Microsoft

Modern App Deployment with Microsoft Azure Kubernetes

Professional



Dawid Borycki

Foreword by Scott Hanselman
VP Developer Community, Microsoft

FREE SAMPLE CHAPTER

Microsoft

Modern App Deployment with Azure Kubernetes

Dawid Borycki

Modern App Deployment with Azure Kubernetes

Published with the authorization of Microsoft Corporation by:
Pearson Education, Inc.

Copyright © 2026 by Pearson Education, Inc.

Hoboken, New Jersey

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit *www.pearson.com/global-permission-granting.html*.

No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein. Please contact us with concerns about any potential bias at *www.pearson.com/report-bias.html*.

ISBN-13: 978-0-13-544250-0

ISBN-10: 0-13-544250-8

Library of Congress Control Number: 2025941170

$PrintCode

Trademarks

Microsoft and the trademarks listed at http://www.microsoft.com on the "Trademarks" webpage are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author, the publisher, and Microsoft Corporation shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the programs accompanying it.

Editor-in-Chief
Julie Phifer

Executive Editor
Loretta Yates

Associate Editor
Shourav Bose

Development Editor
Songlin Qiu

Technical Editor
Charles Pluta

Managing Editor
Sandra Schroeder

Senior Project Editor
Tracey Croom

Copy Editor
Linda Laflamme

Indexer
Ken Johnson

Proofreader
Barbara Mack

Cover Designer
Twist Creative, Seattle

Compositor
codeMantra

Cover Illustration
Yakov24_Shutterstock

Contents at a Glance

Introduction *xix*

PART I CONTAINERIZATION

CHAPTER 1 Motivation 3
CHAPTER 2 Application 13
CHAPTER 3 Containerization with Docker 21
CHAPTER 4 How to Containerize an Application with Docker 37
CHAPTER 5 Azure Container Registry 55

PART II CONTAINER ORCHESTRATION

CHAPTER 6 Container Orchestration with Kubernetes 71
CHAPTER 7 Azure Kubernetes Service 83
CHAPTER 8 Azure SQL Database and Azure Key Vault 97
CHAPTER 9 Diagnosing and Monitoring the Cluster 115
CHAPTER 10 Application Rollouts and Rollbacks 127

PART III AUTOMATION

CHAPTER 11 Automating Image Builds and Deployment Using CI/CD Pipelines 141
CHAPTER 12 Load Testing, Horizontal Pod Autoscaling, and Cluster Autoscaling 161
CHAPTER 13 Deploying and Managing Kubernetes Applications with Helm 181
CHAPTER 14 Infrastructure as Code 205

Index *237*

This page intentionally left blank

Contents

Introduction . . . . . *xix*

PART I CONTAINERIZATION

Chapter 1 Motivation 3

Modern application development and deployment 4

The advantage of disk images 6

The advantages of containers for application deployment 6

Tools 8

- Git 8
- .NET 8 SDK 8
- Docker Desktop 8
- Visual Studio Code 9
- Azure Subscription 9
- Azure Command Line Interface 10
- Notation 10

Summary 11

Chapter 2 Application 13

Running the application 14

Database connection 14

Web API 15

- Retrieving data using the web browser 16
- Swagger 16

Summary 19

Chapter 3 Containerization with Docker 21

Why use containers? 21
Imperative vs. declarative configuration 21
What is containerization, and how can it help you? 22
Containers vs. virtual machines 23
What is Docker? 25
Architecture 25
How to pull and run a container 26
How to manage containers 29
How to remove a container 30
Running multiple containers 31
Volumes 32
Networking 34
Docker Compose 35
Summary 36

Chapter 4 How to Containerize an Application with Docker 37

Dockerfile 37
Containerize the People.WebApp 38
Building the container image 40
Running a containerized People.WebApp 42
Modifying application settings 43
Using Docker Compose to provision a multi-container solution 46
Creating a configuration file 47
Adding Microsoft SQL Server 48
Running an app 50
Testing a solution 51
Cleaning up 53
Summary 54

Chapter 5 Azure Container Registry 55

What is Azure Container Registry? 56
How do you create the Container Registry in Azure? 56

Azure Portal 56
Azure CLI 59
Azure Cloud Shell 64
Clean up 65
How do you push a local image to the container registry? 65
Summary 68

PART II CONTAINER ORCHESTRATION

Chapter 6 Container Orchestration with Kubernetes 71

What is Kubernetes? 72
Kubernetes architecture 73
How does Kubernetes work? 74
Local Kubernetes cluster 75
kubectl 77
How do you deploy an application to the Kubernetes cluster? 78
Summary 82

Chapter 7 Azure Kubernetes Service 83

How Azure Kubernetes Service can help you 83
What does it cost? 84
What resources do you need? 84
Creating a cluster using Azure Kubernetes Service 84
What resources were created with the cluster? 88
Connecting to the cluster 90
Using the local machine 90
Using the Azure Cloud Shell 92
Deploying an application 92
Summary 95

Chapter 8 Azure SQL Database and Azure Key Vault 97

Azure SQL Database 97
Creating the database 98
Configuring the connection 103
Previewing the Azure SQL Database 106

Azure Key Vault 107
Creating the key vault. 107
Retrieving the secret 108
Configuring the key vault permissions 110
Updating the application 110

Summary 112

Chapter 9 Diagnosing and Monitoring the Cluster 115

Kubernetes resources 115
Namespaces, workloads, and services 115
Storage and configuration 117
Running commands 119

Monitoring. 119
Alerts and metrics 119
Container Insights 121

Summary 125

Chapter 10 Application Rollouts and Rollbacks 127

How to retrieve the Pod name. 127

Rollout. 130

Rollout strategies 132

Rollout history and rollbacks 134

Manual scaling 137

Summary 138

PART III AUTOMATION

Chapter 11 Automating Image Builds and Deployment Using CI/CD Pipelines 141

Setting up the workflow 141
Automatically building and pushing a Docker image 143
Configuring the Secret 147
Testing the workflow 151
Updating the deployment in Azure Kubernetes Service 153
Running an updated workflow 157
Automatic updates of the application 159
Summary 160

Chapter 12 Load Testing, Horizontal Pod Autoscaling, and Cluster Autoscaling 161

Load testing with Hey 161
Automating the load test using Docker 164
Building and running the containerized load tests 167
Horizontal Pod autoscaling 170
Configuring horizontal Pod autoscaling? 170
Applying horizontal Pod autoscaling? 172
Resource requests and limits for Pods and containers 173
Performance of the HorizontalPodAutoscaler under load test conditions 177
Summary 179

Chapter 13 Deploying and Managing Kubernetes Applications with Helm 181

Introduction to Helm 182
Creating a Helm chart 182
Rollouts and rollbacks 187
Integration with the CI/CD pipeline 190

Git branch and chart files. 190
Chart and values files 191
Deployment. 194
Service 196
HorizontalPodAutoscaler 197
Helpers 198
GitHub Actions 199
Testing the workflow. 201
Summary 203

Chapter 14 Infrastructure as Code 205

Introduction to Terraform 206
Deploying Azure Container Registry 207
Creating Terraform declarations for Azure Container Registry 208
Remote backend 209
Deploying the infrastructure 210
Deploying Azure Kubernetes Cluster 213
Deploying a SQL database 216
Deploying a Key Vault 219
Integration with the CI/CD pipeline 221
Declaration files 222
Helm files 227
CI/CD declaration. 228
Testing the pipeline. 233
Summary 235
Conclusion 236

Index *237*

About the Author



DAWID BORYCKI is a multifaceted professional known for his significant contributions as a scientist, programmer, author, and conference speaker.

With a robust academic background Dawid has pioneered innovations like interferometric methods for brain and eye imaging. He has authored several books and numerous articles aimed at advancing programming and cloud technologies.

Dawid is also a seasoned consultant and trainer, helping others master a range of technical skills.

This page intentionally left blank

Acknowledgments

I would like to express my deepest gratitude to my wife and children for their unwavering patience and support throughout the writing of this book. Their encouragement and understanding have been invaluable, allowing me to dedicate the necessary time and effort to bring this work to fruition. This book would not have been possible without their love and support. Thank you for being my pillars of strength.

This page intentionally left blank

Preface

I wrote this book because of my extensive experience delivering trainings on Docker, Kubernetes, and Azure for developers, DevOps teams, and IT professionals. One piece of feedback from those courses particularly stood out:

> *Dawid is really good at what he does, he makes something very complicated seem like a stroll in the park. This course is well constructed and very well presented. I finished this course with all the knowledge needed to start a new AKS-based web app or to migrate an existing app to AKS. Which cannot be said for any number of courses I already watched on Udemy/LinkedIn Learning.*

This feedback was the direct trigger for writing this book. My goal is to enable others to gain thorough knowledge on Docker, Kubernetes, and Azure for application deployment.

What the book gives you is:

PRACTICAL EXPERIENCE

The book is based on real-world training sessions delivered to developers, DevOps specialists, and IT professionals, ensuring that the content is practical and applicable.

SIMPLIFIED COMPLEX CONCEPTS

The book breaks down complex topics in Docker, Kubernetes, and Azure into easily understandable segments, making advanced topics feel approachable and manageable.

COMPREHENSIVE KNOWLEDGE

It provides all the necessary knowledge to start a new AKS-based web app or migrate an existing app to AKS, addressing a gap that many online courses fail to fill.

STRUCTURED LEARNING PATH

The book offers a well-constructed and logically presented learning path, building on concepts step-by-step to facilitate thorough understanding.

REAL FEEDBACK-DRIVEN CONTENT

The content is directly influenced by feedback from course participants who appreciated the clarity and effectiveness of the teaching methods.

HANDS-ON EXAMPLES

Includes practical, hands-on examples and exercises that mirror real-world scenarios, enabling readers to apply what they learn immediately.

FOCUSED ON APPLICATION DEPLOYMENT

Special emphasis on deploying applications using Docker, Kubernetes, and Azure, making it highly relevant for modern cloud-based development and operations.

Foreword

I've always said, if you can do something twice, it's time to automate it. The advent of containerization along with a renewed focus as an industry on automation has energized the software development community. Containers allow you to ship not an entire operating system like a virtual machine, but to ship only the programming interfaces (syscals) that are needed to support an application and keep it running reliably. Combine this with automation, and a single individual has the power and influence of a team! Just imagine what a whole team could do with these tools and techniques!

Given that containers have become the new "atoms," you can use these atoms to build "molecules" that make up your systems and applications. These powerful building blocks can orchestrate sophisticated interactions between containers with Kubernetes at the center. You can effectively deploy an entire web farm in one go! Imagine bootstrapping your entire company with a single command!

Automated builds and deployment used to be considered eXtreme Programming but are now standard operating procedure in any software engineering best practice. At Microsoft, we used to fly in a team of consultants to consult on setting up a load balancer, and cluster auto scaling was nearly unheard of. Today, open-source software like Kubernetes along with the Azure Cloud and infrastructure as code tools can turn us all into 10X developers.

Dawid has taken his extensive practical experience and poured it into this fantastic book. He breaks down these complex topics in a way that is both approachable while unapologetically detailed and complete. I truly appreciate how thoughtfully organized the learning path is as he builds each chapter on the previous. Just a few pages in and I was following along with actual examples that work great on macOS, Windows, or Linux. By the end, I'd put together a complete solution that was building in GitHub Actions and happily deploying to the cloud.

While *Modern App Deployment with Azure Kubernetes* may seem daunting, you've got the formula for success in your hands. Stick with Dawid and this book, and you'll be a pro in no time!

