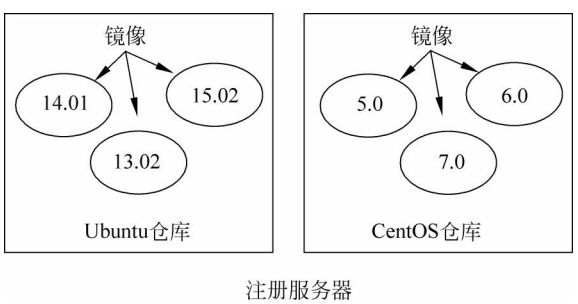

图 3.6　Docker Registry

在 Docker Registry 服务中包含有多个 Repository(集中存放镜像文件的仓库),每个 Repository 又包含了多个 Tag(标签),标签和镜像是一一对应的关系,可以通过<仓库名>:<标签>的格式指定具体的镜像文件。Docker Registry 服务又分为公有的和私有的,公有的 Docker Registry 服务是开放给用户使用、允许用户管理镜像的 Registry 服务,可以满足用户上传和下载镜像的需求,最常使用的 Registry 公有的服务是官方的 Docker Hub,这也是默认的 Registry,并且拥有大量高质量的官方镜像,但是由于种种原因在国内访问这些服务可能会比较慢,国内不少云服务提供商(如时速云、阿里云等)也提供了仓库的本地源,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,将其从仓库上使用 pull 命令拉下来就可以了。

3.4 安装 Docker

在 Ubuntu 上安装 Docker 之前,应先克隆一个新的虚拟机,在克隆的新虚拟机中操作,防止发生错误时改变原环境,如图 3.7 所示。

图 3.7 虚拟机克隆 1

选中原始机器,右击,选择“管理”→“克隆”,然后一直单击“下一步”按钮,如图 3.8～图 3.10 所示。

然后输入新建虚拟机的名称,更改位置,最后单击“完成”按钮,如图 3.11 所示。

开启该虚拟机,以 root 身份登录成功后,输入 ifconfig,获得新地址 192.168.136.133,输出内容如下。

图 3.8　虚拟机克隆 2

图 3.9　虚拟机克隆状态

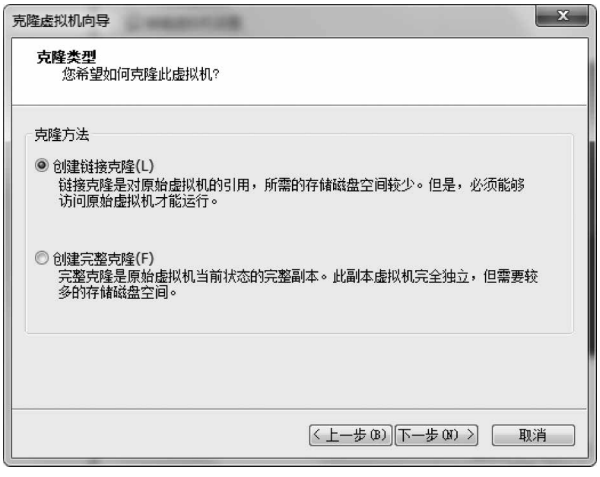

图 3.10　虚拟机克隆方法

图 3.11　新虚拟机命名

```
root@ubuntu:/# ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:d2:44:66
          inet addr:192.168.136.133  Bcast:192.168.136.255
Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fed2:4466/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1118 errors:0 dropped:0 overruns:0 frame:0
          TX packets:338 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:479292 (479.2 KB)  TX bytes:37095 (37.0 KB)
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:160 errors:0 dropped:0 overruns:0 frame:0
          TX packets:160 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:11840 (11.8 KB)  TX bytes:11840 (11.8 KB)
```

使用 Xshell 远程连接,单击“用户身份验证”,输入用户名和密码,如图 3.12 所示。连接成功后,可见 Xshell 中显示如下内容。

```
Xshell for Xmanager Enterprise 5 (Build 0544)
Copyright (c) 2002 - 2015 NetSarang Computer, Inc. All rights reserved.
Type `help' to learn how to use Xshell prompt.
[d:\~]$
Connecting to 192.168.136.134:22...
Connection established.
To escape to local shell, press 'Ctrl + Alt + ]'.
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0 - 21 - generic x86_64)
 * Documentation:  https://help.ubuntu.com/
239 packages can be updated.
150 updates are security updates.
Last login: Mon May 27 11:07:54 2019
```

