# 第 3 章

# 容器技术的流行

在货物运输历史上，安全快捷地交付货物一直是个难题。集装箱是个伟大的发明，如果没有集装箱，当把不同类型的货物堆在一起进行运输的时候，质地软的货物就容易被压坏。集装箱的出现还使货物可以更快地完成装卸。如同集装箱对货物运输的变革，容器技术的出现带来了软件交付的巨大变革。

## 3.1 容器的优势

在软件开发实践中，通常希望软件可以运行在和宿主机隔离的环境中，像虚拟机一样，容器技术可以使进程运行在一个隔离的环境中，但是虚拟机的隔离机制是通过在宿主机上启动另一个完整的操作系统实现的，而容器技术只是在宿主机的操作系统之上借助内核接口做了资源视图的隔离，并没有启动另一套操作系统，所以更轻量。这个优势体现在启动和销毁的速度上，容器的启动时间一般只有数秒，而虚拟机的启动时间通常为数分钟。人类对速度的追求从未止步，从蜗行牛步的畜力车到日行千里的动车再到超声速的飞机，从鸿雁飞书到电报、电话、短信再到现在各种实时通信软件，都在印证人类对速度提升的追求。快速启动也是让容器技术广为流行的一个重要原因。

限制进程使用计算资源是容器技术的另一个重要功能。在资源受限的情况下，如果某个进程使用了过多的内存或者占用了过多的 CPU 时间，则会严重影响在同一台机器上的其他进程的正常工作，系统可能因此而崩溃。产生这种情况的原因可能是进程运行的代码有严重的缺陷，也可能是遭受了某种恶意攻击。进程使用的计算资源超过预期会引起系统不稳定。这时就可以利用容器技术来解决这个问题。容器技术提供了细粒度的资源约束能力，通常包括约束进程占用的 CPU 时间，以及使用的内存大小等。

## 3.2 Docker 简介

Docker 是容器技术的代表，在 2013 年开源，随着 Docker 的迅速普及，它成为容器技术的标准。

2020年知名技术网站StackOverflow举办的问卷调查的结果表明，Docker在“最喜欢的平台”这一项中的排名仅次于Linux。在“未来最想使用的平台”这一项的排名位列第一。这次问卷调查有接近65000名开发者参与，取得这样高的排名可见Docker的受欢迎程度之高。

Docker深刻改变了软件的交付方式，在主流的开源平台上比较受欢迎的软件项目几乎提供了Docker运行的方式。在使用这类提供Docker运行方式的项目时，不需要准备软件所需的环境，只需通过运行开发者提供的镜像就可以把程序运行起来，十分方便。这样的交付方式显著降低了用户使用软件前的学习成本，节省了用户准备软件运行环境的时间成本，Docker帮助开发者实现了一次构建，随处运行的高质量交付。Docker革命性地解决了开发者交付和分享软件的难题。采用Docker运行软件的用户从此不会再遇到环境问题引起的无法运行程序的尴尬。

在各大开源代码托管平台可以看到越来越多的开发者在发布他们的软件项目时，会同时提供运行软件的Docker容器镜像，意味着开发者不仅交付了代码，并且交付了最佳的运行代码的环境。把镜像定义文件加入软件版本控制系统也逐渐成为一种最佳实践。

Docker容器具有高度的可移植性，为容器技术创造了工业标准，Docker屏蔽了基础设施的差异，让应用不管运行在Windows还是Linux平台都能表现出一致的行为。

现在Docker拥有世界上最大的容器镜像仓库Docker Hub，作为Docker默认的镜像仓库，目前它保存的容器镜像已经基本包含了所有常见的应用程序对应的镜像。可以在Docker Hub中查看各个镜像的使用文档。在注册账号以后，就可以把自己构建的镜像上传到Docker Hub来和世界各地的用户分享。