Scott Hanselman
VP Developer Community, Microsoft

This page intentionally left blank

Introduction

Containerization has revolutionized the way software applications are developed, packaged, and deployed, offering a lightweight, portable approach that simplifies application management across diverse environments. By encapsulating applications along with their dependencies into containers, developers achieve consistency and reliability, significantly reducing deployment challenges and enhancing productivity. This approach enables rapid deployment cycles, streamlined updates, and improved scalability, setting a foundation for efficient software delivery.

Azure Kubernetes Service (AKS) builds upon containerization by providing a robust and managed orchestration environment for deploying, scaling, and operating containerized applications in the cloud. AKS simplifies the complexities inherent in managing Kubernetes clusters, seamlessly integrating with essential Azure services, such as Azure SQL Database, Key Vault, and comprehensive monitoring tools. Automation further complements this ecosystem, enabling continuous integration and delivery through powerful tools like GitHub Actions and infrastructure automation via Terraform. This end-to-end automation provides rapid, consistent deployments and empowers teams to effectively scale their applications, maintain high availability, and optimize resource utilization.

Organization of this book

This book is designed to guide you through the essential concepts and practical applications of containerization, Kubernetes, and automation for application deployment.

This book is divided into three comprehensive parts:

- **Part I: Containerization**. This part contains five chapters:
  - Chapter 1: Motivation
  - Chapter 2: Application
  - Chapter 3: Containerization with Docker
  - Chapter 4: How to Containerize an Application with Docker
  - Chapter 5: Azure Container Registry

In Part I of the book, you will learn about the primary motivations for introducing containerization and its relevance to modern application development and deployment. Specifically, Chapter 1 explores how containerization relates to Git, Kubernetes, infrastructure as code, and build/release pipelines. Chapter 2 introduces the starting point: an ASP.NET Core web application called People.WebApp, which stores data in SQL Server. This application represents a typical workload for many companies beginning their transition to the cloud. Chapter 3 covers Docker, explaining what it is and how it functions. You will learn how to use Docker to manage and run containers, as well as how to handle multi-container applications with Docker Compose. Chapter 4 then guides you through containerizing an ASP.NET Core web application. You will learn how to set up a multi-container solution where the app runs in one container and communicates with SQL Server in another. Finally, Chapter 5 introduces container registries. In this chapter, you will create a registry in Azure and push a container image of People.WebApp to the cloud, enabling deployment of the application to a Kubernetes cluster in Part II.

- **Part II: Container Orchestration.** This part contains five chapters:
  - Chapter 6: Container Orchestration with Kubernetes
  - Chapter 7: Azure Kubernetes Service
  - Chapter 8: Azure SQL Database and Azure Key Vault
  - Chapter 9: Diagnosing and Monitoring the Cluster
  - Chapter 10: Application Rollouts and Rollbacks

Part II is about container orchestration, a process that simplifies the running and management of distributed systems composed of hundreds of containers. These containers must run across many virtual machines, either deployed on-premises or in cloud environments. Chapter 6 describes Kubernetes, the open-source and most popular orchestration platform, to demonstrate how to deploy containerized applications with container orchestrators. The goal of this part is to show how to use a local development Kubernetes cluster and then deploy the containerized application from the container registry to a production Kubernetes cluster running in the cloud within the Azure Kubernetes Service (Chapter 7). In Chapter 8, you will learn how to connect the application running in the cluster to other Azure services, such as SQL and KeyVault. Chapter 9 shows how to monitor the containerized application deployed to the cluster. Finally, Chapter 10 explains how to quickly perform application rollouts and rollbacks.

- **Part III: Automation.** This part has four chapters:
  - Chapter 11: Automating Image Builds and Deployment Using CI/CD Pipelines
  - Chapter 12: Load Testing, Horizontal Pod Autoscaling, and Cluster Autoscaling

- Chapter 13: Deploying and Managing Kubernetes Applications with Helm
- Chapter 14: Infrastructure as Code

The aim of Part III is to demonstrate how to automate the application deployment. I will show you how to incorporate continuous integration/continuous delivery (CI/CD) pipelines to gain full control over the application and underlying resource deployment. Specifically, in Chapter 11, you will use GitHub Actions to create the CI pipeline, which will automate the building and pushing of Docker images to Azure Container Registry. This pipeline will be triggered whenever the source code is updated, and a pull request is accepted. In Chapter 12, you will explore how to automatically adjust the application's size (the number of Pods) using the Horizontal Pod Autoscaler (HPA) and adjust the cluster's size depending on the current load. Chapter 13 will delve into deploying applications using Helm charts, enabling you to manage multiple declarations of deployments, services, and accompanying Kubernetes objects, such as HPA. Finally, in Chapter 14, you will learn how to employ Infrastructure as Code (IaC) to deploy and update cloud resources in a declarative way, enabling you to achieve a complete continuous deployment pipeline triggered by changes in the source code.

Audience

This book is intended for readers seeking to deepen their understanding and skills in deploying modern applications on the cloud using Azure's powerful Kubernetes platform. It is an excellent choice for developers, DevOps professionals, and IT experts interested in mastering modern application deployment strategies on Azure.

Companion code

This book includes companion code to enrich your learning experience. The companion code is organized into two repositories:

PEOPLE.WEBAPP Contains the source code of the application that will be containerized and deployed to Azure Kubernetes Service.

github.com/dawidborycki/People.WebApp or *MicrosoftPressStore.com/modernapp/downloads*

PEOPLE.WEBAPP.DECLARATIONS Includes declarations such as Dockerfiles, YAML files with Kubernetes manifests, and Terraform YAML files for infrastructure deployment.

github.com/dawidborycki/People.WebApp.Declarations or *MicrosoftPressStore.com/modernapp/downloads*

Errata, updates, & book support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

MicrosoftPressStore.com/modernapp/errata

If you discover an error that is not already listed, please submit it to us at the same page.

For additional book support and information, please visit *MicrosoftPressStore.com/Support*

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *http://support.microsoft.com*

Stay in touch

Let's keep the conversation going! We're on X: *http://x.com/MicrosoftPress*

CHAPTER 3

Containerization with Docker

This chapter provides an overview of containerization and Docker, delving deeper into the motivation for using containers in application development and deployment. You will learn how to use Docker to create and manage containers and put the People.WebApp to work.

Why use containers?

Containers have become a critical technology with many advantages over traditional approaches to managing development environments and deploying applications. Contrasting these advantages with the limitations of traditional methods will clarify why containers, and specifically Docker, have gained widespread adoption in the industry.

Imperative vs. declarative configuration

Remember the old days of preparing your development machine? You'd install every application individually, from your integrated development environment (IDE) to dependencies, in a process called *imperative configuration*. The state of your development machine was defined by executing a series of installation steps. If your hard drive crashed, you had to replace it and reinstall everything, a time-consuming process that left you unable to work during the restoration.

Hardware manufacturers and software companies proposed a better solution based on hard drive images. After installing all your tools, you could take a drive snapshot to create an image containing all your applications, dependencies, and data. In the event of a crash, you could quickly restore your machine using this image and dedicated software to replicate the image to a new hard drive. As you may remember from Chapter 1, "Motivation," this method is called *declarative configuration*; you declare a desired system state (defined as the image), and the software automatically updates it if necessary.

Declarative configuration using drive images has two advantages:

- **Disaster preparedness and rapid deployment** After you prepare your computer and take a snapshot of the drive, you can deploy this image to hundreds of devices in your company, even over a network, preparing for disaster and enabling rapid deployment of cloned systems.

- **Physical and virtual machines** You can use drive images to set up environments on both physical and virtual machines. In the cloud, you can use an image to create a virtual machine (VM) instance, which can differ from the host's operating system.

What is containerization, and how can it help you?

Given the success of disk and VM images, the next logical step was to apply this concept to application deployment. By packaging an application with all its dependencies into a container image, you can deploy the application as a container instance in local, virtual, or cloud environments. This process, known as *containerization*, allows for the creation and versioning of various container images to fully control deployment. Containerization packages together an application or service, its dependencies, and its configuration (abstracted as deployment manifest files) all into a single container image. This enables you to test and deploy the containerized application as a unit to any host operating system (OS).

In practice, containerization addresses multiple challenges. For instance, if a hypothetical application requires a specific runtime like the Visual C++ redistributable, it may work fine on your development machine but fail on a customer's machine due to missing dependencies. Containers package the application with all necessary runtimes and configurations, ensuring it functions identically everywhere.

Containers simplify application distribution by encapsulating the application, all its dependencies, configurations, and necessary data into a container image. You then distribute this image to end-users, who need to have only a container runtime installed on their machines. This ensures that everything you tested on your development machine works the same way on any other device, provided both machines use the same container runtime.

Another significant advantage of containerization is its ability to support applications or services developed using different programming languages and technologies, all running under the same container runtime. What does this mean? In modern application development, you select the right tool for each job, using dedicated tools to address specific challenges. Specifically, you might use:

- Python for machine learning and AI
- JavaScript, HTML, and CSS for frontends
- Java, Go, PHP, or .NET for backends

Furthermore, all containers can run under the same container runtime. This is similar to how multiple virtual machines, each powered by different guest operating systems, can run simultaneously on a single host machine using the same virtualization software. This is feasible because each container image encapsulates a specific part of the entire software solution—along with all the dependencies (like runtimes), configurations, and data. After the container image is created, the container runtime does not concern itself with the programming language or technology used to develop the app or service.

To better understand software containers and containerization, think of the transportation industry. Teams pack items—cars, electronics, clothing, tools, and more—into physical containers. Trucks and container ships then transport these physical containers (Figure 3-1). Because the containers conform to standard dimensions, they're easy to transfer from train cars to trucks to ships, which are all designed to be compatible with that standard. Importantly, the vehicles do not concern themselves with the contents of the containers, or other items.

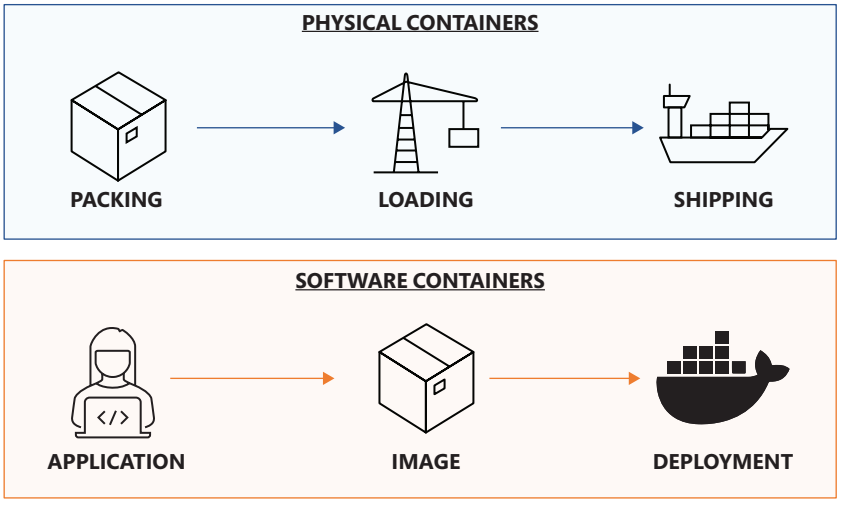

FIGURE 3-1 An analogy between physical and software containers.

The same principle applies to software containers. The container runtime is indifferent to the programming tools used to implement an application or service packaged within a container image. You only need to adhere to specific standards when creating the container image, ensuring the container runtime can correctly initiate one or more container instances from your image as shown in Figure 3-2.