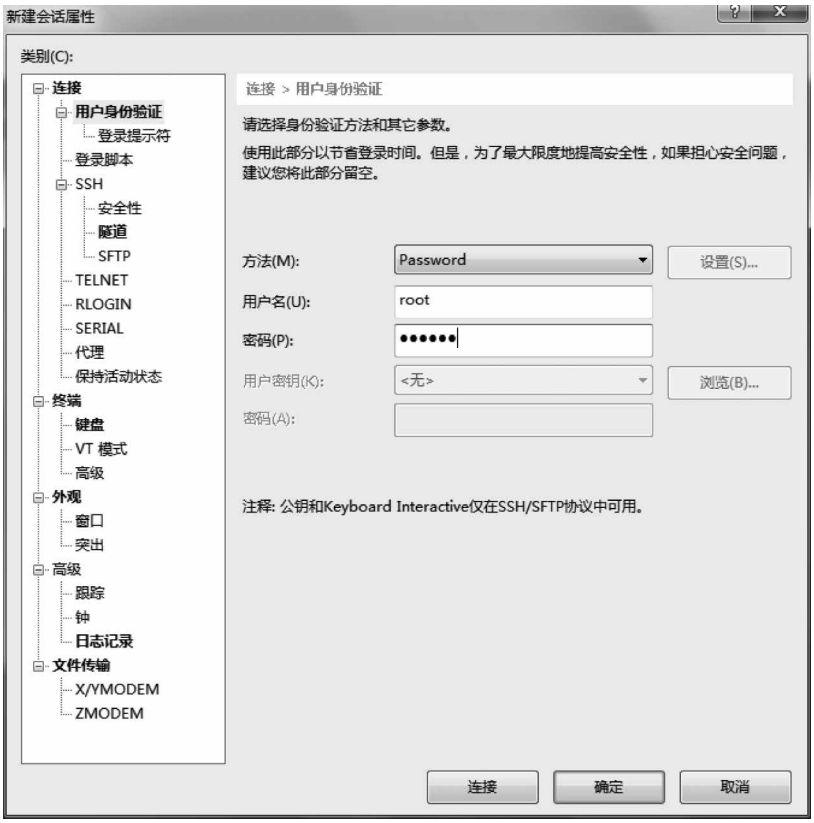

图 3.12　身份验证

3.4.1　安装 curl 工具

输入命令 apt-get update，更新软件包列表，输出如下结果。

```
1  root@ubuntu:/var/lib/dpkg/updates# apt-get update
2  Hit:1 http://mirrors.aliyun.com/ubuntu xenial InRelease
3  Hit:2 http://mirrors.aliyun.com/ubuntu xenial-security InRelease
4  Hit:3 http://mirrors.aliyun.com/ubuntu xenial-updates InRelease
5  Hit:4 http://mirrors.aliyun.com/ubuntu xenial-backports InRelease
6  Hit:5 https://mirror.azure.cn/docker-ce/linux/ubuntu xenial InRelease
7  Reading package lists... Done
8  root@ubuntu:/var/lib/dpkg/updates#
```

此时，输入命令 apt-get install curl，安装 curl 工具，输出如下结果。

```
1  root@ubuntu:~# apt-get install curl
2  Reading package lists... Done
3  Building dependency tree
4  Reading state information... Done
5  curl is already the newest version (7.47.0-lubuntu2.11).
6  0 upgraded, 0 newly installed, 0 to remove and 106 not upgraded
```

3.4.2 通过脚本安装 Docker

安装 Docker 的方法很多，本节将讲述使用脚本的方法安装 Docker。首先需要下载 get-docker.sh 文件，输入命令 curl -fsSL get.docker.com -o get-docker.sh，按 Enter 键，继续输入命令 sh get-docker.sh --mirror Aliyun，以阿里云镜像安装 get-docker.sh 脚本内容，输出如下结果。

```
root@ubuntu:~# curl -fsSL get.docker.com -o get-docker.sh
root@ubuntu:~# sh get-docker.sh --mirror Aliyun
# Executing docker install script, commit: 4957679
+ sh -c apt-get update -qq >/dev/null
+ sh -c apt-get install -y -qq apt-transport-https ca-certificates
curl >/dev/null
+ sh -c curl -fsSL "http://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg"
|apt-key add -qq ->/dev/null
+ sh -c echo "deb [arch=amd64] http://mirrors.aliyun.com/docker-
ce/linux/ubuntu xenial edge" > /etc/apt/sources.list.d/docker.list
+ sh -c apt-get update -qq >/dev/null
+ sh -c apt-get install -y -qq --no-install-recommends docker-
ce >/dev/null
```

3.4.3 配置镜像加速器

输入命令 curl -sSL https://get.daocloud.io/daotools/set_mirror.sh | sh -s http://573b0ee5.m.daocloud.io，该脚本可以将-registry-mirror 加入 Docker 配置文件/etc/default/docker 中，然后输入命令 systemctl restart docker.service，重启 Docker，输出如下结果。

```
root@ubuntu:~# curl -sSL https://get.daocloud.io/daotools/set_mirror.sh |
sh -s http://573b0ee5.m.daocloud.io
docker version >= 1.12
{"registry-mirrors": ["http://573b0ee5.m.daocloud.io"]}
Success.
You need to restart docker to take effect: sudo systemctl restart
docker.service
root@rancher:~ # systemctl restart docker.service
```

重启之后可以输入命令 docker info 检查是否配置成功，如果出现如下字段，说明加速器已经生效。

```
Registry Mirrors:
 http://573b0ee5.m.daocloud.io/
```

