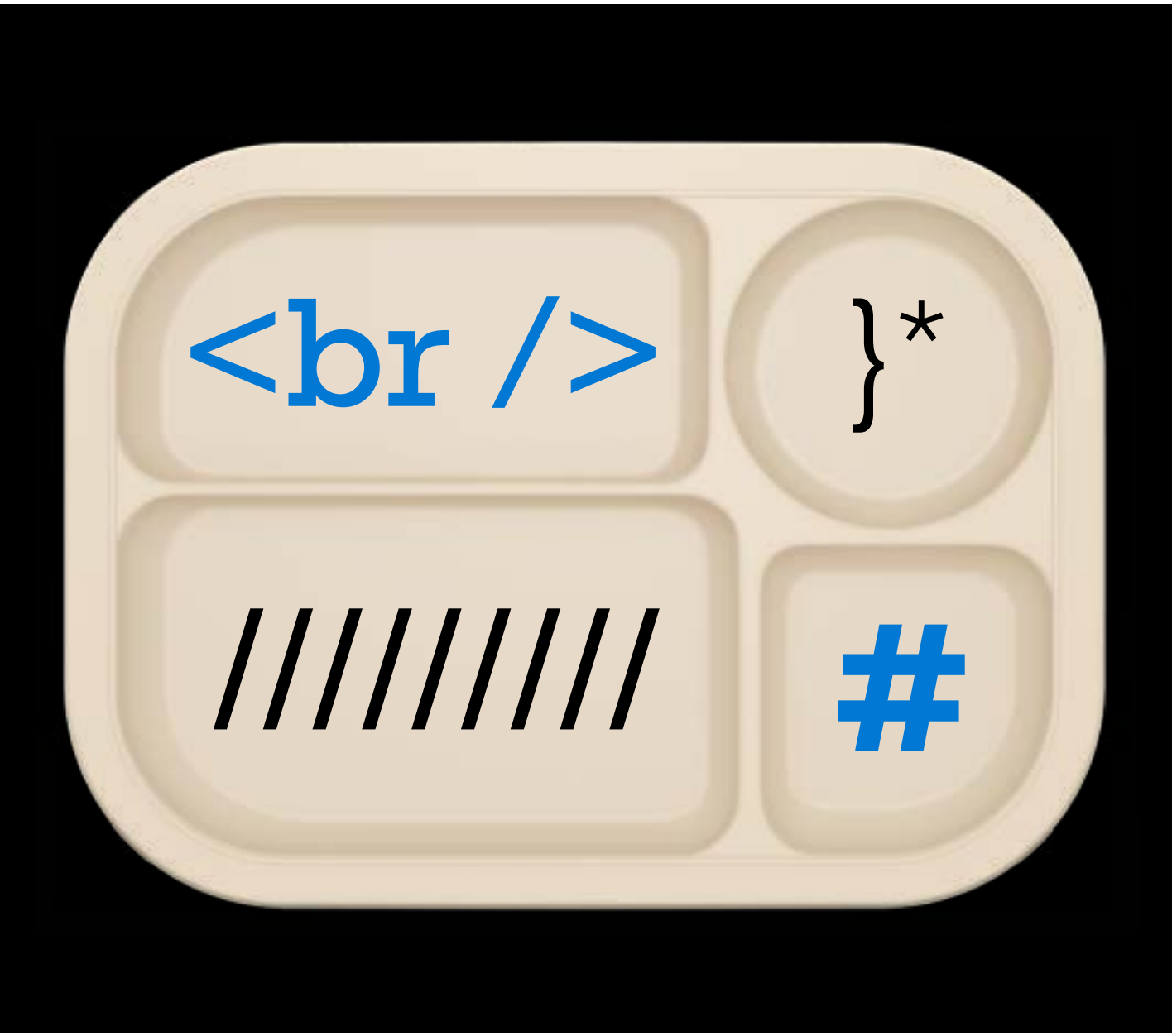


LEARN AZURE IN A MONTH OF LUNCHES

Second Edition
Iain Foulds



www.dbooks.org

Microsoft Azure

Get help with your project.

Talk to a sales specialist >

startHere(Azure);

Dig into 21 Azure lessons

Get a solid foundation in Azure with the lessons in this e book. Sign up for an Azure free account and use your $200 credit to complete the exercises. Continue with your account and get 12 months of popular free services and 25+ always free services.

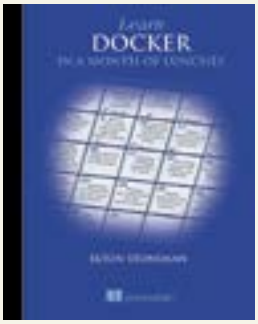
Start free

SAVE 40% ON BOOKS AND VIDEOS FROM MANNING!

Manning publishes high-quality books and videos for technology professionals like you. Use this ***special discount code to save 40% on all eBooks, pBooks, MEAPs, and liveVideo courses at manning.com****, including these selected titles. Just enter* ***azuremsft2*** *in the Promotional Code box when you check out.*



Learn Windows PowerShell in a Month of Lunches
by Don Jones and Jeffery Hicks
December 2016, 384 pages



Learn Docker in a Month of Lunches
Elton Stoneman
Summer 2020, 530 pages

More In A Month of Lunches books

Learn Windows PowerShell in a Month of Lunches, Third Edition
Learn Docker in a Month of Lunches
Learn dbatools in a Month of Lunches
Learn PowerShell in a Month of Lunches, Linux and macOS Edition
Learn PowerShell Scripting in a Month of Lunches
Learn Linux in a Month of Lunches
Learn Amazon Web Services in a Month of Lunches
Learn Cisco Network Administration in a Month of Lunches

Books for Microsoft developers and IT Pros

Azure Data Engineering
Dependency Injection Principles, Practices, and Patterns
Microservices in .NET Core
.NET Core in Action
Microservices Security in Action
Concurrency in .NET
Reactive Applications with Akka.NET
ASP.NET Core in Action
Entity Framework Core in Action
C# in Depth, Fourth Edition
Functional Programming in C#
Kubernetes in Action
Knative in Action
Bootstrapping Microservices with Docker, Kubernetes, and Terraform
Core Kubernetes
GitOps and Kubernetes
Docker in Action, Second Edition
Docker in Practice, Second Edition
Docker in Motion
OpenShift in Action
Cloud Native Patterns

Read Manning books FREE on liveBook

Manning's liveBook platform offers a comfortable, flexible online reading experience. You get ***FREE full access for five minutes a day*** *to each Manning book. In liveBook you can*

- *Ask questions, share code and examples, and interact with other readers in the liveBook forum.*
- *Full-text search across all Manning books—even books you don't own.*
- *Register for a FREE liveBook account at livebook.manning.com.*

You can use your FREE five minutes any way you want—start and stop the timer, hop between books, and try out the interactive exercises. Just log in and ***explore risk free****.*

Praise for the first edition

From the first edition of *Learn Azure in a Month of Lunches* by Iain Foulds:

"An incredible information-packed book to learn core and advanced Azure concepts in a month!"
—Sushil Sharma, Galvanize