Containers vs. virtual machines

Containers are often explained using their similarity to virtual machines (VMs), as shown in Figure 3-2. While containers are somewhat akin to VMs, there are key differences: VMs require more resources because they include the entire guest operating system (OS), applications, and their dependencies. Containers, on the other hand, encapsulate only the applications and their direct dependencies, without an entire guest OS. Therefore, containers are quicker to deploy and start but provide less isolation compared to VMs. The primary goal of a container image is to ensure consistency across different deployments, eliminating the "it works on my machine" issue.

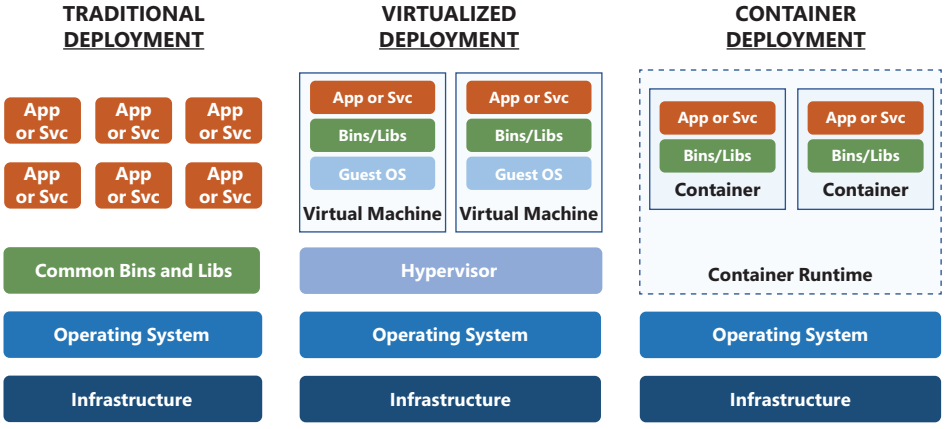

FIGURE 3-2 A comparison of the application deployment models.

Containers offer advantages over VMs in terms of horizontal application scaling. Using VMs to provision multiple instances of an application or service consumes more resources than necessary, as each instance includes the entire operating system. Modern applications, often composed of many components typically designed using microservices, allow each service to be developed independently with different, problem-specific tools and programming languages. Each service can be packaged as a container image and scaled separately, allowing for more precise adjustments to scaling based on actual traffic and application-specific needs.

For example, a typical media streaming platform might include several microservices responsible for user authentication, content browsing, media playback, and recommendation engines. Users typically spend much more time browsing or watching content than logging in. Therefore, in practice, more instances of the browsing and playback services are needed compared to the login or user profile services.

Scaling such an application using VMs would involve scaling the entire application along with the OS and all dependencies, thereby using more hardware resources than necessary. This challenge can be addressed at the app or service level by scaling using containers. In this approach, individual services are scaled horizontally, allowing, for example, for 15 instances of the media playback service and only two of the authentication service. Furthermore, these services can be deployed and updated independently. Although there is still some overhead due to the deployment of app or service dependencies in each container, this overhead is significantly smaller than that associated with VMs.



Note Containers provide a lighter, faster alternative to virtual machines because they do not require an entire guest operating system. This results in quicker startup times and more efficient resource utilization, especially when scaling microservices individually.

What is Docker?

Docker is a renowned open platform for automating and deploying applications as portable, self-sufficient containers that can run either in the cloud or on-premises. The only requirement is that the given machine must provide a Docker host, which serves as the containerization manager and runtime. Docker is also the name of the company that promotes and develops this containerization technology.

Developers can use Docker Desktop on Windows, Linux, or macOS. This tool includes a single-node Kubernetes cluster, which developers can use for local deployments and testing.

Docker is implemented in the Go programming language and employs namespaces technology to provide an isolated workspace, the container. These namespaces are used to isolate running containers. Specifically, a running container has access only to its associated set of namespaces and cannot access the namespaces of other containers. In virtual machines, such isolation is achieved with a hypervisor.

Architecture

The Docker architecture, depicted in Figure 3-3, uses a client-server approach in which a Docker client communicates with the Docker daemon running on the server (Docker Host). The Docker daemon is a background process (server) that listens to requests from the client. These requests are usually sent using the Docker Command Line Interface (Docker CLI) or the remote API.

The major Docker CLI commands include:

- **`docker pull`** Downloads a container image from a container registry, which is a service that can contain multiple repositories for container images, like Git. There are local and remote repositories, so the **docker pull** command (like **git pull**) downloads an image from a remote repository.
- **`docker push`** Sends a local container image to a remote repository.
- **`docker build`** Builds a container image by turning the application, its dependencies, and its configuration into a self-sufficient container image, which can then be sent to a remote repository or used to run the container locally.
- **`docker run`** Creates and runs a container instance from the specified container image. This command first tries to locate the selected container image in the local repository. If the container is unavailable locally, the command pulls the image from the remote repository hosted by the container registry.

By default, Docker uses Docker Hub as the container registry, where you can create public or private repositories. Private repositories are used for images that you do not want to share with others.

Notably, the Docker client and daemon can run on the same machine, as is the case when Docker Desktop is installed.

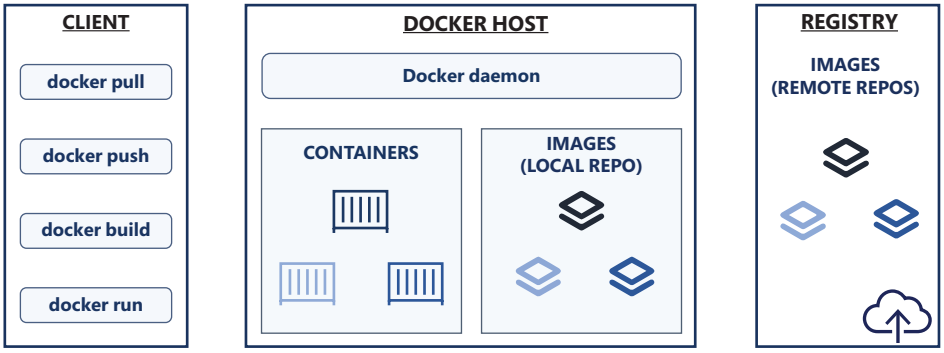

FIGURE 3-3 Docker architecture.

How to pull and run a container

To use the Docker CLI to pull a container image and run it on a local Docker host, you must first ensure that Docker Desktop is running (see Figure 3-4).

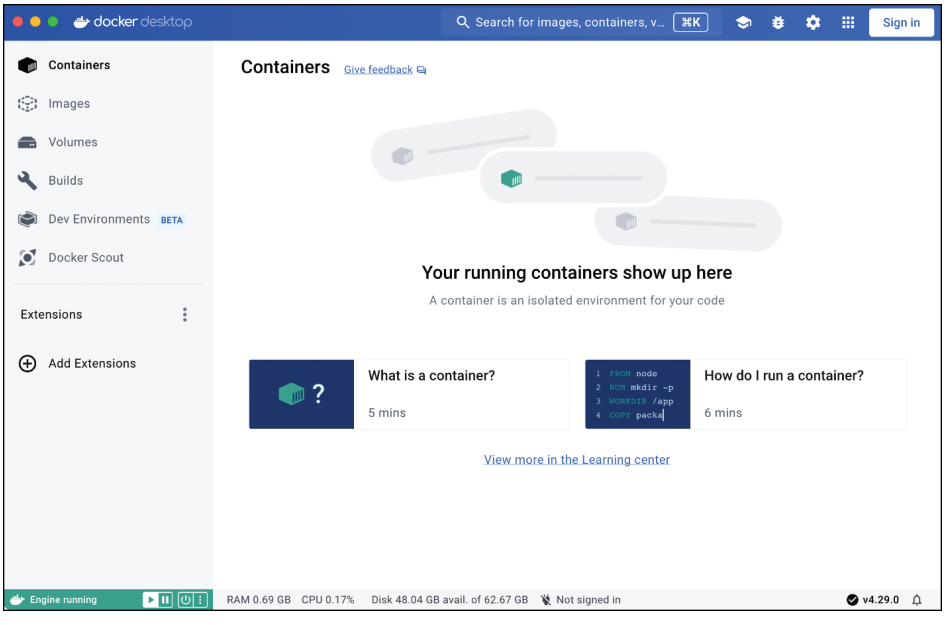

FIGURE 3-4 Docker Desktop.

The icon in the bottom-left corner indicates the status of the Docker Engine (which includes both the daemon and client). The icon should be green. If it is red, you might want to restart the engine or troubleshoot using the Docker documentation.

After ensuring that Docker is up and running, it is helpful to familiarize yourself with the available tabs in Docker Desktop:

- **Containers** Shows the list of running containers. When you start Docker Desktop for the first time, this list will be empty. The Containers tab will display instructions on how to run a container.
- **Images** Includes the list of images in your local and remote repositories.
- **Volumes** Displays the list of volumes, which can be attached to containers to provide non-volatile storage.
- **Builds** Contains the list of builds and builder configurations.
- **Dev Environments** Enables you to manage Dev Environments, which you can use to create reusable environments for developers that contain all the tools they need.
- **Docker Scout** Enables advanced analysis of Docker images.

You can access the same lists through the Docker CLI.

Now, back to the task at hand: To pull the container image with the ASP.NET Core sample app, open the command line or terminal, and then type:

```
docker pull mcr.microsoft.com/dotnet/samples:aspnetapp
```

The command downloads the Docker image from the .NET repository hosted on Microsoft's public container registry (see Figure 3-5). Note the last string after the colon (**aspnetapp**). This is the image tag, an additional label you can apply to images and use for versioning. Docker will use the most recent version of the tag if you do not specify an explicit tag.

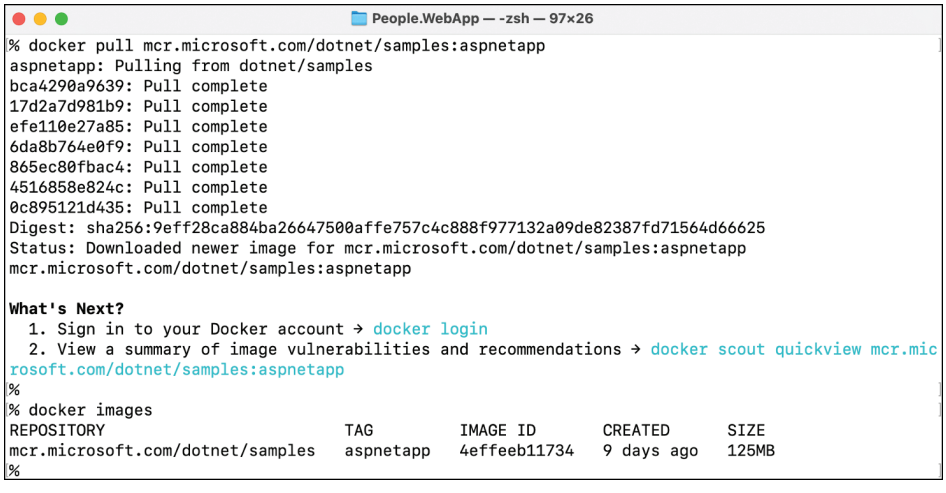

FIGURE 3-5 Pulling a Docker image.

By analyzing the output of the **docker pull** command, you can see that the pull operation is composed of several steps, because this specific image contains multiple layers. Docker uses layers to optimize container builds and transfers, as detailed in the Dockers Layers Guide (*docs.docker.com/build/guide/layers*).

To verify that the image was downloaded, type:

```
docker images
```

The command lists the images available in your local repository. If you return to Docker Desktop and click the Images tab, you will see the same image (as shown in Figure 3-6).