以上步骤完成后，输入命令 docker run hello-world，出现如下效果，则表明 Docker 已经安装成功。

```
[root@rancher ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest:
sha256:0e11c388b664df8a27a901dce21eb89f11d8292f7fca1b3e3c4321bf7897bffe
Status: Downloaded newer image for hello-world:latest
Hello from Docker!
This message shows that your installation appears to be working
correctly.
To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs
the executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which
sent it to your terminal.
To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash
Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/
For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

3.4.4 Docker 常用命令

Docker 常用命令如表 3.1 所示。

表 3.1 Docker 常用命令

命　令	说　明
docker version	显示 Docker 版本信息
docker info	显示 Docker 系统信息，包括镜像和容器数
docker search	从 Docker Hub 中搜索符合条件的镜像
docker pull	从 Docker Hub 中拉取或者更新指定镜像
docker login	按步骤输入在 Docker Hub 注册的用户名、密码和邮箱即可完成登录
docker logout	运行后从指定服务器退出，默认为官方服务器
docker images	列出本地所有镜像，对镜像名称按关键词查询
docker ps	列出所有运行中的容器
docker rmi	从本地移除一个或多个指定的镜像
docker rm	从本地移除一个或多个指定的容器
docker history	查看指定镜像的创建历史
docker start|stop|restart	启动、停止和重启一个或多个指定容器
docker kill	“杀死”一个或多个指定容器进程
docker events	从服务器拉取个人动态，可选择时间区间

续表

命　令	说　明
docker top	查看一个正在运行的容器进程，支持 ps 命令参数
docker inspect	检查镜像或者容器的参数，默认返回 JSON 格式
docker pause	暂停某一容器的所有进程
docker unpause	恢复某一容器的所有进程
docker push	将镜像推送至远程仓库，默认为 Docker Hub
docker logs	获取容器运行时的输出日志
docker run	启动一个容器，在其中运行指定命令

3.5　Docker 操作镜像和容器

3.5.1　下载镜像

Docker Hub 上有大量高质量的镜像可以用，这里就怎么获取这些镜像做出相关演示。从 Docker 镜像仓库获取镜像的命令是 docker pull。其命令格式为：docker pull [选项] [Docker Registry 地址[:端口号]/]仓库名[:标签]。这里的仓库名是两段式名称，即<用户名>/<软件名>。

Docker 镜像仓库地址的格式一般是<域名/IP >[:端口号]，默认地址是 Docker Hub 的官方网站。

对于在 Docker Hub 中下载镜像而言，如果不给出用户名，则默认为 library，也就是官方镜像。如下所示，输入命令 docker pull ubuntu：16.04，则：

```
root@ubuntu:/# docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
9ff7e2e5f967: Pull complete
59856638ac9f: Pull complete
6f317d6d954b: Pull complete
a9dde5e2a643: Pull complete
Digest:
sha256:cad5e101ab30bb7f7698b277dd49090f520fe063335643990ce8fbd15ff920ef
Status: Downloaded newer image for ubuntu:16.04
```

上面的命令中没有给出 Docker 镜像仓库地址，因此将会从 Docker Hub 获取镜像。而镜像名称是 ubuntu:16.04，因此将会获取官方镜像 library/ubuntu 仓库中标签为 16.04 的镜像。从下载过程中可以看到，之前提及的分层存储的概念，镜像是由多层存储构成的。下载也是一层层地去下载，并非单一文件。下载过程中给出了每一层的 ID 的前 12 位。并且下载结束后，给出该镜像完整的 sha256 的摘要，以确保下载一致性。在使用上面命令的时候，会发现所看到的层 ID 以及 sha256 的摘要与这里的不一样。这是因为官方镜像是一直在维护的，有任何新的 Bug 或者版本更新，都会进行修复再以原来的标签发布，这样可以确保任何使用这个标签的用户都可以获得更安全、更稳定的镜像。

如果从 Docker Hub 下载镜像时下载速度非常缓慢，可能是因为刚刚配置的镜像加速

器没有配置成功。

3.5.2 运行容器

有了镜像后，就能够以这个镜像为基础启动并运行一个容器。以上面的 ubuntu:16.04 为例，如果打算启动里面的 bash 并且进行交互式操作，可以执行下面的命令：

```
$ docker run -it --rm \
    ubuntu:16.04 \
bash
```

docker run 就是运行容器的命令，如下所示。

```
root@ubuntu:~# docker run -it --rm \
>    ubuntu:16.04 \
>     bash
```

上述命令中，-it 代表的是两个参数，一个是-i，表示交互式操作，另一个是-t，表示终端。由于进入 bash 执行一些命令并且需要查看返回的结果，因此这里需要交互式终端。

--rm：这个参数代表容器退出后随之将其删除。默认情况下，为了排障需求，退出的容器并不会立即被删除，除非手动输入 docker rm 命令。由于这里只是执行命令以看看结果，不需要排障和保留结果，因此使用--rm 可以避免浪费空间。

ubuntu:16.04：指用 ubuntu:16.04 镜像为基础启动容器。

bash：代表放在镜像名后的是命令，这里代表希望有一个交互式 Shell，因此用的是 bash。进入容器后，可以在 Shell 下操作，执行任何所需的命令。

执行 Linux 常用的查看当前系统版本的命令为 cat /etc/os-release，从返回的结果可以看到容器内是 Ubuntu 16.04.5 LTS 系统，返回结果如下。