大量的软件公司已经把Docker融入日常工作流程中，用Docker容器作为软件项目的开发环境，用Docker容器运行自动化测试和持续集成。

另外，得益于Docker容器对计算资源的细粒度约束能力，Docker可以帮助软件公司更加集约地使用计算资源，从而有效降低成本，产生良好的经济效益。

Docker提供了非常完善的CLI工具，可以让用户轻松地和Docker交互，这个工具让Docker的易用性大大增加，并且Docker还提供了RESTful API，让用户可以远程管理Docker。Docker友好的交互接口大幅降低了使用容器的技术门槛。

成功没有偶然，Docker有着丰富详尽的文档，简洁易用的交互工具，有活跃的项目更新升级，稳定且优异的性能，Docker镜像有着极佳的移植性和便携性。Docker提供了诸多容器交付的方式，可以上传到镜像仓库，也可以打包成压缩文件。这些都是Docker获得成功的重要因素。

有些工程师接触Docker技术时，对容器技术并没有充分理解，只是把它当作虚拟机来用。虚拟机配置烦琐，启动时间相对漫长，而Docker往往只需一条简单的命令就可以运行需要的程序。例如只用下面的命令就可以启动一个MySQL服务，代码如下：

```
docker run -e MYSQL_ROOT_PASSWORD=123 mysql
```

Docker的设计使它很容易上手，上面这条命令使Docker自动下载最新的MySQL镜像，然后启动这个镜像，用户用一条命令就可以把MySQL数据库运行起来。这甚至比在宿主机上通过下载并安装MySQL安装包，然后运行还要方便。当有更复杂的应用场景时，这种优势会更加明显。举个例子，如果需要同时启动5.6、5.7、8.0这3个版本的MySQL数据库，在宿主机上则需要先找到这3个版本的MySQL并安装，然后指定不同的端口参数来分别启动，不然会因端冲突而无法同时启动，并且同时安装3个版本的MySQL可能因为安装位置冲突而引起文件覆盖等诸多问题，而使用Docker只需下面的三条命令：

```
docker run -e MYSQL_ROOT_PASSWORD=123 mysql:5.6
docker run -e MYSQL_ROOT_PASSWORD=123 mysql:5.7
docker run -e MYSQL_ROOT_PASSWORD=123 mysql:8.0
```

只要指定对应的版本号作为镜像的标签，Docker就会启动对应版本的MySQL容器。

相信Docker的绝大部分初学者在接触之后都会对它的简洁和易用性大加赞赏。作为容器时代的操作系统，Docker正在并将继续为软件的高效交付赋能。

Docker是由Go语言编写的，通过docker version命令可以查看Docker的版本信息，包括开发时使用的Go语言版本等。

查看命令如下：

```
docker version
```

会看到类似如下的输出信息：

```
Client: Docker Engine - Community
 Cloud integration: 1.0.7
 Version:           20.10.2
 API version:       1.41
 Go version:        go1.13.15
 Git commit:        2291f61
 Built:             Mon Dec 28 16:12:42 2020
 OS/Arch:           darwin/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.2
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       8891c58
  Built:            Mon Dec 28 16:15:28 2020
  OS/Arch:          Linux/amd64
```

```
 Experimental:     false
containerd:
 Version:          1.4.3
 GitCommit:        269548fa27e0089a8b8278fc4fc781d7f65a939b
runc:
 Version:          1.0.0-rc92
 GitCommit:        ff819c7e9184c13b7c2607fe6c30ae19403a7aff
docker-init:
 Version:          0.19.0
 GitCommit:        de40ad0
```

可以看到输出信息包括两大部分，分别是 Client 和 Server，也就是客户端和服务器端。这是因为 Docker 采用了常见的客户端/服务器端的软件架构。版本信息值得引起重视，如果 Docker 客户端版本高于服务器端版本，则客户端的有些命令就无法运行成功。这是因为服务器端的版本较低，所以对某些 API 还未支持。

服务器端指 Docker Engine，表现为一个叫 dockerd 的进程。它提供了容器运行和管理的核心功能，还提供了 API，使用户可以通过 API 来和 Docker 交互，称为 Docker Engine API。