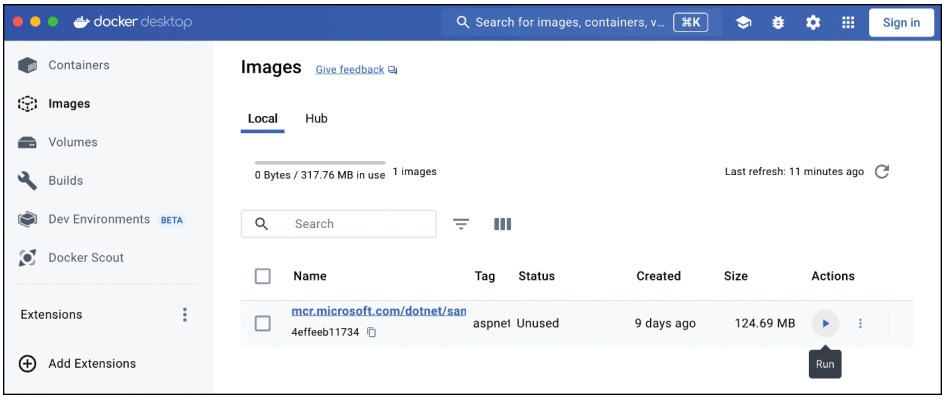

FIGURE 3-6 A list of local images.

To run a container, use the Docker CLI. In the command line or terminal, type (see Figure 3-7):

```
docker run -p 80:8080 --name aspnet-sample-app mcr.microsoft.com/dotnet/↵
    samples:aspnetapp
```

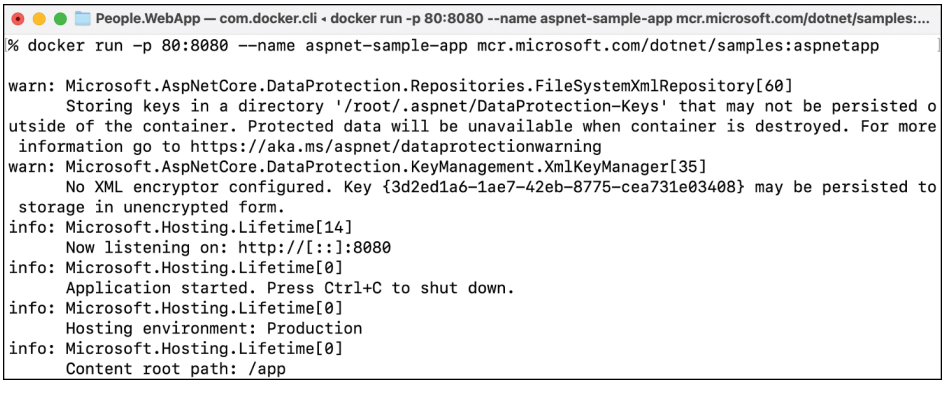

FIGURE 3-7 Running a container.

This command runs the container named aspnet-sample-app using the **--name** parameter. If you omit this parameter, Docker will assign a random name to the container. The running container will be accessible from the host on port 80, which will be mapped to port 8080 inside the container. This mapping is configured using the **-p** parameter. The first value is the host port, and the second is the container port. Finally, you provide the image to spin up the container. If the image was not pulled beforehand, the **docker run** command will pull it for you as shown earlier in Figure 3-6.

The container is up and running in the background. To access the application, open a web browser and type **localhost** in the address bar. The application will appear as shown in Figure 3-8.

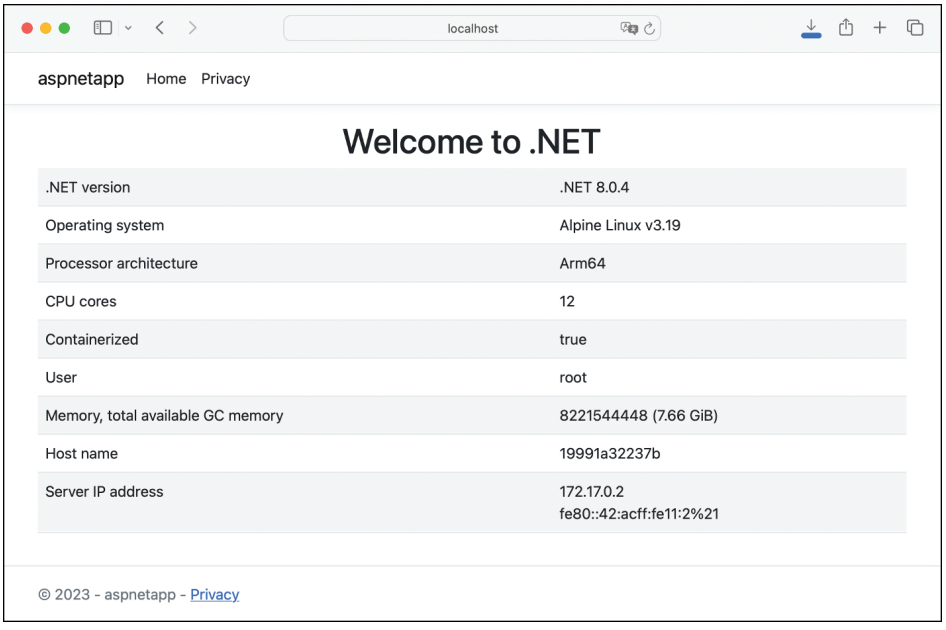

FIGURE 3-8 Running a containerized ASP.NET Core application.

How to manage containers

To see a list of running container instances, go to Docker Desktop and click Containers. The Containers tab lists each running container's name (aspnet-sample-app) and identifier (19991a32237b), as shown in Figure 3-9. It also displays the name of the image used to spin up the container, the container's status, port mapping, uptime, and additional actions. You can use these actions to stop, pause, or restart the container.

Additionally, if you click the ellipsis under Actions and select View Details, you will see the app's output and the container's stats, such as CPU and memory usage. From this view, click Exec to access the terminal inside the running container. From here, you can invoke any command available in the container, for example, *top* (see Figure 3-10). The available commands will depend on the base image used to create the container and the tools installed within it.

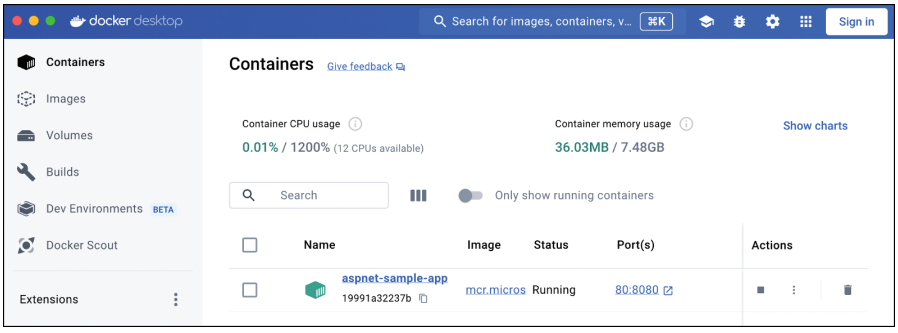

FIGURE 3-9 Docker Desktop showing a containerized ASP.NET Core application.

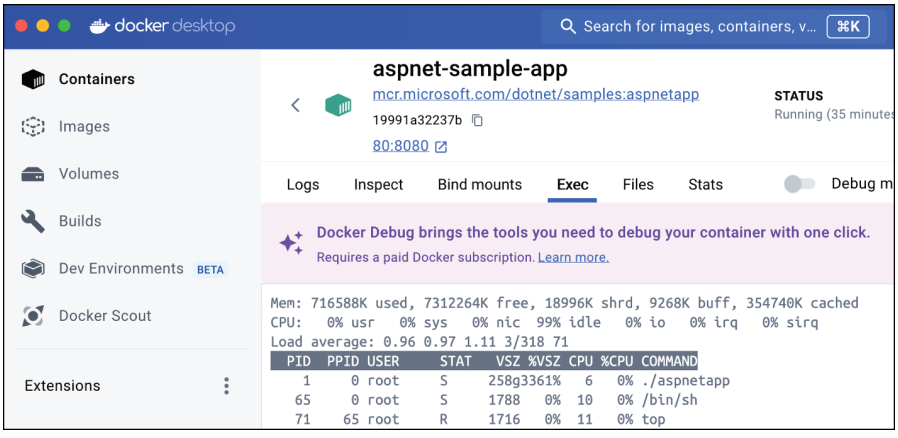

FIGURE 3-10 Connecting to a terminal of the running container.

Importantly, you can achieve similar results using the Docker CLI. Specifically, to get a list of running containers, use the command:

```
docker ps
```

To connect to a terminal inside a running container named aspnet-sample-app, you can use the following command:

```
docker exec -it aspnet-sample-app /bin/sh
```

This command executes **/bin/sh** inside the running container. Note the usage of the **-it** flag, which indicates that the command should be invoked in interactive mode, giving you access to the standard input for the container (here, a command line or terminal).

How to remove a container

When you launch a container, the command line displays the app's output running in the container. If you close the output by pressing Ctrl+C (Windows) or Command+C (macOS), the application running in the container will stop and the container's status will change to Exited. Consequently, you cannot

re-run the container with the same name. Therefore, you need to delete the aspnet-sample-app container, either by clicking the Trash icon in Docker Desktop or by typing:

```
docker rm aspnet-sample-app
```

Finally, Docker Desktop enables you to run a local image using the UI. To do this, open the Images tab of Docker Desktop, select a container image, and click its Run icon under Actions (Figure 3-6). In the Run A New Container window that appears, you configure the container's name, port mapping, volumes, and more, as shown in Figure 3-11.

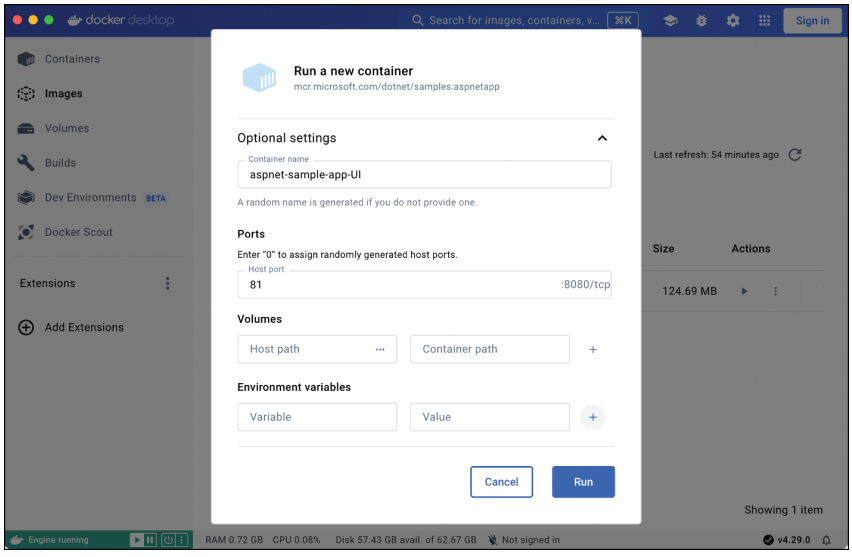

FIGURE 3-11 Running a container.

Running multiple containers

Consider the processes (instances of applications) in the operating system. Typically, multiple applications run in parallel. The applications can use the system's storage to persist files or data, interact with the system services, or use the network to communicate with external applications and services. For example, an application can consume a database in the Microsoft SQL Server running as a system service. Also, the same application can save photos to a drive and synchronize its local data with the remote service provisioned in the cloud. Of course, multiple applications can run on the same computer, communicating with the same Microsoft SQL Server but accessing different databases.