```
root@ubuntu:~# cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
UBUNTU_CODENAME=xenial
```

最后输入 exit 即可退出容器，该容器也会随着退出而消亡。

3.5.3 删除镜像

要想列出已经下载的镜像，可以使用 docker image ls 或 docker images 命令，如下所示。

```
root@ubuntu:~# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
hello-world         latest              fce289e99eb9        4 months ago
1.84kB
tomcat              latest              1a51cb5e3006        4 months ago
462MB
mysql               5.7.22              6bb891430fb6        10 months ago
372MB
```

上面的 IMAGE ID 是镜像的唯一标识，一个镜像可以对应多个标签。因此，在上面的例子中，可以看到 ubuntu:18.04 和 ubuntu:latest 拥有相同的 ID，因为它们对应的是同一个镜像。如果要删除本地的镜像，可以使用 docker image rmi 命令，其格式为：$ docker image rmi [选项] <镜像 1> [<镜像 2>…]其中，<镜像>可以是镜像 ID、镜像名或者镜像摘要，例如：docker rmi fce289e99eb9。

1. 镜像体积

上面的 SIZE 代表该镜像的体积，ubuntu:18.04 的体积为 86.7MB，ubuntu:16.04 的体积为 117MB，这两个镜像的体积总和为 203.7MB。但是值得注意的是，它们所占的空间实际上并没有这么大，反而远低于 203.7MB，这是因为，通过之前的学习可以得知镜像的结构是分层的，这两个镜像之间如果存在相同的层就会只存在一个，这一个存在的层可以被两个镜像所复用。

2. 虚悬镜像

镜像版本更新时，就会把原先存在的镜像名取消，变为< none >，而最新版本的镜像则会拥有原先旧镜像的名称。镜像重新构建时，会导致名称重复的情况，这时也会把名称给予最新构建的镜像，旧的镜像名称被取消，出现仓库名、标签均为< none >的镜像。因此，当执行 docker pull 操作和 docker build 操作时，极易导致虚悬镜像的出现。一般来说，虚悬镜像已经失去了存在的价值，是可以随意删除的，可以用命令 docker image prune 删除。

3.6 使用 Docker Compose 官方容器编排项目

Compose 项目是 Docker 官方的开源项目，它负责实现对 Docker 容器集群的快速编排。通过前面的介绍，可以了解 Dockerfile 模板文件，让用户很方便地定义一个单独的应用容器。但是，在日常工作中，经常会遇到需要多个容器相互配合完成某项任务的情况。例如，当需要实现一个 Web 项目时，除了 Web 服务容器本身之外，往往还需要再配置与之相关的后端数据库服务容器，甚至包括负载均衡容器等，在这种情况下，使用 Docker Compose 技术，就能够很简单地满足这些需求。

定义和运行多个容器的 Docker 应用程序，在 Compose 中可以使用 YML 文件配置应用服务。然后，只需要一个简单的命令，就可以创建并启动配置的所有服务。

使用 Compose 基本有如下三步流程。

(1) 在 Dockfile 中定义应用环境，使其可以在任何地方复制。

(2) 在 docker-compose.yml 中定义组成应用程序的服务，以便它们可以在隔离的环境

中一起运行。

(3) 运行 dcoker-compose up,Compose 将启动并运行整个应用程序。

Compose 中还有两个重要的概念：服务和项目。

服务(Service)：一个应用的容器,实际上可以包括若干运行相同镜像的容器实例。

项目(Project)：由一组关联的应用容器组成的一个完整业务单元,在 docker-compose.yml 文件中定义。

Compose 的默认管理对象是项目,通过子命令对项目中的一组容器进行便捷的生命周期管理。

Compose 项目用 Python 编写,调用 Docker 服务提供的 API 实现对容器的管理。因此,只要所操作的平台支持 Docker API,就可以在其上利用 Compose 进行编排管理。

1. Docker Compose 的安装

Compose 可以通过 Python 的包管理工具 pip 进行安装,也可以直接下载编译好的二进制文件使用,甚至能够直接在 Docker 容器中运行。前两种方式是传统方式,适合在本地环境下安装使用；最后一种方式则不破坏系统环境,更适合云计算场景。

Docker for Mac、Docker for Windows 自带 docker-compose 二进制文件,安装 Docker 之后可以直接使用。在 Linux 上的也安装十分简单,从官方 GitHub Release 处直接下载编译好的二进制文件即可。例如,在 Linux 64 位系统上直接下载对应的二进制包,输入以下命令：

curl -L https://github.com/docker/compose/releases/download/1.22.0/docker-compose-'uname -s'-'uname -m' > /usr/local/bin/docker-compose。

上述命令是指下载 1.22.0 版本的 docker-compose 到/usr/local/bin/目录下,并命名为 docker-compose,返回结果如下。