"Microsoft Azure is fast becoming a leader in the public cloud space. Following the exercises in this book will quickly get you up to speed with this technology."
—Michael Bright, Developer Advocate, freelance consultant

"Excellent introduction to Azure with many hands on examples. Covers a broad range of current topics."
—Sven Stumpf, ING-DiBa AG

"Azure is like an ocean. This book keeps you afloat by providing the best way to learn in a form of lunches rich in practice and examples."
—Roman Levchenko, Microsoft MVP

"All a busy developer needs to get running on Azure."
—Rob Loranger, freelance developer

"A great way to understand the breadth of Azure offerings, by following a concise, activity focused approach."
—Dave Corun, Avanade

"The most comprehensive book on Azure I found to start developing my academic projects!"
—Marco Giuseppe Salafia, PhD student, Università degli Studi di Catania

"This is the go-to book for the Azure platform. It is well organized, thorough, and comprehensive. Starting with the basics, it guides the reader through creating increasingly complex configurations with the Azure platform to provide scalability, high performance, and redundancy for hosted applications and services. This book will serve both as a tutorial for the beginner and a reference for the more experienced user."
—Robert Walsh, Excalibur Solutions

www.dbooks.org

Learn Azure in a Month of Lunches

SECOND EDITION

IAIN FOULDS



MANNING
SHELTER ISLAND

For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact

Special Sales Department
Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964
Email: orders@manning.com

©2020 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

♾ Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15 percent recycled and processed without the use of elemental chlorine.



Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964

Acquisitions editor:	Mike Stephens
Development editor:	Frances Lefkowitz
Technical development editor:	Karsten Strøbaek
Review editor:	Aleksandar Dragosavljevic
Production editor:	Anthony Calcara
Graphics editor:	Jennifer Houle
Copy editor:	Kathy Simpson
Proofreader:	Katie Tennant
Technical proofreader:	Karsten Strøbaek
Typesetter:	Marija Tudor
Cover designer:	Leslie Haimes

ISBN 9781617297625
Printed in the United States of America

www.dbooks.org

To the ABCs of my life:
Abigail, Bethany, and Charlotte

www.dbooks.org

contents

preface xv
acknowledgments xvi
about this book xvii
about the author xxi

PART 1 AZURE CORE SERVICES 1

1 ***Before you begin*** 3

1.1 Is this book for you? 3

1.2 How to use this book 4

The main chapters 4 ▪ *Try it now* 5 ▪ *Hands-on labs* 5
Source code and supplementary materials 5

1.3 Creating your lab environment 5

Creating a free Azure account 5 ▪ *Bonus lab exercise: Creating a free GitHub account* 7

1.4 A little helping hand 7

1.5 Understanding the Azure platform 8

Virtualization in Azure 10 ▪ *Management tools* 11

2 ***Creating a virtual machine*** 14

2.1 Virtual machine configuration basics 15

Regions and availability options 15 ▪ *VM images* 16
VM sizes 17 ▪ *Azure storage* 18 ▪ *Virtual networking* 19

2.2 Creating an SSH key pair for authentication 20
2.3 Creating a VM from your web browser 22
2.4 Connecting to the VM and installing the web server 24
Connecting to the VM with SSH 24 ▪ Installing the web server 26
2.5 Allowing web traffic to reach the VM 27
Creating a rule to allow web traffic 28 ▪ Viewing the web server in action 28
2.6 Lab: Creating a Windows VM 29
2.7 Cleaning up resources 30
2.8 Houston, we have a problem 31

3 *Azure Web Apps 33*
3.1 Azure Web Apps overview and concepts 34
Supported languages and environments 34 ▪ Staging different versions with deployment slots 35 ▪ App service plans 35
3.2 Creating a web app 37
Creating a basic web app 37 ▪ Deploying a sample HTML site 39
3.3 Viewing diagnostic logs 42
3.4 Lab: Creating and using a deployment slot 44

4 *Introduction to Azure Storage 47*
4.1 Managed Disks 47
OS disks 48 ▪ Temporary disks and data disks 49 Disk-caching options 50
4.2 Adding disks to a VM 50
4.3 Azure Storage 52
Table storage 53 ▪ Queue storage 55 ▪ Storage availability and redundancy 56
4.4 Lab: Exploring Azure Storage 57
VM-focused 57 ▪ Developer-focused 57

5 *Azure Networking basics 58*
5.1 Virtual network components 58
Virtual networks and subnets 59 ▪ Virtual network interface cards 61 ▪ Public IP address and DNS resolution 62

www.dbooks.org

5.2 Securing and controlling traffic with network security groups 64

Creating a network security group 64 ▪ Associating a network security group with a subnet 66 ▪ Creating network security group filtering rules 67

5.3 Building a sample web application with secure traffic 68

Creating remote access network connections 68 ▪ Creating VMs 69 ▪ Using the SSH agent to connect to your VMs 70

5.4 Lab: Installing and testing the LAMP web server 72

PART 2 HIGH AVAILABILITY AND SCALE 73

6 *Azure Resource Manager 75*

6.1 The Azure Resource Manager approach 75

Designing around the application lifecycle 76 ▪ Securing and controlling resources 78 ▪ Protecting resources with locks 79 Managing and grouping resources with tags 80

6.2 Azure Resource Manager templates 81

Creating and using templates 82 ▪ Creating multiples of a resource type 84 ▪ Tools to build your own templates 85 ▪ Storing and using templates 87

6.3 Lab: Deploying Azure resources from a template 87

7 *High availability and redundancy 90*

7.1 The need for redundancy 90

7.2 Infrastructure redundancy with Availability Zones 92

Creating network resources across an Availability Zone 94 Creating VMs in an Availability Zone 95

7.3 VM redundancy with Availability Sets 96

Fault domains 96 ▪ Update domains 97 ▪ Distributing VMs across an Availability Set 98 ▪ View distribution of VMs across an Availability Set 101

7.4 Lab: Deploying highly available VMs from a template 102

8 *Load-balancing applications 106*

8.1 Azure load-balancer components 106

Creating a frontend IP pool 108 ▪ Creating and configuring health probes 110 ▪ Defining traffic distribution with

load-balancer rules 112 ▪ Routing direct traffic with Network Address Translation rules 114 ▪ Assigning groups of VMs to backend pools 116

8.2 Creating and configuring VMs with the load balancer 119

8.3 Lab: Viewing templates of existing deployments 122

9 Applications that scale 124

9.1 Why build scalable, reliable applications? 124

Scaling VMs vertically 125 ▪ Scaling web apps vertically 127 Scaling resources horizontally 128

9.2 Virtual machine scale sets 129

Creating a virtual machine scale set 131 ▪ Creating autoscale rules 133

9.3 Scaling a web app 136

9.4 Lab: Installing applications on your scale set or web app 139

Virtual machine scale sets 139 ▪ Web apps 140

10 Global databases with Cosmos DB 141

10.1 What is Cosmos DB? 141

Structured (SQL) databases 142 ▪ Unstructured (NoSQL) databases 142 ▪ Scaling databases 143 ▪ Bringing it all together with Cosmos DB 144

10.2 Creating a Cosmos DB account and database 145

Creating and populating a Cosmos DB database 145 Adding global redundancy to a Cosmos DB database 149