To achieve a similar functionality with multiple containerized applications, you simply need to extend what you now know about running and managing an individual container. Docker offers some additional tools to help you, including container networking, volumes, and Docker Compose.

As you remember, a container is created from a container image. The container then acts as a self-contained unit. It can process and store data, but when you close the container, the data stored in it is lost. To preserve this data, you can keep it on the host disk or in a database. You can use *volumes* to

allow the container to persist the data on the Docker host. The database can run inside the container or in another container. For the latter case, you need to *network* multiple containers. Finally, the *Docker Compose tool* enables you to run and manage multiple containers, their networks, and volumes using a declarative approach. Take a closer look at each tool in the following sections.

Volumes

Docker provides several mechanisms for sharing data between containers, all of which are detailed in the official Docker documentation. One common feature of these approaches is that the data becomes available inside the container as any other element of the container's file system. Therefore, the app or service running inside the container does not require any additional changes and can use the same data access methods as it would outside of containerization.

In this section, I will show you how to use volumes, which offer a preferred way to share data between containers. Specifically, you will learn how to create a volume, then spin up two containers that access the same volume.

To start, open Docker Desktop, navigate to the Volumes tab, and click Create (top-right corner). In the New Volume window that appears, name your volume **aspnet-volume**, and click Create (see Figure 3-12). A new volume will appear on the Volumes list in Docker Desktop.

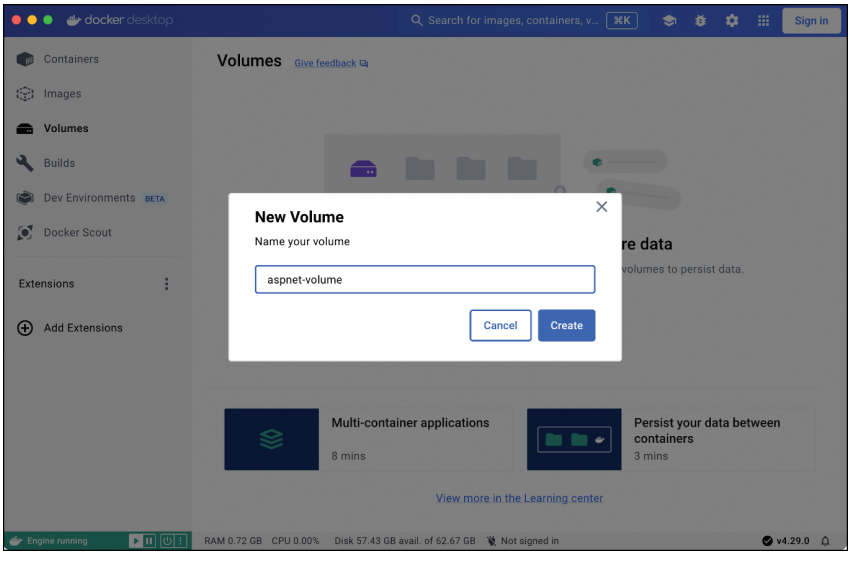

FIGURE 3-12 Creating a volume using Docker Desktop.

Alternatively, you could create the new volume with a Docker CLI command:

```
docker volume create --name aspnet-volume
```

To show the list of volumes using Docker CLI, type:

```
docker volume ls
```

With your volume ready, you can spin up the first container. Use the same container image as before and supplement the **docker** command with two additional parameters: **-d** to run the container in detached (background) mode and **-v** to attach aspnet-volume such that it will be accessible inside the container from the /etc/data folder:

```
docker run -dp 80:8080 --name aspnet-sample-app -v aspnet-volume:/etc/data ↵

    mcr.microsoft.com/dotnet/samples:aspnetapp
```

The command outputs the container identifier, as shown in Figure 3-13.



FIGURE 3-13 Attaching a volume to a container.

You can now go back to Docker Desktop, click the Containers tab, and then open the Terminal of the running container (under Actions select the ellipsis, View Details, then Exec). Then, in the container's Terminal, type the following commands:

```
ls -la /etc/data

echo "Hello from container 1">> /etc/data/test.file

cat /etc/data/test.file
```

The first command lists the contents of the /etc/data folder to confirm that the volume is initially empty. The second creates a new text file inside that folder. The last command displays the contents of the new file.

To view the items in the volume, you can use Docker Desktop. Click Volumes, select a volume, and then click Data.

One done, one to go. Spin up another container using the same container image as before. This new container will use the same volume (aspnet-volume) and port 81 on the host. To avoid a name conflict, name the second container aspnet-sample-app2:

```
docker run -dp 81:8080 --name aspnet-sample-app2 -v aspnet-volume:/etc/data ↵

    mcr.microsoft.com/dotnet/samples:aspnetapp
```

Again, the command outputs a container identifier. You now have two containers running, both with the same volume attached. In particular, the second container, aspnet-sample-app2, can access the test.file created from the aspnet-sample-app container. To see this, open the Terminal of the second container (aspnet-sample-app2) and type:

```
cat /etc/data/test.file
```

The message "Hello from container 1" displays as shown in Figure 3-14.

You've just learned how to use volumes to persist data. The file you created will be available to other containers until the volume is removed.

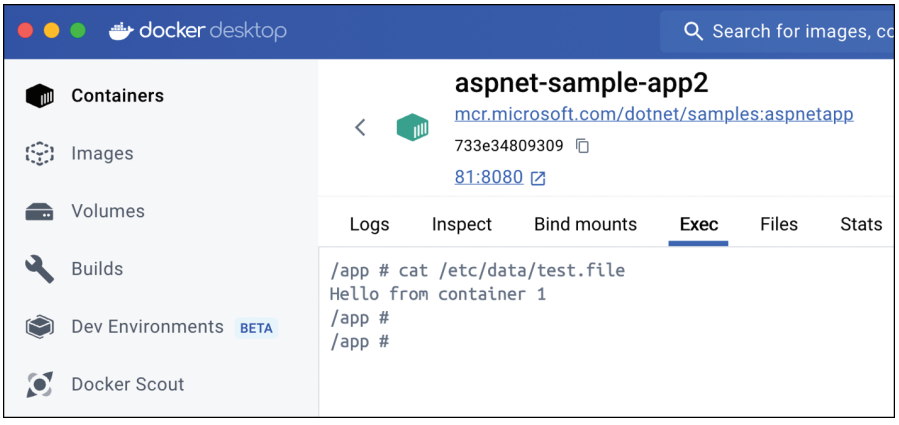

FIGURE 3-14 Accessing a file created by another container.

Networking

In the previous section, you spun up two containers. Both containers joined the default Docker network (bridge), which Docker creates along with two other default networks. This setup allows both containers to communicate with each other. To confirm this, use the Terminal of the first container (aspnet-sample-app) to type:

```
hostname -i
```

The command will output the local IP address of the first container within the Docker network (172.17.0.2, in my case). Open the Terminal of the second container (aspnet-sample-app2), and type the following commands:

```
hostname -i
```

```
ping 172.17.0.2
```

The first command displays the local IP address of the second container (172.17.0.3, in my case). The second command sends a ping to the first container, and you should see a response similar to what is shown in Figure 3-15. This confirms that both containers are inside the same network and can communicate with each other.

The practical implication of this communication capability is that you can create multi-container solutions where, for example, one container runs a web application, and another container runs a database server that the application consumes.

Moving forward, you can even create a more advanced network architecture where some containers are isolated within their own networks, so they cannot be reached by containers running in other networks.

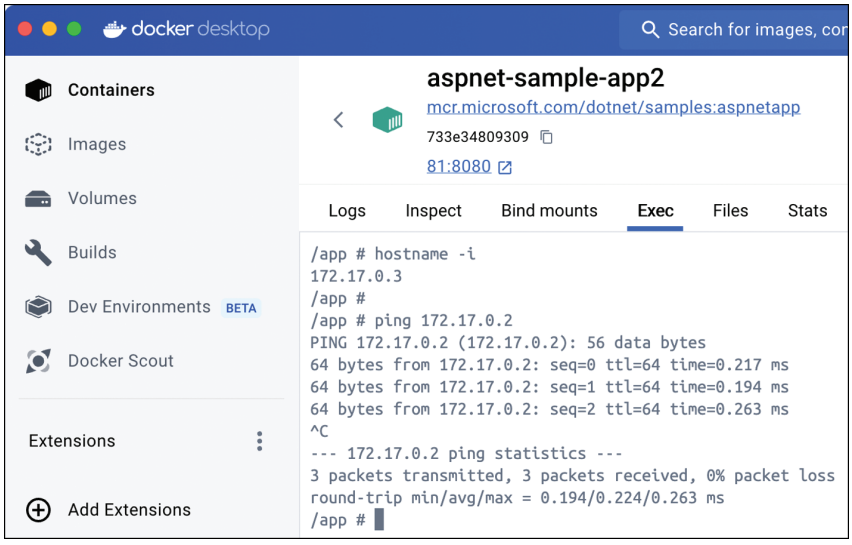

FIGURE 3-15 Communication Between Containers.

To create a new network, use the **docker network create** command. Once the network is created, you can instruct Docker to have a new container join that network using the **--network** parameter of the **docker run** command. Additionally, you can connect a running container to a specified network with the **docker network connect** command. For specific scenarios, you can refer to various Docker networking tutorials available in the official Docker documentation.



Note Containers use declarative configuration, allowing consistent, reproducible deployment by packaging applications and dependencies into images. This is a significant improvement over traditional, imperative setup methods, simplifying deployment and disaster recovery.

Docker Compose

Although your new containers can share data and communicate with each other, you had to create the volume and spin up the containers manually. Such manual management of Docker components is fine initially, but as your solution grows and becomes more complicated, you might prefer a more automated solution. To address this, Docker provides Compose.

Compose enables you to define, run, and manage Docker applications composed of multiple containers. To define a multi-container Docker application, you use a dedicated YAML file similar to Listing 3-1.

The file defines two application services: `people.webapp` and `sql-server`. The first corresponds to a hypothetical frontend, and the second is a database server. Thus, the compose creates two containers. It creates the first, `people.webapp`, using the container image with the sample ASP.NET Core app. It spins up the second container using an official containerized version of SQL Server.

The definition of the first service also includes port mapping (equivalent to the **-p** parameter of the **docker run** command) and volume creation and attachment.

LISTING 3-1 A Docker Compose YAML File Defines an App Composed of Two Containers.

```
services:
  people.webapp:
    image: "mcr.microsoft.com/dotnet/samples:aspnetapp"
    ports:
      - "8080:8080"
    volumes:
      - .:/usr/data  sql-server:
    image: "mcr.microsoft.com/mssql/server"
```

If you were to create such a solution manually, you would need to create a volume and then invoke the **docker run** command twice. Instead, you can use the Compose YAML file to provide a declaration for the manual container application and then provision all Docker components simply by issuing a single command:

```
docker compose up
```

To shut down the entire application, you use:

```
docker compose down
```

Notably, the containers created with Compose will join the same network and see each other by the hostnames set based on the service name. This means that `people.webapp` can reach `sql-server` via the service name on port 1433: sql-server,1433.



Tip Use Docker Compose to simplify multi-container application management. With a single YAML file, you can easily define, launch, and manage complex applications composed of multiple interconnected containers, significantly streamlining your workflow.