```
root@ubuntu:/etc/docker# curl -L
https://github.com/docker/compose/releases/download/1.22.0/docker-
compose-
'uname -s'-'uname -m' > /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total  Spent   Left    Speed
100   617    0   617    0     0     77     0 --:--:-- 0:00:08 --:--:--   162
100 11.2M  100 11.2M    0     0  34384       0  0:05:41 0:05:41 --:--:--
148k
root@ubuntu:/etc/docker#
```

下载完成后进入/usr/local/bin/目录,可见 root 用户对文件的权限为可读、可写,其他用户只有读权限,如下所示。

```
root@ubuntu:~# cd /usr/local/bin/
root@ubuntu:/usr/local/bin# ll
total 11484
drwxr-xr-x  2 root root     4096 Jan 27 13:42 ./
drwxr-xr-x 12 root root     4096 Jan 27 15:31 ../
```

```
6  -rw-r--r-- 1 root root 11750136 Jan 27 13:54 docker-compose
7  root@ubuntu:/usr/local/bin#
```

由于 docker-compose 文件需要被执行，因此，需要为该文件添加执行权限，命令为 chmod +x docker-compose，如下所示。

```
1  root@ubuntu:/usr/local/bin# chmod +x docker-compose
2  root@ubuntu:/usr/local/bin# ll
3  total 11484
4  drwxr-xr-x  2 root root     4096 Jan 27 13:42 ./
5  drwxr-xr-x 12 root root     4096 Jan 27 15:31 ../
6  -rwxr-xr-x  1 root root 11750136 Jan 27 13:54 docker-compose*
7  root@ubuntu:/usr/local/bin#
```

此时输入查询版本命令 docker-compose version，如果可以打印出如下所示版本号，即为安装成功。

```
1  root@ubuntu:/usr/local/bin# docker-compose version
2  docker-compose version 1.22.0, build f46880fe
3  docker-py version: 3.4.1
4  CPython version: 3.6.6
5  OpenSSL version: OpenSSL 1.1.0f  25 May 2017
6  root@ubuntu:/usr/local/bin#
```

安装成功后，在/usr/local/目录下新建 docker/tomcat 文件夹，放置配置文件 docker-compose.yml，如下所示。

```
1  root@ubuntu:/usr/local# cd /usr/local/
2  root@ubuntu:/usr/local# ll
3  total 48
4  drwxr-xr-x 12 root root 4096 Jan 27 15:31 ./
5  drwxr-xr-x 10 root root 4096 Jan  2 18:01 ../
6  drwxr-xr-x  2 root root 4096 Jan 27 13:42 bin/
7   rwxr-xr-x  2 root root 4096 Apr 21  2016 etc/
8  drwxr-xr-x  2 root root 4096 Apr 21  2016 games/
9  drwxr-xr-x  2 root root 4096 Apr 21  2016 include/
10 drwxr-xr-x  3 root root 4096 Jan  2 18:01 lib/
11 lrwxrwxrwx  1 root root    9 Jan  2 18:01 man -> share/man/
12 drwxr-xr-x  2 root root 4096 Apr 21  2016 sbin/
13 drwxr-xr-x  6 root root 4096 Jan  2 18:08 share/
14 drwxr-xr-x  2 root root 4096 Apr 21  2016 src/
15 drwxr-xr-x  2 root root 4096 Jan 27 15:40 tomcat/
16 root@ubuntu:/usr/local# mkdir docker
17 root@ubuntu:/usr/local# ll
18 total 48
19 drwxr-xr-x 12 root root 4096 Jan 27 15:31 ./
20 drwxr-xr-x 10 root root 4096 Jan  2 18:01 ../
21 drwxr-xr-x  2 root root 4096 Jan 27 13:42 bin/
```

```
22 drwxr-xr-x  3 root root 4096 Jan 27 14:53 docker/
23 drwxr-xr-x  2 root root 4096 Apr 21  2016 etc/
24 drwxr-xr-x  2 root root 4096 Apr 21  2016 games/
25 drwxr-xr-x  2 root root 4096 Apr 21  2016 include/
26 drwxr-xr-x  3 root root 4096 Jan  2 18:01 lib/
27 lrwxrwxrwx  1 root root    9 Jan  2 18:01 man -> share/man/
28 drwxr-xr-x  2 root root 4096 Apr 21  2016 sbin/
29 drwxr-xr-x  6 root root 4096 Jan  2 18:08 share/
30 drwxr-xr-x  2 root root 4096 Apr 21  2016 src/
31 root@ubuntu:/usr/local# cd docker
32 root@ubuntu:/usr/local/docker# mkdir tomcat
33 root@ubuntu:/usr/local/docker# ll
34 total 16
35 drwxr-xr-x  4 root root 4096 May 27 15:10 ./
36 drwxr-xr-x 12 root root 4096 Jan 27 15:31 ../
37 drwxr-xr-x  2 root root 4096 May 27 15:10 tomcat/
38 root@ubuntu:/usr/local/docker# cd tomcat/
39 root@ubuntu:/usr/local/docker/tomcat# vi docker-compose.yml
```

模板文件是使用 Compose 的核心，涉及的指令关键字也比较多。输入命令 vi docker-compose.yml 编辑该文件，添加 Tomcat 配置，通过使用 docker-compose 完成开机自启 Tomcat 服务，如下所示。