10.3 Accessing globally distributed data 152

10.4 Lab: Deploying a web app that uses Cosmos DB 156

11 Managing network traffic and routing 158

11.1 What is Azure DNS? 158

11.2 Delegating a real domain to Azure DNS 160

11.3 Global routing and resolution with Traffic Manager 162

Creating Traffic Manager profiles 164 ▪ Globally distributing traffic to the closest instance 167

11.4 Lab: Deploying web apps to see Traffic Manager in action 174

www.dbooks.org

12 *Monitoring and troubleshooting 175*

12.1 VM boot diagnostics 175

12.2 Performance metrics and alerts 178

Viewing performance metrics with the VM diagnostics extension 178 ▪ *Creating alerts for performance conditions 181*

12.3 Azure Network Watcher 182

Verifying IP flows 183 ▪ *Viewing effective NSG rules 184* *Capturing network packets 186*

12.4 Lab: Creating performance alerts 188

PART 3 SECURE BY DEFAULT 189

13 *Backup, recovery, and replication 191*

13.1 Azure Backup 191

Policies and retention 193 ▪ *Backup schedules 196* *Restoring a VM 198*

13.2 Azure Site Recovery 201

13.3 Lab: Configuring a VM for Site Recovery 204

14 *Data encryption 206*

14.1 What is data encryption? 206

14.2 Encryption at rest 208

14.3 Storage Service Encryption 209

14.4 VM encryption 211

Storing encryption keys in Azure Key Vault 211 ▪ *Encrypting an Azure VM 213*

14.5 Lab: Encrypting a VM 214

15 *Securing information with Azure Key Vault 216*

15.1 Securing information in the cloud 216

Software vaults and hardware security modules 217 *Creating a key vault and secret 219*

15.2 Managed identities for Azure resources 221

15.3 Obtaining a secret from within a VM with managed identity 224

15.4 Creating and injecting certificates 229

15.5 Lab: Configuring a secure web server 232

16 *Azure Security Center and updates* 234

16.1 Azure Security Center 234

16.2 Just-in-time access 237

16.3 Azure Update Management 241

Combined Azure management services 243 ▪ *Reviewing and applying updates* 245

16.4 Lab: Enabling JIT and updates for a Windows VM 249

PART 4 THE COOL STUFF 251

17 *Machine learning and artificial intelligence* 253

17.1 Overview and relationship of AI and ML 254

Artificial intelligence 254 ▪ *Machine learning* 255 *Bringing AI and ML together* 256 ▪ *Azure ML tools for data scientists* 257

17.2 Azure Cognitive Services 259

17.3 Building an intelligent bot to help with pizza orders 260

Creating an Azure web app bot 260 ▪ *Language and understanding intent with LUIS* 261 ▪ *Building and running a web app bot with LUIS* 264

17.4 Lab: Adding channels for bot communication 267

18 *Azure Automation* 269

18.1 What is Azure Automation? 269

Creating an Azure Automation account 271 ▪ *Azure Automation assets and runbooks* 272

18.2 Azure Automation sample runbook 274

Running and viewing output from a sample runbook 276

18.3 PowerShell Desired State Configuration (DSC) 278

Defining and using PowerShell DSC and an Azure Automation pull server 280

18.4 Lab: Using DSC with Linux 282

19 *Azure containers* 284

19.1 What are containers? 284

19.2 The microservices approach to applications 287

19.3 Azure Container Instances 289

www.dbooks.org

19.4 Azure Kubernetes Service 293
Creating a cluster with Azure Kubernetes Services 294
Running a basic website in Kubernetes 295

19.5 Lab: Scaling your Kubernetes deployments 298

20 *Azure and the Internet of Things 300*

20.1 What is the Internet of Things? 300

20.2 Centrally managing devices with Azure IoT Hub 303

20.3 Creating a simulated Raspberry Pi device 306

20.4 Streaming Azure IoT hub data into Azure web apps 309

20.5 Azure IoT component review 315

20.6 Lab: Exploring use cases for IoT 316

21 *Serverless computing 317*

21.1 What is serverless computing? 317

21.2 Azure messaging platforms 319
Azure Event Grid 320 ▪ *Azure Event Hubs and Service Bus 321*
Creating a service bus and integrating it with an IoT hub 322

21.3 Creating an Azure logic app 325

21.4 Creating an Azure function app to analyze IoT device data 328

21.5 Don't stop learning 332
Additional learning materials 333 ▪ *GitHub resources 333*
One final thought 333

index 335

www.dbooks.org

preface

This second edition of *Learn Azure in a Month of Lunches* reminds me that things change quickly and that you must always keep learning. Gone are the days when you could take a week-long course in Windows Server and comfortably run it for years without changing much. This doesn't mean that the IT world is a scarier place, but you do need to approach cloud computing with an open mind and be willing to adjust constantly.

When I started to work with Azure, the number of available services was almost overwhelming. I knew that I should pay attention to security, performance, redundancy, and scale, but I didn't know how to adapt more than a decade of large-scale server administration to the world of cloud computing. Over time, I began to learn about the various Azure services that provide those key components. These services rarely work in isolation, but I didn't know the best way to integrate them or how to decide which service to use for each task. This book is a way to explain to my past self, and to many others following a similar path, how to quickly understand the core services in Azure and make them work together.

This book is more than 350 pages long, yet it barely scratches the surface of what you can do in Azure! To help give you a solid understanding of the concepts needed to be successful as you build solutions in Azure, I had to choose which topics to write about. The book doesn't cover all 100 or more Azure services, and it doesn't go into exhaustive detail about the services that are included. Instead, it focuses on the core areas of some of the primary services, and shows examples of how to connect everything securely, and introduces the possibilities of what you can build in Azure.

Cloud computing is continually changing. There are no three- or four-year release cycles and large update deployments. I think that today is a great time to be building solutions and writing code; there's always an opportunity to learn something new and improve yourself. I hope that you learn to run great applications in Azure and enjoy exploring all the available services.

acknowledgments

Many people behind the scenes at Manning Publications helped publish this book. Special thanks go to Mike Stephens for having the vision to get this project started. I thank my publisher, Marjan Bace, and everyone on the editorial and production teams. My thanks go to the technical peer reviewers led by Aleksandar Dragosavljević—Ariel Gamino, Charles Lam, Ernesto Cardenas Cangahuala, George Onofrei, Glen Thompson, Jose Apablaza, Juraj Borza, Michael Langdon, Michael Wall, Peter Kreyenhop, Rick Oller, Rob Ruetsch, Robert Walsh, and Vishal Singh. And finally, on the technical side, thanks go to Karsten Strøbaek, who served as both the book's technical editor and proofreader.

For this second edition, I offer a big thanks to Phil Evans and Davanand Bahall for their support and the freedom to update this book. This was an after-hours project outside my Microsoft day job, but a lot of people were excited and affected by it. Continued thanks go to David Tolkov and Tim Teebken, who gave me opportunities to develop into someone capable of writing this book. And look, Jean-Paul Connock, we've won a Stanley Cup since the last time! Go Blues!