Summary

In this chapter, you learned how to use Docker for running and managing single and multi-container applications. In the next chapter, you will apply this knowledge to containerize and run People.WebApp. Then, you will learn how to use Docker Compose to launch a multi-container application consisting of People.WebApp and a containerized SQL Server. Before proceeding, clean up your Docker environment by closing both containers and removing a volume with the following commands:

```
docker rm -f aspnet-sample-app

docker rm -f aspnet-sample-app2

docker volume rm aspnet-volume
```

Index

A

accessing
  Azure Cloud Shell, 64
  files from other containers, 34
  GET method, Swagger UI, 17
  namespaces, 115
  web API, 15
ACR (Azure Cloud Registry), 92
adding
  Docker Compose files to workspaces, 47
  Microsoft SQL Server to containers, 48–50
  people to databases, POST method, 17–18
AKS (Azure Kubernetes Service), 153–157
  advantages of, 83–84
  clusters, Terraform deployments, 213–216
  cost of, 84
  resource requirements, 84
  role of, 83
alerts
  clusters, 119–120
  rules, 120
Analytics, Log, 123–125
analyzing resource usage, 173–176
annotating deployments, 135
API (Application Programming Interfaces)
  Kubernetes API, 73, 75
  web API
    accessing, 15
    data retrieval, web browser, 16
    testing methods, 19
apps (applications)
  cloning, 14
  configuring settings, 43–46
  containers, importance in app development, 6–7
  database connections, 14–15
  deploying, 92–95
    Helm, 188–189
    listing deployments, 130
    to clusters, 78–82
  in-memory databases, 15
  launching, 14–15
  load-testing
    automated load-testing, Docker, 164–167
    containerized load tests, 167–169
    Hey tool, 161–165
    HPA performance, 177–179
  manual scaling, 137
  modern app workflows, 55
  modern development and deployment workflow
    inner loops, 4
    outer loops, 4
    overview, 4–6
  rollbacks, 135–136
  rollouts
    listing deployments, 130
    load balancing, 132
    status, checking, 130–131
    strategies, 132–134
  running, 14, 50–52
  SQL Server, 15
  tags, listing, 129–130
  testing, 51–53
  uninstalling, Helm, 189–190
  updating, 127–131, 159
  upgrading, Helm, 188–189
  viewing, running apps, 14
  web API
    accessing, 15
    data retrieval, web browser, 16
architectures
  clusters, 73
  Docker, 25–26
attaching volumes to containers, 33
automating
  app updates, 159
  image builds
    building/pushing Docker images, 143–147
    CI/CD branches, 142
    cloning source code, 142

automating *(continued)*
importing repositories, 141–142
running updated workflows, 157–158
Secrets, 147–150
setting up workflows, 141–143
testing workflows, 151–152
triggering workflows with GitHub, 151
updating deployments, AKS, 153–157
workflow declaration files, 142–143
load-testing, Docker, 164–167
autoscaling
Cluster Autoscaler, 177–179
HPA, 170
applying, 172–173
configuring, 170–172
Helm charts, 186
Helm integration, 197–198
load-testing, 177–179
Azure, subscriptions, 9
Azure CLI (Command-Line Interface), 10
building
container registries, building, 63–64
resource groups, removing, 65
commands, invoking with Azure Cloud Shell, 64
Help, 60
output formats, 61–63
overview, 59–61
Azure Cloud Shell
accessing, 64
Azure CLI commands, 64
cluster connections, 92
building
container registries, 64
Helm charts, 182–183
resource groups, displaying, 64
Azure Container Registry, 56–59
Azure CLI
building container registries, 56–59
Help, 60
invoking commands with Azure Cloud Shell, 65
output formats, 61–63
overview, 59–61
removing resource groups, 65
Azure Cloud Shell
accessing, 64
building container registries, 64
displaying resource groups, 64
removing resource groups, 65
Azure Portal, building container registries, 56–59
defined, 56
deploying with Terraform, 208–209
listing
images, 68
repositories, 68
local images, pushing to container registries, 65–68
Azure Key Vault, 107
key vaults
building, 107
permissions, 110
Secrets
displaying, 108
retrieving, 108–109
Azure Portal
clusters
alerts, 119–120
metrics, 120–121
monitoring features, 119–125
commands, running, 119
Container Insights, 121–125
container registries, building, 56–59
namespaces
accessing, 115
building, 115
listing, 115–116
removing, 115
services
ingresses, 116–117
listing, 116–117
viewing, 116–117
Azure SQL Database
Basics tab, 100–101
building, 98–103
configuring, 99
connecting to, 106
connection strings, 97, 103–105
dashboard, 98–99
deploying databases, Terraform, 216–219
DTU, 97
elastic data pools, 98
Networking tab, 101–102
overview, 103, 106
People.WebApp example
modifying deployments, 104–105
querying data, 106–107
updating people, 105

previewing, 106–107
pricing options, 97–98
requirements, 97
Review tab, 102
serverless, 98, 103
servers, building, 100
vCore, 97

B

backends, remote, 209–210
balancing loads, Pods, 132
Basics tab, Azure SQL Database, 100–101
branches, CI/CD pipelines, building, 142
building
Azure SQL Database, 98–103
branches, CI/CD pipelines, 142
clusters with AKS, 84–87
configuration files, 47–48
container images, 40–42
container registries
Azure CLI, 56–59
Azure Cloud Shell, 64
Azure Portal, 56–59
containerized load tests, 167–169
Docker images, automated image builds, 143–147
Dockerfiles, 37–40
Helm charts, 182–187
images (automating)
automatic app updates, 159
building/pushing Docker images, 143–147
CI/CD branches, 142
cloning source code, 142
Docker images, building/pushing, 143–147
importing repositories, 141–142
running updated workflows, 157–158
Secrets, configuring, 147–150
setting up workflows, 141–143
testing workflows, 151–152
triggering workflows with GitHub, 151
updating deployments, AKS, 153–157
workflow declaration files, 142–143
key vaults, 107–108
namespaces, 115
services, 81
volumes, 32, 117
Builds tab, Docker Desktop, 27
Burns, Brendan
IaC, 236
Kubernetes operation, 74–75

C

charts, Helm
building, 182–187
CI/CD pipeline integration, 190–194
deployment template files, 186
HPA, 186
linting, 187
modifying, 183–186
rollbacks, 187–190
rollouts, 187–190
structure of, 182–183
checking
status of rollouts, 130–131
workflow statuses, GitHub Action extension, 151–152
CI/CD pipelines
branches, building, 142
declarations, Terraform, 228–233
Helm integration, 190
chart/value files, 191–194
deployments, 194–196
Git branches/chart files, 190–191
GitHub Actions, 199–201
helpers, 198–199
HPA, 197–198
services, 196–197
testing workflows, 194–196
IaC integration, 221–222
testing, Terraform, 233–235
claims, persistent volume, 117
CLI (Command-Line Interfaces)
Azure CLI, 10
building container registries, 56–59
Help, 60
invoking commands with Azure Cloud Shell, 64
output formats, 61–63
overview, 59–61
removing resource groups, 65
Docker CLI commands, 25, 32
cloning
apps, 14
People.WebApp example app, 14
source code, automated image builds, 142
closing containers, 36
cloud computing
benefits of, 3
role of, 3
Cloud Registry, Azure, 92

Cloud Shell, Azure
  accessing, 64
  Azure CLI commands, 64
  cluster connections, 92
  container registries, building, 64
  resource groups, displaying, 64
cloud-controller-manager, 74
Cluster Autoscaler, 177–179
clusters
  AKS
    building clusters, 84–87
    deploying clusters with Terraform, 213–216
  alerts, 119–120
  app deployments, 92–95
  architectures, 73
  configuring, ConfigMaps, 117–118
  connections, 77
    Azure Cloud Shell, 92
    local machines, 90–91
  Container Insights, 121–125
  containers
    listing, 123
    runtimes, 74
  Create Kubernetes Cluster wizard, 85–86
  defined, 72
  deploying apps to, 78–82
  kubectl
    displaying lists of nodes, 77
    displaying lists of services, 78
    retrieving lists of Pods, 77–78
  kubelets, 74
  kube-proxies, 74
  local clusters
    installing, 75–76
    setting up, 75–77
  metrics, 120–121
  namespaces
    accessing, 115
    building, 115
    listing, 115–116
    removing, 115
  Node pools, configuring, 86–87
  resource groups, listing, 88–90
  running, 81–82
  services
    ingresses, 116–117
    listing, 116–117
    viewing, 116–117
  storage, 117–118
code blocks, notation, 10
Command Palette, Visual Studio Code, 38
commands
  Azure CLI commands, invoking with Azure Cloud Shell, 64
  Docker CLI commands, 25, 32
  notation, 10
  running from Azure Portal, 119
Compose, Docker, 31–32, 35–36
  files, adding to workspaces, 47
  multi-container solutions, provisioning, 46–47
ConfigMaps, storing cluster configuration data, 117–118
configuration files
  building, 47–48
  notation, 10
configuring
  app settings, 43–46
  Azure SQL Database, 99
  clusters
    ConfigMaps, 117–118
    storing cluster configuration data, 117–118
  connection strings, 50, 103–105
  Container Insights, 121
  container settings, 43–46
  Helm charts, 183–186
  HPA, 170–172
  imperative configurations, 21
  key vault permissions, 110
  Node pools, 86–87
  Secrets, automated image builds, 147–150
connecting to
  Azure SQL Database, 106
  clusters
    Azure Cloud Shell, 92
    local machines, 90–91
connection strings
  Azure Key Vault, 107
    key vaults, building, 107
    key vaults, configuring permissions, 110
    Secrets, displaying, 108
    Secrets, retrieving, 108–109
  Azure SQL Database, 97, 103–105
  configuring, 50, 103–105
Container Insights, 121–125
Container Registry, Azure, 56–59
  Azure CLI
    building container registries, 56–59
    Help, 60
    invoking commands with Azure Cloud Shell, 65

output formats, 61–63
overview, 59–61
removing resource groups, 65
Azure Cloud Shell
accessing, 64
building container registries, 64
displaying resource groups, 65
removing resource groups, 65
Azure Portal, building container registries, 56–59
defined, 56
listing
images, 68
repositories, 68
local images, pushing to container registries, 65–68
containers/containerization
adding Microsoft SQL Server to, 48–50
advantages of, 21, 22–23
closing, 36
configuring settings, 43–46
declarative configurations, 35
defined, 22–23
files, accessing from other containers, 34
"Hello from container 1" messages, 33
identifiers, viewing, 32–33
images
building, 40–42
local images, pushing to container registries, 65–68
pulling with Docker, 26–28
importance in app development, 6–7
listing, Containers Insights, 123
load tests, 167–169
logs, People.WebApp example, 125
managing with Docker, 29–30
multi-container solutions, provisioning with Docker Compose, 46–47
multiple containers, stopping, 53
networking, 34–35
orchestration tools. *See* Kubernetes
People.WebApp example
adding Microsoft SQL Server, 48–50
building configuration files, 47–48
building container images, 40–42
building Dockerfiles, 37–40
configuring settings, 43–46
multi-container solutions, provisioning with Docker Compose, 46–47
running, 50–52
testing, 51–53
physical and software container comparisons, 23
pulling with Docker, 26–28
registries, building
Azure CLI, 56–59
Azure Cloud Shell, 64
Azure Portal, 56–59
removing with Docker, 30–31
role of, 3
running
multiple containers, 31–32
onnec, 44–45
with Docker, 28–29
with Docker Desktop, 31
runtimes, 74
VM vs.,23–24
volumes, attaching to containers, 33
Containers tab, Docker Desktop, 27
contents of volumes, listing, 33
control plane (Kubernetes), defined, 72
CPU (Central Processing Units), resource requests/limits, 173–176
Create Kubernetes Cluster wizard, 85–86
creating
branches, CI/CD pipelines, 142
clusters with AKS, 84–87
configuration files, 47–48
container images, 40–42
container registries
Azure CLI, 56–59
Azure Cloud Shell, 64
Azure Portal, 56–59
containerized load tests, 167–169
Docker images, automated image builds, 143–147
Dockerfiles, 37–40
Helm charts, 182–187
images (automating)
automatic app updates, 159
building/pushing Docker images, 143–147
CI/CD branches, 142
cloning source code, 142
importing repositories, 141–142
running updated workflows, 157–158
Secrets, 147–150
setting up workflows, 141–143
testing workflows, 151–152
triggering workflows with GitHub, 151

creating *(continued)*
updating deployments, AKS, 153–157
workflow declaration files, 142–143
key vaults, 107–108
namespaces, 115
servers, Azure SQL Database, 100
services, 81
volumes, 32, 117