```
1  version: '3'
2  services:
3    tomcat:
4      restart: always
5      image: tomcat
6      container_name: tomcat
7      ports:
8        - 8080:8080
```

上述 docker-compose.yml 文件中的内容，格式要求严格，不允许出现制表符(不能使用 Tab 键)，只允许有空格。

第一行 version 代表 docker-compose 的版本，冒号后有一个空格，然后是单引号，引号里面填写自己安装的版本号，然后按 Enter 键。

第二行 services 代表服务，目前这个容器中只配置了一个 Tomcat 服务。

YML 配置文件里面有着严格的层级关系，文件中的第一行和第二行是一个等级，接下来配置的服务信息又属于一个等级。

第三行之前要有空格，为了与之前的等级加以区分，tomcat 为服务的名称，可以自定义。

第四行为 Tomcat 服务做相关配置，因此，之前还要有空格与前两级加以区分。restart 后跟 always，代表总是重新启动，也就是添加了服务开机重启的功能。

第五行为要添加的镜像为 tomcat。

第六行代表为该容器起个名字。

第七行代表将要配置映射端口的信息。

第八行为 Tomcat 服务的端口号，冒号左边是宿主机端口，右边为容器的端口-8080:8080。

编辑完成后保存、退出，运行 docker-compose up 命令；如果该虚拟机上未下载 Tomcat 镜像，则启动后会自行先下载 Tomcat 镜像然后自动启动 Tomcat 服务，如下所示。

```
1  root@ubuntu:/usr/local/tomcat# docker-compose up
2  Creating tomcat ... done
3  Attaching to tomcat
4  tomcat     | 27-May-2019 08:10:10.681 INFO [main]
5  org.apache.catalina.startup.VersionLoggerListener.log Server version:
6  Apache Tomcat/8.5.37
7  //……由于篇幅有限，此处打印日志内容省略
8  tomcat     | 27-May-2019 08:10:19.534 INFO [main]
9  org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["ajp-
10 nio-8009"]
11 tomcat     | 27-May-2019 08:10:19.582 INFO [main]
12 org.apache.catalina.startup.Catalina.start Server startup in 6939 ms
```

服务启动成功后，新开一个连接，可以发现 Tomcat 镜像被启动了。如果想删除该容器，可以输入命令 docker-compose down。在浏览器中输入服务器地址、8080 端口，即可访问到 Tomcat 默认主页面，如图 3.13 所示。

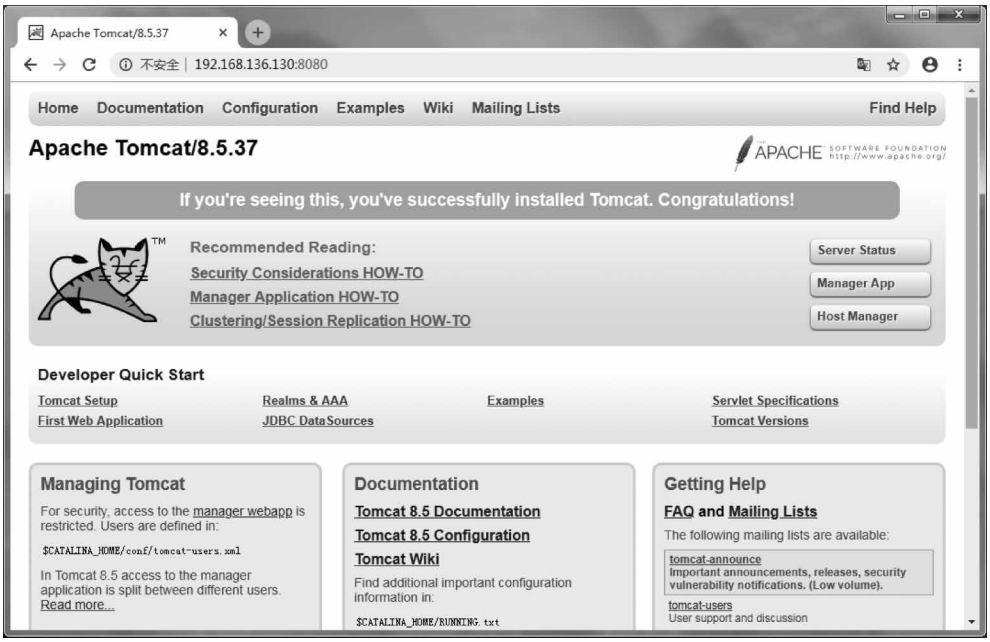

图 3.13　访问服务

2. 卸载

如果是通过二进制包方式安装的，删除二进制文件即可。删除命令为 $ sudo rm /usr/

local/bin/docker-compose；如果是通过 pip 安装的，则执行删除命令 $ sudo pip uninstall docker-compose。

3. Docker Compose 命令对象与格式

对于 Compose 来说，大部分命令的对象既可以是项目本身，也可以指定为项目中的服务或者容器。如果没有特别的说明，命令对象是项目，这意味着项目中所有的服务都会受到命令影响。

执行 docker-compose [COMMAND] --help 或者 docker-compose help [COMMAND] 可以查看具体某个命令的使用格式。

docker-compose 命令的基本使用格式是：