Thanks go to Rick Claus for supporting the need for strong technical documentation in Azure, and to Marsh Macy and Neil Peterson for their personal support and guidance in writing the original version of this book. We still need to get started on that school bus.

www.dbooks.org

about this book

This book is designed to give you a solid foundation for success as an IT engineer or developer in Azure. You'll learn about both Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) solutions, as well as when to use each approach. As you work through the chapters, you'll learn how to plan appropriately for availability and scale, keep security in mind, and consider cost and performance. By the end of the book, you should be able to integrate upcoming technologies such as containers and Kubernetes, artificial intelligence and machine learning (AI + ML), and the Internet of Things (IoT).

When it comes to building and running your applications and services, Azure lets you choose the operating system, application tools, and platform you're most comfortable with. This book mostly discusses non-Microsoft technologies such as Linux, Python, and Node.js. Command examples use the Azure CLI, not Azure PowerShell. These were conscious decisions to show you that using Azure doesn't mean you have to use Windows Server, IIS, or ASP.NET.

As you work in the cloud, you often work across platforms and must learn new topics, which is another reason for showing non-Microsoft technologies and platforms. I wanted to introduce you to some of these new areas before you encounter them in the real world. Throughout the book, I'll teach you the concepts and steps needed to integrate Azure services, so you can switch platforms or languages as you wish and have the same knowledge apply.

Roadmap

The book is organized into 4 parts and 21 chapters:

- Part 1 covers some of the core Azure infrastructure and platform services: virtual machines, web apps, storage, and networking.

- Part 2 dives into how to provide high availability and redundancy: templates, availability sets and zones, load balancers, autoscaling, distributed databases, and traffic routing. By the end of chapter 12, you should have a solid knowledge of how to build high-performance distributed applications in Azure.
- Part 3 covers security aspects such as backup and recovery, encryption, digital key management, and updates. By the time you've completed chapter 16, you'll be well on the way to creating secure, stable applications in Azure.
- To finish the book, part 4 introduces a little fun, exploring new areas of computing such as serverless computing and container-based applications. These chapters introduce areas of Azure that give you a glimpse of what the future of production applications could look like.

Except in part 4 (which is aptly named "The cool stuff"), you should try to work through the book's chapters in order. You won't work on the same project over successive chapters, but each chapter builds on earlier theory and hands-on lab examples.

Chapter 1 guides you through creating a free trial account in Azure, which is enough to complete the hands-on lab exercises in each chapter. I also provide a little more background on Azure and how to find additional help along the way. I mention this web page a few times in the book (maybe I'm a little biased!), but http://docs.microsoft.com/azure is the best place to go for additional documentation and support on any areas of Azure that interest you.

About the examples and source code

This book contains many examples of source code, both in numbered listings and inline with normal text. In both cases, source code is formatted in a `fixed-width font like this` to separate it from ordinary text.

In many cases, the original source code has been reformatted, with line breaks and reworked indentation added to accommodate the available page space in the book. In rare cases, even this was not enough, and listings include line-continuation markers (➥). Additionally, comments in the source code are removed from the listings when the code is described in the text. Code annotations accompany many of the listings, highlighting important concepts.

This book's source code, along with accompanying scripts, templates, and supporting resources, is available at https://www.manning.com/books/learn-azure-in-a-month-of-lunches-second-edition and at the book's GitHub repo (https://github.com/fouldsy/azure-mol-samples-2nd-ed).

All the hands-on exercises can be completed in the Azure portal and with Azure Cloud Shell, a browser-based interactive shell for both the Azure CLI and Azure PowerShell. There are no tools to install on your machine, and you can use any computer and OS you wish, provided that it supports a modern web browser.

The Azure portal often implements minor changes. Part of the challenge of using any cloud service is that things may be a little different than they were the day before.

www.dbooks.org

This second edition of the book tries to minimize the number of portal screenshots, but don't worry if what you see is still a little different from what's shown in the book. The required parameters are usually the same; the layout may just be different. If there are new options in the portal that I don't specifically call out in an exercise or lab, it's usually safe to accept the defaults that are provided.

If you work outside Azure Cloud Shell, take care with the command examples. Windows-based shells such as PowerShell and CMD treat line breaks and continuations differently from *nix-based shells such as Azure Cloud Shell. Many of the command examples run across multiple lines. Commands are shown with a backslash (\) character to indicate that the command continues on the next line, as in the following example:

```
az resource group create \
--name azuremol \
--location eastus
```

You don't have to type those backslash characters, but doing so may make long commands more readable on your screen. If you choose to work locally on your computer with a Windows shell, you can use a backtick (`) instead of a backslash. In a PowerShell or CMD shell with Python for Windows installed, for example, change the previous command as follows:

```
az resource group create `
--name azuremol `
--location eastus
```

This convention may seem to be confusing at first, but I follow it in the book because the official documentation at https://docs.microsoft.com/azure uses this format. Azure CLI commands, which are mostly used in this book, assume a *nix-based shell and therefore use a backslash character. Azure PowerShell commands assume a Windowsbased shell and so use a backtick. This difference in behavior will make sense quickly, and you'll find that it's easy to transition between shells. If you're new to working across platforms, this difference can be a fun little gotcha!

I recommend that you check out the Windows Subsystem for Linux (WSL) if you run Windows 10 and want to dive into the Azure CLI and *nix-based systems in general; you can find details at https://docs.microsoft.com/windows/wsl. WSL, and the latest improvements in WSL2 give you a native Linux kernel experience while running Windows. Don't try to wrap your head around that idea too much! Just know that you can run native Linux commands and applications without worrying about different line breaks or variable definitions. To really blow your mind, PowerShell is available for .NET Core, which also runs on Linux. You can run PowerShell on Linux while in Windows.

liveBook discussion forum

Purchase of *Learn Azure in a Month of Lunches* includes free access to a web forum run by Manning Publications where you can make comments about the book, ask technical

questions, and receive help from the author and from other users. To access the forum, go to https://livebook.manning.com/book/learn-azure-in-a-month-of-lunches-second-edition/discussion. You can also learn more about Manning's forums and the rules of conduct at https://livebook.manning.com/discussion.

Manning's commitment to our readers is to provide a venue where a meaningful dialogue between individual readers and between readers and the author can take place. It is not a commitment to any specific amount of participation on the part of the author, whose contribution to the forum remains voluntary (and unpaid). We suggest that you try asking him some challenging questions, lest his interest stray! The forum and the archives of previous discussions will be accessible from the publisher's website as long as the book is in print.

www.dbooks.org

about the author

IAIN FOULDS is a senior content developer at Microsoft, currently writing technical documentation for Azure Active Directory. Previously, he was a premier field engineer with Microsoft for virtualization technologies such as Azure, Hyper-V, and System Center Virtual Machine Manager. With more than 15 years of experience in IT, most of it in operations and services, he embraced virtualization early with VMware and has helped build and teach others about cloud computing for years.