D

dashboard, Azure SQL Database, 98–99
data retrieval (web API), web browser, 16
database connections
apps, 14–15
launching apps, 14–15
People.WebApp example app, 14–15
databases
adding people to databases, POST method, 17–18
app connections, 14–15
launching apps, 14–15
People.WebApp example app, 14–15
Azure SQL Database
Basics tab, 100–101
building servers, 100
configuring, 99
connecting to, 106
connection strings, 97, m 103–105
dashboard, 98–99
databases, deploying with Terraform, 216–219
Networking tab, 101–102
overview, 103, 106
People.WebApp example, modifying deployments, 104–105
People.WebApp example, querying data, 106–107
People.WebApp example, updating people, 105
previewing, 106–107
Review tab, 102
vCore, 97
connections
apps, 14–15
launching apps, 14–15
People.WebApp example app, 14–15
DTU, 97
in-memory databases, 15
SQL Database, Azure
building, 98–103
DTU, 97
elastic data pools, 98
pricing options, 97–98
requirements, 97
serverless, 98, 103
declaration files
CI/CD pipelines, 228–235
Terraform, 222–227
workflows, automated image builds, 142–143
declarative configurations, 21–22, 35
deploying
AKS clusters, Terraform, 213–216
annotating deployments, 135
apps, 92–95
Helm, 188–189
modern development and deployment workflow, 4–6
to clusters, 78–82
Azure Container Registry, Terraform, 208–209
Azure SQL Database, People.WebApp example, 104–105
Dockerfiles, 40
Helm and CI/CD pipeline integration, 194–196
infrastructures, Terraform, 210–213
key vaults, Terraform, 219–221
modifying deployments, 104–105
People.WebApp example, 92–95, 104–105
rapid deployments, declarative configurations, 21
rollouts, listing deployments, 130
services, 81
updating deployments, AKS, 153–157
deployment template files, Helm charts, 186
Deployments, Kubernetes, 74–75
Desktop, Docker, 25–27
building volumes, 32
Builds tab, 27
Containers tab, 27
Dev Environments tab, 27
Docker Scout tab, 27
Images tab, 27
installing, 8–9
local Kubernetes clusters, 75–77
running containers, 31, 43–46
Volumes tab, 27
Dev Environments tab, Docker Desktop, 27
disaster preparedness, declarative configurations, 21
disk images, advantages of, 6
displaying
lists of
nodes, with kubectl, 77
services, with kubectl, 78
resource groups, Azure Cloud Shell, 64

Docker
architecture of, 25–26
automated load-testing, 164–167
CLI commands, 25, 32
Compose, 31–32, 35–36
adding files to workspaces, 47
provisioning multi-container solutions, 46–47
containers
accessing files from other containers, 34
attaching volumes to, 33
closing, 36
images, 40–42
managing, 29–30
networking, 34–35
pulling, 26–28
removing, 30–31
running, 28–29
running, Docker Desktop, 31
running, multiple containers, 31–32
defined, 25
Docker Desktop, 25–27
building volumes, 32
Builds tab, 27
Containers tab, 27
Dev Environments tab, 27
Docker Scout tab, 27
Images tab, 27
installing, 8–9
local Kubernetes clusters, 75–77
running containers, 31, 43–46
Volumes tab, 27
Docker Hub, 25
Dockerfiles
building, 37–38
deploying, 40
images
automated image builds, 143–147
pushing to container registries, 67–68
updating, 129–131
volumes, 32
attaching to containers, 33
listing contents of, 33
name conflicts, 33
removing, 36
showing with Docker CLI command, 32
DTU (Database Transaction Units), 97

E

elastic data pools, Azure SQL Database, 98
etcd, key-value store, 74

F

files
accessing from other containers, 34
configuration files, building, 47–48
declaration files
CI/CD pipelines, 228–235
Terraform, 222–227
workflows, automated image builds, 142–143
deployment template files, Helm charts, 186
Docker Compose files, adding to workspaces, 47
Dockerfiles, building, 37–38
Helm chart files
CI/CD pipeline integration, 190–194
Terraform, 227–228
OpenAPI files, exposing with Swagger, 16–17
state files, managing, 209–210

G

generator (Swagger), activating, 16
GET method, accessing Swagger UI, 17
Git
Helm and CI/CD pipeline integration, 190–191
installing, 8
role of, 3
GitHub
repositories
building Secrets, 150
importing, 141–142
workflows
running updated workflows, 157–158
statuses, 152
triggering, 151
GitHub Actions
Helm and CI/CD pipeline integration, 199–201
workflow statuses, 151–152
GitOps, 205

H

"Hello from container 1" messages, 33
Helm
apps
deploying, 188–189
uninstalling, 189–190
upgrading, 188–189
charts
building, 182–187
CI/CD pipeline integration, 190–194

Helm *(continued)*
deployment template files, 186
HPA, 186
linting, 187
modifying, 183–186
rollbacks, 187–190
rollouts, 187–190
structure of, 182–183
CI/CD pipelines, 190
chart/value files, 191–194
deployments, 194–196
Git branches/chart files, 190–191
GitHub Actions, 199–201
helpers, 198–199
HPA, 197–198
services, 196–197
testing workflows, 194–196
commands, running from Azure Portal, 119
introduction to, 182
repositories, 182
Terraform and Helm files, 227–228
Help, Azure CLI, 60
helpers, Helm and CI/CD pipeline integration, 198–199
Hey tool, load-testing apps, 161–165
histories, rollouts, 134
HPA (HorizontalPodAutoscaler), 170
applying, 172–173
configuring, 170–172
Helm and CI/CD pipeline integration, 197–198
Helm charts, 186
load-testing, 177–179
performance, 177–179

I

IaC (Infrastructure as Code), 205
role of, 3
state files, managing, 209–210
Terraform, 206–207
AKS clusters, deploying, 213–216
Azure Container Registry deployments, 208–209
Azure SQL Databases, deploying, 216–219
CI/CD pipelines, 221–222, 228–235
declaration files, 222–227
Helm files, 227–228
infrastructures, deploying, 210–213
key vaults, deploying, 219–221
remote backends, 209–210
identifiers (containers), viewing, 32–33
images
automating builds
automatic app updates, 159
building/pushing Docker images, 143–147
CI/CD branches, 142
cloning source code, 142
Docker images, building/pushing, 143–147
importing repositories, 141–142
running updated workflows, 157–158
Secrets, configuring, 147–150
setting up workflows, 141–143
testing workflows, 151–152
triggering workflows with GitHub, 151
updating deployments, AKS, 153–157
workflow declaration files, 142–143
container images
building, 40–42
pulling with Docker, 26–28
disk images, advantages of, 6
docker images, pushing to container registries, 67–68
listing, Azure Container Registry, 68
local images, pushing to container registries, 65–68
updating, 129–131
Images tab, Docker Desktop, 27
imperative configurations, 21
importing repositories, 141–142
in-memory databases, 15
infrastructure deployments, Terraform, 210–213
ingresses
listing, 116–117
viewing, 116–117
inner loops, modern app development and deployment workflows, 4
Insights, Container, 121–125
installing
Docker Desktop, 8–9
Git, 8
local Kubernetes clusters, 75–77
NET 8 SDK, 8
Visual Studio Code, 9

J - K

k8s. *See* Kubernetes
Key Vault, Azure, 107
key vaults
building, 107

configuring permissions, 110
deploying with Terraform, 219–221
Secrets
displaying, 108
retrieving, 108–109
key-value store, etcd, 74
KQL (Kusto Query Language), 123–125
kubectl
commands, running from Azure Portal, 119
namespaces
accessing, 115
building, 115
listing, 115–116
removing, 115
services
ingresses, 116–117
listing, 116–117
viewing, 116–117
Kubernetes (k8s)
advantages of, 72–73
AKS
advantages of, 83–84
cost of, 84
resource requirements, 84
role of, 83
updating deployments, 153–157
alerts, 119–120
API, 73
API Server (kube-api-server), 75
cloud-controller-manager, 74
clusters
app deployments, 92–95
architectures, 73
building with AKS, 84–87
configuring Node pools, 86–87
connections, Azure Cloud Shell, 92
connections, local machines, 90–91
container runtimes, 74
defined, 72
deploying apps to, 78–82
displaying lists of nodes with kubectl, 77
displaying services of nodes with kubectl, 78
kubectl, cluster connections, 77
kubelets, 74
kube-proxy, 74
listing resource groups, 88–90
local clusters, installing, 75–77
local clusters, setting up, 75–77
retrieving lists of services with kubectl, 78
running, 81–82
ConfigMaps, storing cluster configuration data, 117–118
Container Insights, 121–125
control plane, defined, 72
Create Kubernetes Cluster wizard, 85–86
defined, 71–72
Deployments, 74–75
etcd, key-value store, 74
kube-controller-manager, 73–74
kubectl
cluster connections, 77
displaying lists of nodes, 77
displaying lists of services, 78
retrieving lists of Pods, 77–78
kubelets, 74
kube-proxies, 74
kube-scheduler, 73
load balancing, 132
metrics, 120–121
monitoring, 119–125
operating, 74–75
Pods, 74–75, 132
Secrets, 117–118
Services, 75
services
building, 81
deploying, 81

L

launching apps, database connections, 14–15
limits/resource requests, 173–176
linting, Helm charts, 187
listing
alert rules, 120
containers in clusters, 123
contents of volumes, 33
images, Azure Container Registry, 68
ingresses in clusters, 116–117
namespaces, 115–116
nodes with kubectl, 77
Pods, 77–78, 131
ReplicaSets, 130–131
repositories, Azure Container Registry, 68
resource groups, 88–90
rollouts, deployments, 130
running containers, 44–45
services
in clusters, 116–117
with kubectl, 78
tags, 129–130

lists, Web API users, retrieving lists of, 53–54
load balancing, Pods, 132
load-testing apps
  automated load-testing, Docker, 164–167
  containerized load tests, 167–169
  Hey tool, 161–165
  HPA performance, 177–179
local clusters
  installing, 75–76
  setting up, 75–77
local images, pushing to container registries, 65–68
local machines, cluster connections, 90–91
Log Analytics, 123–125
logs (container), People.WebApp example, 125

M

managing containers with Docker, 29–30
manual scaling, Pods, 137
memory, resource requests/limits, 173–176
metrics, clusters, 120–121
Microsoft SQL Server, 15, 48–50
modern app development and deployment workflows, 4–6
modern app workflows, 55modifying
  app settings, 43–46
  Azure SQL Database, 99
  clusters
    ConfigMaps, 117–118
    storing cluster configuration data, 117–118
  connection strings, 50, 103–105
  Container Insights, 121
  container settings, 43–46
  Helm charts, 183–186
  HPA, 170–172
  imperative configurations, 21
  key vault permissions, 110
  Node pools, 86–87
  Secrets, automated image builds, 147–150
monitoring features (Azure Portal), 119–125
multi-container solutions, provisioning with Docker Compose, 46–47
multiple containers
  running with Docker, 31–32
  stopping, 53