客户端指 Docker CLI 命令行工具，常见的 docker run、docker exec 等命令就是客户端的工具。

## 3.3 Docker 安装

在个人计算机上通过安装 Docker Desktop 应用程序来使用 Docker。不同的平台需要安装不同的版本。Windows 平台安装的是 Docker Desktop for Windows 版本，Mac 平台安装的是 Docker Desktop for Mac 版本。Docker Desktop 安装过程比较简单，只需下载相应的安装包，然后运行安装包并按提示完成操作。

Docker Desktop 内置了 Kubernetes 等许多组件，并且提供了友好的 UI 界面，包括简洁的仪表盘等，方便可视化地对容器进行管理。

在 Linux 系统安装 Docker，建议使用流行的包管理工具来安装，也可以直接安装 Docker 二进制文件，二进制文件的下载网址为 https://download.docker.com。

下面介绍在 Ubuntu 上使用包管理工具 apt-get 安装 Docker 的步骤。

老版本的 Docker 在 apt 仓库中的名字是 docker、docker.io 或者 docker-engine，如果已经安装了这些包，则在安装最新版本的 Docker 前，需要先删除它们，命令如下：

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

首先更新 apt 仓库，命令如下：

```
sudo apt-get update
```

接着安装常用的依赖包，命令如下：

```
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

现在添加 Docker 官方的 GPG key，命令如下：

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/
share/keyrings/docker-archive-keyring.gpg
```

完成以上操作后，开始安装 Docker Engine，命令如下：

```
sudo apt-get update \
      && apt-get install docker-ce docker-ce-cli containerd.io
```

到这里已经完成了 Docker 安装，需要进一步配置 Docker，实现开机自动启动。

大部分的 Linux 发行版使用 systemd 来管理系统服务，下面把 Docker 设置为开机自动启动，命令如下：

```
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
```

最后通过运行 docker version 命令来验证 Docker 是否可以运行。

如果希望以非 root 用户的身份运行 Docker 命令，可以把当前用户加入 docker 组中，命令如下：

```
sudo usermod -aG docker $USER
```

需要注意的是加入 docker 组中的用户具有较高的权限等级，可以用来运行获取宿主机 root 权限的容器，存在一定的安全风险。

熟悉 Ansible 的用户可以参考下面的 ubuntu.yaml 文件，通过命令 ansible-playbook 来自动完成在 Ubuntu 系统上 Docker 的安装。文件内容如下：

```
---
- name: install docker
  hosts: dockers
```

```
  remote_user: Ubuntu

  tasks:
  - name: Run "apt-get update"
    apt:
      update_cache: yes
    become: yes

  - name: install tools
    apt:
      pkg:
        - apt-transport-https
        - ca-certificates
        - curl
        - gnupg
    become: yes

  - name: Add Docker's official GPG key
    ansible.builtin.shell: curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
    args:
      executable: /bin/bash

  - name: Set up repo
    ansible.builtin.shell: echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
| sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
    args:
      executable: /bin/bash
    become: yes

  - name: Run "apt-get update"
    apt:
      update_cache: yes
    become: yes

  - name: ensure docker is at the latest version
    apt:
      pkg:
        - docker-ce
        - docker-ce-cli
        - containerd.io
    become: yes

  - name: ensure docker daemon can autostart
    ansible.builtin.command: systemctl enable docker
```

```
    become: yes

  - name: ensure docker daemon is running
    ansible.builtin.command: systemctl start docker
    become: yes

  - name: ensure user is in docker group
    ansible.builtin.command: usermod -aG docker {{ ansible_ssh_user }}
    become: yes
```

## 3.4　Docker 在开发领域的价值

Docker 很好地解决了在第 2 章中提到的传统软件生产方式下开发环境带来的问题。在构建完一个可以让软件正常运行的 Docker 容器镜像之后，软件就可以在任何支持 Docker 运行的环境下正常运行了。