Originally from England, he has lived in the United States for more than a decade and currently resides just outside Seattle with his wife and two young children, to whom this book is dedicated. He is a fan of football (unfortunately called "soccer" where he lives) and also enjoys ice hockey and almost any form of motor racing. Outside computing, his interests include performance and classic cars, aviation photography, and claiming to play the guitar. He's also a big model-train nerd, regularly attending and volunteering at shows and events throughout the Pacific Northwest.

www.dbooks.org

Part 1

Azure core services

To build the next great application, you need a solid understanding of the basic resources in Azure. Things like storage and networking may not be the coolest things to look at, but they're central to much that you run in Azure. Before you can start to get into redundant, multi-instance virtual machines or Azure web apps, it helps to see the available options and management tasks for a single instance. This approach lets you learn about the differences and similarities between the IaaS approach of VMs and the PaaS approach of web apps. Chapters 1–5 explore VMs, web apps, core storage, and virtual networking features.

www.dbooks.org

1 Before you begin

Azure is one of the largest public cloud computing providers for services such as virtual machines (VMs), containers, serverless computing, and machine learning. We won't dive into all 100 or more Azure services in this book, but you're going to learn about the core services and features that cover most of what you need to start building and running solutions in Azure. We'll look at a common example of how to build and run a web application, and you'll see how to use some of the core infrastructure and platform services that can make your job easier.

With Azure, you don't need a magic wand to predict how many servers or how much storage you'll need over the next three years. No more delays as you gain budget approval; wait for new hardware to ship; and then rack, install, and configure everything. You don't need to worry about what software versions or libraries are installed as you write your code.

Instead, select a button, and create whatever resources are needed. You pay only for each minute those resources are running or for the amount of storage space or network bandwidth used. When you don't need the resources anymore, you can power down or delete them. If you suddenly need to increase the amount of compute power by a factor of 10, select a button, wait a couple of minutes, and it's there. And all this is managed by someone else, freeing you to focus on your applications and customers.

1.1 Is this book for you?

The IT industry is in a bit of a transition period when it comes to job titles. You may refer to yourself as an IT pro, a software developer, a system administrator, or a DevOps engineer. Regardless of what you call yourself, if you want to learn the core

skills needed to build and run secure, highly available applications in the cloud, you're in the right place. In generic terms, you probably fall on the IT operations or development side of things. The truth is, there's a lot of crossover, especially in cloud computing. Whether you're in development or operations, it's important to understand the core infrastructure and platform services to build and run applications that best serve your customers.

This second edition of the book introduces some of these core concepts in Azure and teaches you the skills you need to make informed decisions. Coming into this book, you should have some experience with VMs and know the basics of networking and storage. You should also be able to create a basic website, as well as understand what an SSL certificate and a database are. After we cover the core processes, we'll take a quick look at new and upcoming technologies. You want to stay ahead of where your job may take you, so you'll learn about containers, the Internet of Things, machine learning, artificial intelligence, and serverless computing. Both self-described developers and IT pros should find some neat new areas to learn about!

1.2 How to use this book

I like sandwiches, so lunch is a great time for me to play with cool new technology. You may be a night owl who has some extra time in the evening, or you may be an early-morning person (what's wrong with you?!) who can work through a chapter at breakfast. There's no right or wrong time to learn, but if you can set aside about 45 minutes, you should be able to read a chapter and complete its exercises. Each chapter covers something new, so give yourself time to absorb each day's lesson.

1.2.1 The main chapters

The book is broken into four parts, which is convenient if you believe that there are four weeks in a month:

- Part 1 (chapters 1–5) covers some of the core Azure resources. If nothing else, try to follow these chapters in order to have a solid understanding. Then you can focus on the other chapters that most excite you.
- Part 2 (chapters 6–12) covers availability and scale. You'll learn how to automatically scale resources in and out, load-balance traffic, and handle maintenance events without downtime. To learn about running highly available applications on a global scale, this part is for you.
- Part 3 (chapters 13–16) is for the security geeks. It covers things like how to encrypt VMs, store SSL certificates in a secure vault, and back up and restore your data.
- Part 4 (chapters 17–21) covers a mix of cool areas to give you a taste of what Azure can do for you and your customers. We'll look at automation, containers, the Internet of Things, and serverless computing. Pick something that interests you, and have fun!

www.dbooks.org

1.2.2 Try it now

Do you just want to read, or do you want to roll up your sleeves and play with Azure? Throughout the book are little tasks that let you quickly try something new. If you have the time, try them. Most of the hands-on time comes in a lab exercise at the end of the chapter, but there's a lot of value in breaking up the reading by trying new concepts along the way. Some of these exercises guide you through things step by step; others are going to make you think a little harder and learn how to build solutions by yourself like you would in the real world.

1.2.3 Hands-on labs

Each chapter wraps up with a hands-on lab exercise. Some chapters, like this one, have a lab exercise in the middle of the chapter. These lab exercises are where you learn how all the pieces of Azure come together and start to build some mental muscle memory. Grab your keyboard and mouse, and begin building something awesome!

1.2.4 Source code and supplementary materials

This book's source code, along with accompanying scripts, templates, and supporting resources, can be found at https://www.manning.com/books/learn-azure-in-a-month-of-lunches-second-edition and on the book's GitHub repo at https://github.com/fouldsy/azure-mol-samples-2nd-ed. In addition, you can participate in the book's forum at https://livebook.manning.com/book/learn-azure-in-a-month-of-lunches-second-edition/discussion.

1.3 Creating your lab environment

This book isn't heavy on concepts and architecture; it's more about hands-on time with the Azure platform. To get this practice, you need an Azure account.

1.3.1 Creating a free Azure account

Azure offers a free trial account that's good for 30 days and provides up to $200 of free credit. This credit should be enough for you to make it through all the chapters and exercises, with room to explore a little and have fun along the way. Many Azure services and features remain free even after your trial period ends.

> **Try it now**
>
> Follow along with the steps in this section to create your free Azure account:

1. Open your web browser to https://azure.microsoft.com/free, and choose the option to get started with a free Azure account.
2. When prompted, sign in to your Microsoft account. If you need a Microsoft account or want to create a new one, choose the Create a New Microsoft Account link.

3. When you're signed in to a Microsoft account, complete the prompts to create a free Azure account:
   - Enter your personal details as requested.
   - To help minimize abuse and fraud, provide a phone number to verify your identity by text message or phone call.
   - A credit card is also required for identity verification, but there's no catch here. Your account doesn't start billing you until after 30 days or when you use up your $200 credit. You won't automatically transition to a pay-as-you-go subscription at the end of your trial. You may see a small $1 (or equivalent local currency) verification hold that's refunded in a few days.
4. Review and accept the Azure subscription agreement and privacy policy, and then select Sign Up. It may take a few minutes to get your Azure subscription ready.
5. When the sign-up process finishes, and the Azure portal loads, take the quick tour to learn how to move around.

Your dashboard—the home page of the portal—looks empty right now. But in chapter 2, you'll dive into creating your first VM, and it will start to look like figure 1.1!