```
docker-compose [-f=<arg>...] [options] [COMMAND] [ARGS...]
```

-f，--file FILE 指定使用的 Compose 模板文件，默认为 docker-compose.yml，可以多次指定。

-p，--project-name NAME 指定项目名称，默认将使用所在目录名称作为项目名。

--x-networking 使用 Docker 的可拔插网络后端特性。

--x-network-driver DRIVER 指定网络后端的驱动，默认为 bridge。

--verbose 输出更多调试信息。

-v，--version 打印版本并退出。

4. docker-compose 常用命令

docker-compose 常用命令如表 3.2 所示。

表 3.2　docker-compose 常用命令

命　令	说　明
docker-compose up －d	构建启动容器
docker-compose exec xxx bash	登录到 xxx 容器中
docker-compose down	删除所有当前运行的容器
docker-compose ps	显示所有容器
docker-compose restart	按步骤输入在 Docker Hub 注册的用户名、密码和邮箱即可完成登录
docker logout	重新启动容器
docker-compose build	构建镜像
docker-compose build	用来创建或重新创建服务使用的镜像
docker-compose kill	用于通过容器发送 SIGKILL 信号强行停止服务
docker-compose logs	显示服务的日志信息
docker-compose pause/unpause	暂停服务/恢复被暂停的服务
docker-compose port	查看服务中的端口与物理机的映射关系
docker-compose pull	拉取服务依赖的镜像
docker-compose restart	重启某个服务中的所有容器
docker-compose rm	删除停止的服务(服务里的容器)
docker-compose run	在服务中运行一个一次性的命令
docker-compose start/stop	启动运行/停止运行某个服务的所有容器

3.7 Docker Compose 快速部署 Tomcat & MySQL

Docker Compose 部署 Tomcat 服务在 3.6 节中已经做了详细介绍，Docker Compose 部署 MySQL 的步骤与之类似，只需要把 docker-compose.yml 文件中的 Tomcat 服务换成 MySQL。需要注意的是，在书写配置文件时 MySQL 的版本信息，版本不同，配置文件也会有所改变。下面将列出不同版本的配置文件。

1. MySQL 5

```
version: '3.1'
services:
  mysql:
    restart: always
    image: mysql:5.7.22
    container_name: mysql
    ports:
      - 3306:3306
    environment:
      TZ: Asia/Shanghai
      MYSQL_ROOT_PASSWORD: 123456
    command:
      --character-set-server=utf8mb4
      --collation-server=utf8mb4_general_ci
      --explicit_defaults_for_timestamp=true
      --lower_case_table_names=1
      --max_allowed_packet=128M
      --sql-
mode="STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION,NO_
ZERO_DATE,NO_ZERO_IN_DATE,ERROR_FOR_DIVISION_BY_ZERO"
    volumes:
      - mysql-data:/var/lib/mysql
volumes:
  mysql-data:
```

2. MySQL 8

```
version: '3.1'
services:
  db:
    image: mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: 123456
    command:
      --default-authentication-plugin=mysql_native_password
      --character-set-server=utf8mb4
      --collation-server=utf8mb4_general_ci
      --explicit_defaults_for_timestamp=true
      --lower_case_table_names=1
    ports:
```

```
      - 3306:3306
    volumes:
      - ./data:/var/lib/mysql
  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
```

3.8 快速部署应用到容器

在本节中,将详细讲解通过 docker-compose 部署整个应用程序。该程序还是以一个网上商城的项目——锋迷网为例,项目名为 chapter16。首先,在/usr/local/docker 目录下创建一个新文件夹 mkdir chapter16,然后编辑 docker-compose.yml 文件,输入命令 vi docker-compose.yml,编辑内容如下。