如果很不幸在开发的过程中，计算机坏了，需要换另一台计算机继续开发，这时 Docker 容器镜像的优势就体现出来了，可以轻松地把软件所需的开发环境通过 Docker 容器镜像恢复回来，避免了从头开始搭建开发环境的麻烦。对于有些复杂项目而言，开发环境的搭建可能需要几天时间，所以把开发环境构建到一个容器镜像中是一个明智的选择，容器镜像可以把搭建过程中付出的劳动保存下来。

在开发团队多人协作的场景下，Docker 的价值更加凸显，公司都会有正常的人员流动，当新成员加入团队的时候，团队希望新成员能够尽快熟悉软件项目，尽快开始贡献代码。如果团队已经把开发环境构建成了容器镜像，新成员就可以把搭建环境的时间节省下来，通过运行和调试代码来熟悉项目。这个看似不起眼的技术可以提高新成员在磨合期的幸福感，减小熟悉项目的压力。

当一个软件项目开发团队中的成员对开发环境做出变更后，可以及时地发布新的开发环境的容器镜像，这样团队中的其他成员就可以第一时间无感地获取一个统一的变更后的开发环境。这大大节省了为了同步开发环境所付出的协作成本，提高了整个团队的工作效率，也减少了因为环境问题而导致的错误。

Docker 的使用也节省了大量的软件部署时间成本，当新版本的软件发布以后，可以使用镜像快速完成部署，如果新版本的发布不幸存在 Bug，需要紧急回滚，也可以使用上一版本的镜像快速完成回滚，避免代码回滚而环境停留在最新的版本的情况。

为了更好地协作或者为了更高的安全性，有的企业需要由一台企业内网的服务器提供公共的开发环境。这种情况下，开发人员共用一个开发环境会有互相干扰的潜在问题，例如网络端口冲突，文件被覆盖及资源占用不均等。可以通过为每个开发人员分配一个 Docker 容器作为开发环境来解决。

在容器提供的隔离环境下，每个人在自己的容器中看到的进程空间都与自己相关，别人

在开发环境启动的某个进程不会被自己看到，这样降低了操作时的心智负担。在容器中误杀的进程也只会影响到自己的开发环境，其他人不会受牵连。通过限制容器的计算机资源的使用可以避免资源被独占。这些都提高了开发团队的工作效率，带来了更好的开发体验。团队成员离职后，只需中止分配的对应容器就完成了回收开发环境的工作。

在软件开发实践中，通常会引入 CI/CD，也就是采用持续集成和持续交付的方法来加速软件的开发和交付流程。在持续集成和持续交付领域，需要用脚本实现一定程度的自动化。持续集成一般会在每次代码发布后自动触发。因为代码的发布比较频繁，所以提供持续集成的服务需要部署到不间接运行的服务器上，这样就需要在服务器搭建一个和开发环境保持同步的环境。Docker 可以用来保证环境的一致性，从而很好地支持自动化脚本来完成软件集成和交付工作。

## 3.5 Docker 在测试领域的价值

软件测试往往需要在多个目标平台运行软件项目来发现软件的兼容性问题，使用 Docker 可以方便地模拟目标平台。不必为了运行一个历史版本的软件而卸载及重装，或者重新购置服务器，可以用 Docker 模拟一个目标平台，这将会节省很多时间和 IT 成本。

例如某个软件在开发时使用了旧版本 5.0 的 Redis 数据库，后期想升级数据库，当使用新版本 6.0 的 Redis 数据库时，需要安装 6.0 版本的 Redis 并测试，来检查软件的功能是否和之前表现一致。如果使用传统方式可能需要卸载旧版本的 Redis，然后安装新版本的 Redis，而利用 Docker 容器，只需启动新版本的 Redis 镜像，就可以开始测试了。

使用 Docker 容器作为测试环境之后，测试环境和开发环境可以轻松地实现同步。测试人员只需下载最新的镜像就可完成环境的同步，不用关心环境发生了什么变更或如何进行同步。实践中经常会发生因为环境变更而导致之前运行正常的测试用例运行失败的情况。

对于需要运行在远端服务器的自动化测试场景而言，容器化的测试环境可以方便地使用脚本来管理起来，对于提高测试自动化程度有着重要作用。