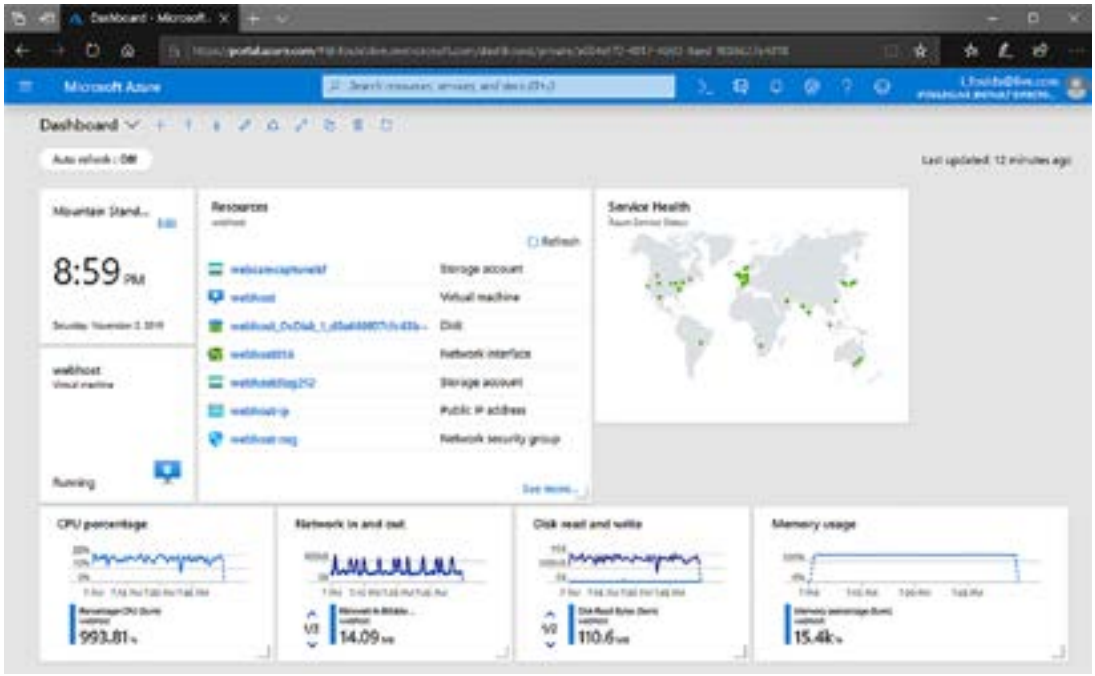


Figure 1.1 The Azure portal, ready for you to create your own applications and solutions

www.dbooks.org

Is free truly free?

Azure has a Marketplace that contains hundreds of prebuilt images (the basis of VMs) and solutions you can deploy. We'll use some of these Marketplace offerings throughout the book; they're great ways to deploy an entire application suite quickly.

Not all of these Azure Marketplace offerings are free; some third-party publishers combine licensing or support costs into the solution you deploy. A VM that you deploy from Red Hat, for example, may incur an additional fee that covers the Red Hat support agreement and license. These charges aren't covered by your free trial credit; only the base VM use is covered.

The exercises in this book use only resources that remain within the free trial. But if you go off exploring other cool Marketplace offerings in Azure, pay attention to what you build. Any solution that includes additional fees should clearly spell those fees out before you deploy!

1.3.2 Bonus lab exercise: Creating a free GitHub account

GitHub is a free web service that many organizations and individuals use to manage projects such as code, templates, and documentation. Azure has hundreds of free templates and script examples that you can use and contribute to. This is one of the strengths of the open source community—sharing and giving back to others.

Some of the exercises in this book use resources from GitHub. You don't need a GitHub account to do any of this, but if you don't have an account, you won't be able to save any modifications and begin to build your own collection of templates and scripts. Creating a GitHub account is an optional, but highly recommended, part of building your lab environment:

1. Open your web browser to https://github.com.
2. To create a free GitHub account, provide a username, email address, and password. You'll receive a validation message from GitHub.
3. Select the link in the validation email to activate your account.
4. Check out some of the Azure repositories that provide sample resources:
   - Azure quick-start templates—https://github.com/Azure/azure-quickstart-templates
   - Azure CLI—https://github.com/Azure/azure-cli
   - Azure DevOps utilities—https://github.com/Azure/azure-devops-utils
   - *Learn Azure in a Month of Lunches* book resources—https://github.com/fouldsy/azure-mol-samples-2nd-ed

1.4 A little helping hand

This book can't cover everything Azure offers. Even if I tried, by the time you read this chapter, I bet there will be something new in Azure! Cloud computing moves quickly, and new services and features are always being released. I may be a little biased, but as

you start to explore Azure and want to learn about additional services, the excellent https://docs.microsoft.com/azure site is the best place to start. Every Azure service is documented with quick-start examples, tutorials, code samples, developer references, and architecture guides. You can also access both free and paid support options if you need help along the way.

1.5 Understanding the Azure platform

Before you get into the rest of this book, let's take a step back and cover what Azure is and the services that are available. As I mentioned earlier, Azure is a cloud computing provider on a global scale. At the time of writing, there are 54 Azure regions. Each region contains one or more data centers. By comparison, the two other major cloud providers operate in 23 regions (Amazon Web Services [AWS]) and 20 regions (Google Cloud).

Cloud computing provides more than just compute resources. Azure has more than 100 services, grouped in families of related services such as compute, web + mobile, containers, and identity. With all these services, Azure covers many service models. Let's grab a slice of pizza for lunch to understand what this means (figure 1.2).

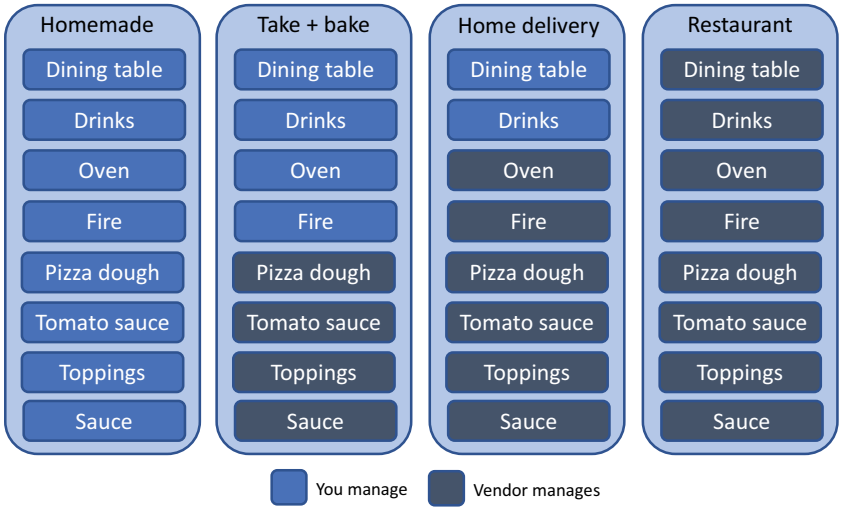


Figure 1.2 Pizza as a Service model. As you move from homemade pizza, where you provide everything, to the restaurant model, where you just show up, the responsibilities and management demands change accordingly.

In the Pizza as a Service model, you have four options to choose among. As you progress through the models, you worry less and less about the process of eating a slice of pizza:

- *Homemade*—You make the dough; add the sauce, toppings, and cheese; bake the pizza in your oven; get drinks; and sit down to eat at your dining table.

www.dbooks.org

- *Take + bake*—You buy a ready-made pizza. You just need to bake it in your oven, get drinks, and sit down to eat at your dining table.
- *Home delivery*—You order a pizza delivered to your home. You just need to get drinks and sit down to eat at your dining table.
- *Restaurant*—You want to go out and eat pizza with minimal effort!