N

name conflicts, volumes, 33
namespaces
  accessing, 115
  building, 115
  listing, 115–116
  Pod names, retrieving, 127–130
  removing, 115
NET 8 SDK, installing, 8
networking containers, 34–35
Networking tab, Azure SQL Database, 101–102
Node pools, configuring, 86–87
nodes
  listing with kubectl, 77, 78
  usage limits, 173–176
notation
  code blocks, 10
  commands, 10
  configuration files, 10

O

OpenAPI files, exposing with Swagger, 16–17
outer loops, modern app development and deployment workflows, 4
output formats, Azure CLI, 61–63

P

people, adding to databases, POST method, 17–18
People.WebApp example, 13
  automated image builds
    building/pushing Docker images, 143–147
    configuring Secrets, 147–150
    importing repositories, 141–142
    running updated workflows, 157–158
    testing workflows, 151–152
    updating deployments, AKS, 153–157
  automatic app updates, 159
  cloning, 14
  Cluster Autoscaler, 177–179
  connection strings, configuring, 103–105
  containerization
    adding Microsoft SQL Server, 48–50
    building configuration files, 47–48
    building container images, 40–42
    building Dockerfiles, 37–40
    configuring settings, 43–46
    multi-container solutions, provisioning with Docker Compose, 46–47
    running containers, 42–46
  container logs, 125
  database connections, 14–15

deploying, 92–95, 104–105
Helm and CI/CD pipeline integration
chart/value files, 191–194
deployments, 194–196
Git branches/chart files, 190–191
GitHub Actions, 199–201
helpers, 198–199
HPA, 197–198
services, 196–197
testing workflows, 190–191
HPA, 170
applying, 172–173
configuring, 170–172
load-testing, 177–179
IaC with Terraform, 228–233
AKS cluster deployments, 213–216
Azure Container Registry deployments, 208–209
Azure SQL Database deployments, 216–219
CI/CD pipeline integration, 221–222, 228–233
declaration files, 222–227
Helm files, 227–228
infrastructures deployments, 210–213
key vault deployments, 219–221
remote backends, 209–210
load-testing
automated load-testing, Docker, 164–167
containerized load tests, 167–169
Hey tool, 161–165
local images, pushing to container registries, 65–68
manual scaling, 137
namespaces
accessing, 115
building, 115
listing, 115–116
removing, 115
Pod names, retrieving, 127–130
querying data, 106–107
resource requests/limits, 173–176
rollbacks, 135–136, 187–190
rollouts, 130–131
Helm rollouts, 187–190
histories, 134
load balancing, 132
strategies, 132–134
running, 14, 50–52, 109
services
ingresses, 116–117
listing, 116–117
viewing, 116–117
Swagger
activating, 16
OpenAPI files, exposing, 16–17
UI, 17–19
tags, listing, 129–130
testing, 51–53
updating, 105, 110–112, 127–131
usage limits, 173–176
web API, 15–16
performance (HPA), load-testing, 177–179
permissions, key vault, 110
persistent volume claims, 117
physical and software container comparisons, 23
physical machines, declarative configurations, 21–22
pipelines, CI/CD
branches, building, 142
declarations, Terraform, 228–233
Helm integration, 190
chart/value files, 191–194
deployments, 194–196
Git branches/chart files, 190–191
GitHub Actions, 199–201
helpers, 198–199
HPA, 197–198
services, 196–197
testing workflows, 194–196
IaC integration, 221–222
testing, Terraform, 233–235
Pods
Cluster Autoscaler, 177–179
descriptions, 80–81
HPA, 170
applying, 172–173
configuring, 170–172
load-testing, 177–179
Kubernetes, 74–75
listing, 77–78, 131
load balancing, 132
manual scaling, 137
names, retrieving, 127–130
ReplicaSets, checking status, 130–131
resource requests/limits, 173–176
usage limits, 173–176
volumes, building for Pods, 117
Portal, Azure
clusters
alerts, 119–120
metrics, 120–121
monitoring features, 119–125
commands, running, 119

Portal, Azure *(continued)*
Container Insights, 121–125
container registries, building, 56–59
namespaces
accessing, 115
building, 115
listing, 115–116
removing, 115
services
ingresses, 116–117
listing, 116–117
viewing, 116–117
POST method, adding people to databases, 17–19
previewing, Azure SQL Database, 106–107
provisioning multi-container solutions, Docker Compose, 46–47
pulling containers with Docker, 26–28
pushing images
Docker images, automated image builds, 143–147
local images to container registries, 65–68

Q

queries
data queries, People.WebApp example, 106–107
KQL, 123–125

R

rapid deployments, declarative configurations, 21
registries
ACR, 92
Azure Container Registry
Azure CLI, building container registries, 56–59
Azure CLI, Help, 60
Azure CLI, invoking commands with Azure Command Shell, 65
Azure CLI, output formats, 61–63
Azure CLI, overview, 59–61
Azure CLI, removing resource groups, 65
Azure Cloud Shell, accessing, 64
Azure Cloud Shell, building container registries, 64
Azure Cloud Shell, displaying resource groups, 65
Azure Cloud Shell, removing resource groups, 65
Azure Portal, building container registries, 56–59
defined, 56
deploying, Terraform, 208–209
listing images, 68
listing repositories, 68
pushing local images to container registries, 65–68
container registries
building with Azure CLI, 56–59
building with Azure Cloud Shell, 64
building with Azure Portal, 56–59
pushing local images to container registries, 65–68
Docker Hub, 25
releases, defined, 182
remote backends, 209–210
removing
containers with Docker, 30–31
namespaces, 115
resource groups, 65
volumes, 36
ReplicaSets
listing, 130–131
Pods, checking status, 130–131
repositories
building Secrets, 150
Helm, 182
importing, 141–142
listing, Azure Container Registry, 68
running updated workflows, 157–158
workflow statuses, 152
resource groups
displaying in Azure Cloud Shell, 64
listing, 88–90
removing, 65
resource requests/limits, 173–176
retrieving
web browser data, web API, 16
lists of
Pods with kubectl, 77–78
Web API users, 53–54
Pod names, 127–130
Secrets, key vaults, 108–109
Review tab, Azure SQL Database, 102
rollbacks, 135–136, 187–190
rollouts
deployments
anotating, 135
listing, 130
Helm charts, 187–190
histories, 134
load balancing, 132
status, checking, 130–131
strategies, 132–134

rules, alerts, 120
running
  apps, 14, 50–52
  clusters, 81–82
  commands, from Azure Portal, 119
  containers
    listing running containers, 44–45
    load tests, 167–169
    multiple containers, 31–32
    People.WebApp example, 42–46
    with Docker, 28–29
    with Docker Desktop, 31
  People.WebApp example, 14, 50–52, 109
  updated workflows, automated image builds, 157–158
  viewing running apps, 14
runtimes, containers, 74

S

scaling pods, manual, 137
Secrets, 117–118
  building
    GitHub, 150
    Visual Studio Code, 149
  configuring, automated image builds, 147–150
  key vaults
    displaying, 108
    retrieving, 108–109
serverless Azure SQL Database, 98, 103
servers
  Azure SQL Database servers, building, 100
  Kubernetes API Server (kube-api-server), 75
  Microsoft SQL Server, 15, 48–50
  vCore, 97
services
  building, 81
  clusters
    AKS clusters, Terraform deployments, 213–216
    listing in clusters, 116–117
    viewing in clusters, 116–117
  deploying, 81
  Helm and CI/CD pipeline integration, 196–197
Services, Kubernetes, 75
setting up
  local Kubernetes clusters, 75–77
  workflows, automated image builds, 141–143
showing volumes with Docker CLI command, 32
software, physical and software container comparisons, 23
source code, cloning in automated image builds, 142
SQL Database, Azure
  Basics tab, 100–101
  building, 98–103
  configuring, 99
  connecting to, 106
  connection strings, 97, 103–105
  dashboard, 98–99
  database deployments, Terraform, 216–219
  DTU, 97
  elastic data pools, 98
  Networking tab, 101–102
  overview, 103, 106
  People.WebApp example
    modifying deployments, 104–105
    querying data, 106–107
    updating people, 105
  previewing, 106–107
  pricing options, 97–98
  requirements, 97
  Review tab, 102
  serverless, 98, 103
  servers, building, 100
  vCore, 97
SQL Server, 15, 48–50
state files, managing, 209–210
status checks
  rollouts, 130–131
  workflows with GitHub Action extension, 151–152
stopping multiple containers, 53
storage, clusters, 117–118
subscriptions, Azure, 9
Swagger
  generator, activating, 16
  OpenAPI files, exposing, 16–17
  UI, 17–19
    adding people to databases, POST method, 17–18
    containerized People.WebApp example, 44–45
    GET method, accessing, 17
    Try It Out button, 17

T

tags, listing, 129–130
Terraform
  Azure Container Registry deployments, 208–209
  CI/CD pipeline integration, 221–222
  CI/CD pipelines
    declarations, 228–233
    testing, 233–235

Terraform *(continued)*
declaration files, 222–227
deploying
AKS clusters, 213–216
Azure SQL Databases, 216–219
infrastructures, 210–213
key vaults, 219–221
Helm files, 227–228
introduction to, 206–207
remote backends, 209–210
testing
apps, 51–53
CI/CD pipelines, Terraform, 233–235
Helm and CI/CD pipeline integration workflows, 201–203
People.WebApp example, 51–53
POST method, 19
workflows, 151–152, 201–203
triggering workflows with GitHub, 151
Try It Out button, Swagger UI, 17

U

UI (User Interface), Swagger UI, 17–19
adding people to databases, POST method, 17–18
containerized People.WebApp example, 44–45
GET method, accessing, 17
Try It Out button, 17
uninstalling apps, Helm, 189–190
updating
apps, 127–131, 159
deployments, AKS, 153–157
images, 129–131
People.WebApp example, 105, 110–112, 127–131
workflows, running, 157–158
upgrading apps, Helm, 188–189
usage limits, 173–176
users (Web API), retrieving lists of, 53–54

V

vCore, 97
viewing
container identifiers, 32–33
ingresses in clusters, 116–117
running apps, 14
services in clusters, 116–117
Visual Studio Code
Command Palette, 38
Dockerfiles, building, 38
GitHub Action extension, workflow statuses, 151–152
installing, 9
repositories, building Secrets, 149
VM (Virtual Machines)
containers vs.23–24
declarative configurations, 21–22
volumes
attaching to containers, 33
building
for Pods, 117
with Docker, 32
listing contents of, 33
name conflicts, 33
persistent volume claims, 117
removing, 36
showing with Docker CLI command, 32
Volumes tab, Docker Desktop, 27

W

web API (Application Programming Interfaces)
accessing, 15
data retrieval, web browser, 16
People.WebApp example app
accessing web API, 15
retrieving data, 16
Swagger, exposing OpenAPI files, 16–17
Swagger UI, 17–19
retrieving lists of users, 53–54
testing methods, 19
workflows
automated image builds, 141–143
declaration files, 142–143
Helm and CI/CD pipeline integration, 201–203
modern app development and deployment, overview, 4–6
modern app workflows, 55
testing, 151–152, 201–203
triggering with GitHub, 151
updating, 157–158
workspaces, adding Docker Compose files to, 47