```
version: '3'
services:
  web:
    restart: always
    image: tomcat
    container_name: web
    ports:
      - 8080:8080
    volumes:
      - /usr/local/docker/chapter16:/usr/local/tomcat/webapps
  mysql:
    restart: always
    image: mysql:5.7.22
    container_name: mysql
    ports:
      - 3306:3306
    environment:
      TZ: Asia/Shanghai
      MYSQL_ROOT_PASSWORD: 123456
    command:
      --character-set-server=utf8mb4
      --collation-server=utf8mb4_general_ci
      --explicit_defaults_for_timestamp=true
      --lower_case_table_names=1
      --max_allowed_packet=128M
      --sql-
mode="STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION,NO_
ZERO_DATE,NO_ZERO_IN_DATE,ERROR_FOR_DIVISION_BY_ZERO"
    volumes:
      - mysql-data:/var/lib/mysql
volumes:
  mysql-data:
```

由上述配置文件信息可以看出，在 services 中配置了两个服务：Web（Tomcat）和 MySQL。每个服务的具体配置信息在之前的章节中已经给出，此处不再详解。需要注意的是，在 Web 服务中新增了 volumes（数据卷）的信息。冒号左边为宿主机目录/usr/local/docker/chapter，右边为容器目录/usr/local/tomcat/webapps。由于将要上传的项目是一个 war 包，所以需要放置 Tomcat 服务的 webapps 目录下，在重启 Tomcat 服务时，即可自动解压完成项目的部署，编辑完成后保存退出。上传的 war 包应传至宿主机中/usr/local/docker/chapter16 目录下，在启动容器之前先查看有没有启动的服务容器，如果有则关闭，确保接下来部署项目时端口不被占用。最后，在/usr/local/docker/chapter16 目录下输入 docker-compose up 命令，即可完成包括构建镜像、（重新）创建服务、启动服务，并关联服务相关容器的一系列操作，可见终端打印如下信息。

```
root@ubuntu:/usr/local/docker/chapter16# docker-compose up
Creating network "chapter16_default" with the default driver
Creating web   ... done
Creating mysql ... done
Attaching to mysql, web
mysql     | 2019-05-27T07:23:31.842104Z 0 [Note] mysqld (mysqld 5.7.22)
starting as process 1 ...
//……由于篇幅有限，此处打印日志内容省略
web       | 27-May-2019 07:24:04.201 INFO [main]
org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["ajp-
nio-8009"]
web       | 27-May-2019 07:24:04.433 INFO [main]
org.apache.catalina.startup.Catalina.start Server startup in 19273 ms
```

通过浏览器访问 192.168.136.133:8080/chapter16（输入自己服务的地址）即可发现项目部署成功，如图 3.14 所示。

图 3.14　访问项目

3.9 本章小结

通过本章学习，了解 Docker 与传统虚拟化方式的区别之处，能够使用 Docker 构建镜像、运行容器等，熟悉常用 Docker 命令完成基本操作。在 docker-compose 的学习中，应当掌握使用容器编排项目、快速部署项目到容器，真正理解“一次构建，到处运行”给项目部署带来的便利之处。

3.10 习　　题

1. 填空题

(1) _____是一个开源的应用容器引擎，可以实现虚拟化。

(2) Docker 是使用_____架构模式，通过远程 API 管理和创建容器的。

(3) 使用_____后，容器无论是被删除还是被重新运行，数据都不会丢失。

(4) 输入命令_____，可以更新软件包列表。

(5) _____是运行容器的命令。

2. 选择题

(1) 下列关于 Docker 容器化引擎说法中错误的是(　　)。

A. 简单来说，Docker 就是使用 B/S 架构模式，通过远程 API 管理和创建容器的

B. Docker 容器化引擎也是一种服务器，这种服务器可以长时间地运行，可以通过 REST API 和守护进程完成通信

C. 客户端命令将请求通过命令行工具调用 Docker 提供的 REST API，将命令发送给守护进程

D. 通过 API 可以对镜像(Image)、容器(Container)、网络(Network)、数据卷(Data Volumes)进行管理，守护进程最后会把命令的执行结果返回给客户端

(2) 下列选项中，对 Docker 常用命令描述不准确的是(　　)。

A. docker version 显示 Docker 版本信息

B. docker pull 从 Docker Hub 中拉取或者更新指定镜像

C. docker images 表示下载 Docker 镜像

D. logout 表示注销

(3) 下列选项中，说法不正确的是(　　)。

A. 从 Docker 镜像仓库获取镜像的命令是 docker images

B. docker run 命令用来运行 Docker 容器

C. 要想列出已经下载下来的镜像，可以使用 docker image ls 或 docker images 命令

D. 运行 docker-compose up，Compose 将启动并运行整个应用程序

(4) 以下(　　)命令可以“杀死”一个或多个指定容器进程。

A. docker kill

B. docker search

C. docker logout

D. docker rmi

(5) 下列选项中,说法错误的是(　　)。

A. docker run -it -rm 命令表示交互式运行容器

B. 在 Compose 中可以使用 YML 文件配置应用服务

C. docker rm 命令用来从本地移除一个或多个指定的镜像

D. 镜像(Image)和容器(Container)的关系,就像是面向对象编程思想中的类和实例一样,容器是镜像运行时的实体

3. 思考题

使用 Docker Compose 官方容器怎样编排项目?