Now that you're hungry, let's look at the more traditional model that involves some compute resources (figure 1.3). This model looks a little more like something you see in Azure.

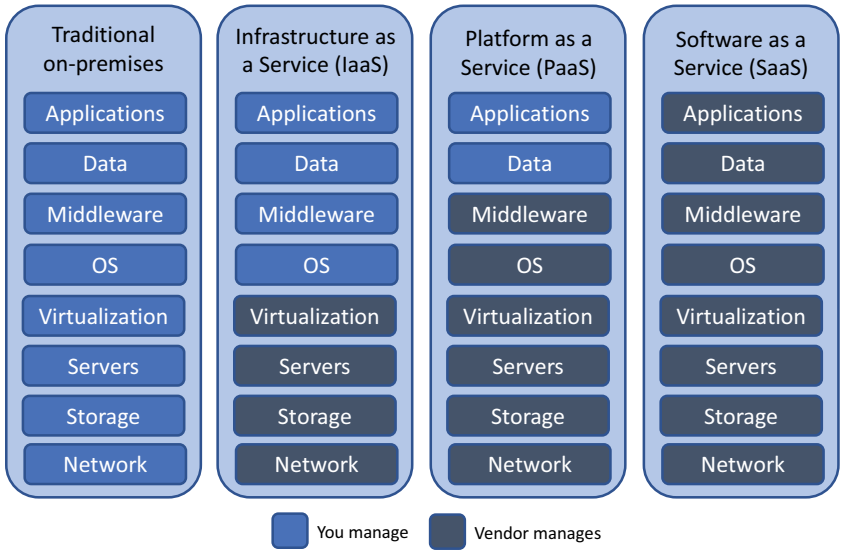


Figure 1.3 Cloud computing service model

As you progress through the models, you manage fewer of the underlying resources and can focus more of your time and energy on your customers:

- *On-premises*—You configure and manage the entire data center, such as the network cables, storage, and servers. You're responsible for all parts of the application environment, support, and redundancy. This approach provides maximum control, but with a lot of management overhead.
- *Infrastructure as a Service (IaaS)*—You purchase the base compute resources from a vendor that manages the core infrastructure. You create and manage the VMs, data, and applications. The cloud provider is responsible for the physical infrastructure, host management, and resiliency. You may still have an infrastructure team to help support and deploy VMs, but the team is free from the time and cost of managing the physical equipment.

This approach is good when you start to move applications out of your own on-premises environment. The management and operations are often similar to an on-premises environment, so IaaS provides a natural progression for the business, IT, and application owners to become comfortable with the cloud.

- *Platform as a Service (PaaS)*—You purchase the underlying platform stack from a vendor that manages the OS and patches, and bring your applications and data. You don't worry about VMs or the virtual network, and your operations team can focus more of their time on application reliability and performance.

  This approach often starts making the IT organization and the business comfortable with running applications in the cloud. Your focus is on the applications and your customers, with fewer worries about the infrastructure needed to run those apps.

- *Software as a Service (SaaS)*—You just need access to software, with a vendor providing everything else. Developers can build against an existing platform to provide customizations or unique features without having to maintain a large code base.

  This approach is often daunting at first, but you likely already know about and use a successful SaaS offering such as Salesforce, Office 365, or the Google suite of Mail or Docs. You use email, create documents or presentations, or manage customer contact information and sales information. Your focus is on the content that you create and manage, not on how to make the application run.

Most of what you create in Azure falls into the IaaS and PaaS areas. The main use cases include VMs and virtual networking (IaaS) or the Azure Web Apps, Functions, and Cosmos DB (PaaS) services. If you're a developer, the PaaS solutions are probably the areas you're most interested in, because Microsoft covers the infrastructure parts to let you focus on your code. IT pros may lean more toward the IaaS solutions to build out and control the Azure infrastructure.

Never stop learning

Don't forget that even as a business moves from IaaS toward the PaaS model, the IT pro remains relevant! It's important to understand what goes on underneath the PaaS layer when you design or troubleshoot a solution. If you're an IT pro, don't skip the chapters on PaaS solutions in Azure; you can add a lot of value for your business and customers if you understand the transition to that deployment model.

1.5.1 Virtualization in Azure

Virtualization is the real magic behind Azure. The IaaS, PaaS, and SaaS models use virtualization to power their services. The concept of virtualization is nothing new, going all the way back to the mainframe days of the 1960s. In the mid-2000s, server virtualization in the data center started to gain momentum, and by now, only a few workloads are deployed to bare-metal servers rather than being virtualized.

www.dbooks.org

Entire books are dedicated to virtualization, but here's a quick overview: virtualization logically divides physical resources in a server into virtual resources that can be securely accessed by individual workloads. A VM is one of the most common resources in cloud computing. A VM contains a virtual CPU (vCPU), memory (vRAM), storage (vDisk), and network connectivity (vNIC), as shown in figure 1.4.

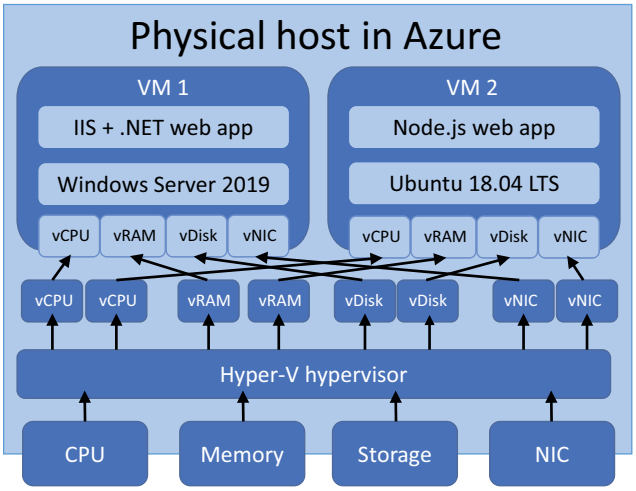


Figure 1.4 Virtualization in action on a physical host in Azure

In addition to physical servers, storage and networking are commonly virtualized, which allows the Azure platform to quickly define everything you need in software. No physical interaction or manual configuration of devices is required. You don't have to wait for another team to provide an IP address, open a network port, or add storage for you.

At its core, Azure runs on Windows—sort of. A modified version of the Hyper-V hypervisor powers the compute servers. Hyper-V is a type 1 (bare-metal) hypervisor that has been available in Windows Server for a decade. And don't worry—you can still run Linux as a fully supported, first-class workload! Microsoft is a huge contributor to the Linux community and kernel. Some of the core software-defined networking in Azure is powered by a custom-built solution based on Debian Linux—Software for Open Networking in the Cloud (SONiC)—that Microsoft has made open source. You can take a virtual tour of Microsoft's data centers at https://azure.microsoft.com/global-infrastructure.

1.5.2 Management tools

With so many Azure services, how do you use them? Any way you want! If you want to select everything in a web browser, there's an awesome web-based portal. Comfortable with PowerShell? As you'd expect, there's an Azure PowerShell module. There's also a

cross-platform command-line interface (CLI) tool that's great if you're on macOS or Linux. And developers can interact with Azure through REST APIs using a variety of common languages such as .NET, Python, and Node.js.

AZURE PORTAL

The Azure portal should work in any modern web browser, and it's a convenient way to use Azure without installing anything on your computer. The portal is also a great way to learn how to create and manage resources by quickly seeing a visual representation of everything.

New features and services are continually being added to Azure, so the portal may change ever so slightly from what you see in the screenshots in this book or online documentation and blogs. The wording on a button may change a little, or a new option may be added, but all the core operations remain the same. Welcome to the brave new world of cloud computing!

AZURE CLOUD SHELL

If you want to get your hands on the keyboard and type commands, the portal also includes the Azure Cloud Shell, shown in figure 1.5. This shell is a web-based interactive console that provides a Bash shell, the Azure CLI, and some preinstalled application development tools such as Git and Maven. There's also a PowerShell version of Cloud Shell that, as the name implies, provides access to the latest Azure PowerShell cmdlets.

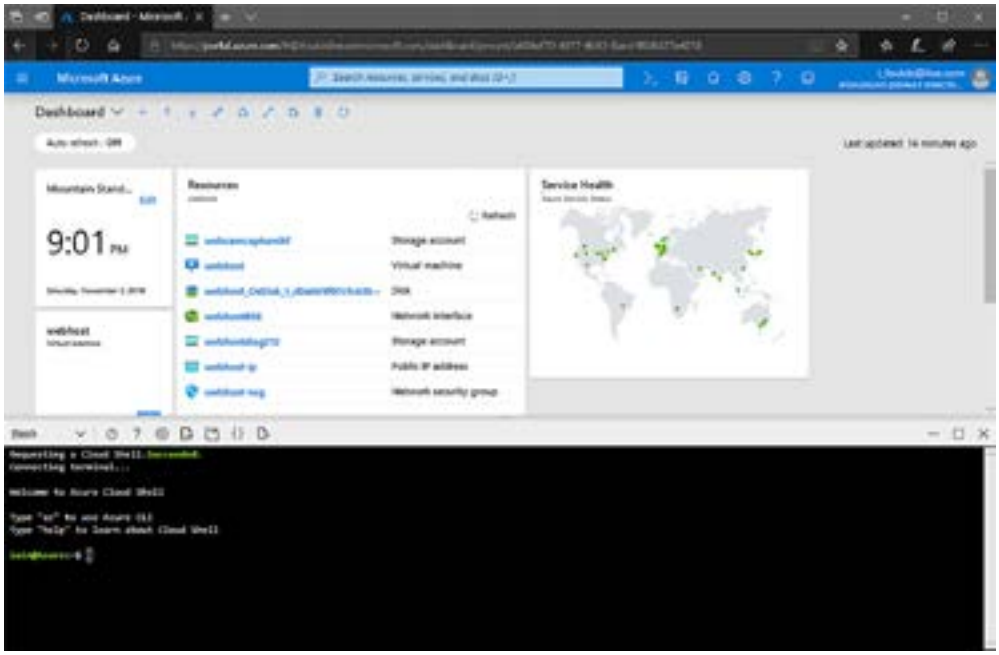


Figure 1.5 The Azure Cloud Shell in the web-based portal

www.dbooks.org

You can access the Azure Cloud Shell from a web browser on any computer without needing to install any tools at https://shell.azure.com. Editors like Visual Studio Code (https://code.visualstudio.com) provide Cloud Shell access within the application. There's even an Azure app available for iOS and Android that allows you to use the Azure Cloud Shell straight from your smartphone.

With the Azure Cloud Shell, you always have access to the latest version of the CLI or PowerShell tools. Persistent storage allows you to create and save scripts, templates, and configuration files.

Local Azure Cli and Powershell Tools

Although there are advantages to the Azure Cloud Shell, you often need access to your local filesystem and tools. You can install the Azure CLI or Azure PowerShell locally so that you can work with local resources and Azure resources.

In this book, we'll mostly use the Azure CLI (technically, the Azure CLI 2.0). It may seem odd to choose it over Microsoft's native PowerShell; the advantage is that both the samples and the exercises can work in the Azure Cloud Shell and locally on your computer, regardless of what OS you use. Although this information isn't part of setting up your lab environment, the following guides detail how to install the Azure management tools on your computer:

- *Getting Started with Azure PowerShell*—https://docs.microsoft.com/powershell/azure/get-started-azureps
- *Install Azure CLI*—https://docs.microsoft.com/cli/azure/install-azure-cli

2 Creating a virtual machine

Ready to see how quickly you can set up a web server in Azure? In this chapter, we'll dive straight into one of the most common requests when it comes to VMs: building a basic web server. This workload is a great example of the core Infrastructure as a Service (IaaS) components in Azure.

Assume that you work for a pizza store that wants to expand its operations and accept online orders for pizza delivery or takeout. To build an online presence, you need a website. In the first couple of parts of this book, we'll explore the different features and services in Azure that let you build and run both IaaS and Platform as a Service (PaaS) web applications. You can start to make informed decisions as to when to build and run a VM to power a website and when you might use PaaS to do so. But the first step is building a web server.

In this chapter, you'll create an Ubuntu Linux VM and install a basic web server. Don't worry about using Linux; you'll create a Windows VM in the end-of-chapter

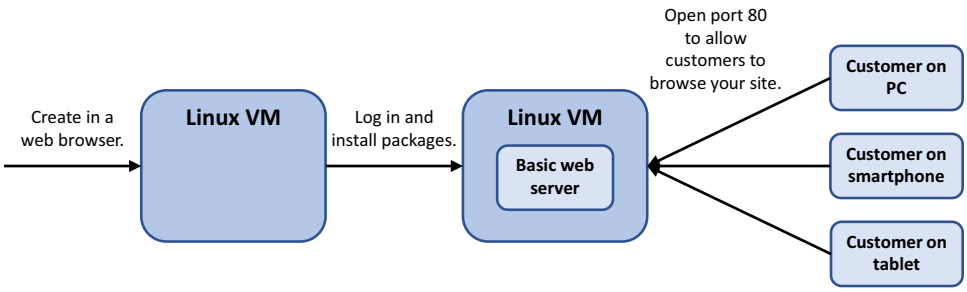


Figure 2.1 In this chapter, you create a basic VM, log in to install a web server, and then open a network port to allow customers to browse to the sample website.

www.dbooks.org

lab exercise! Ubuntu is a common web server platform, and it's a great way to learn about SSH public-key authentication. Then you'll see how to open a network port for customers to access your website on the internet. A high-level overview of this basic environment is shown in figure 2.1.

2.1 Virtual machine configuration basics

VMs are among the most common building blocks you'll use when you start to run applications in the cloud. Why? They're usually familiar territory. Most IT departments run a lot of workloads by using Hyper-V or VMware in an on-premises environment, so you likely already have some experience in building and running VMs. Organizations often take their first steps in Azure with VMs, as IaaS workloads don't require the big mental adjustment that you need to make when you start to run PaaS workloads.

There are solutions to migrate VMs from an on-premises environment such as Hyper-V or VMware into Azure, but before you get too carried away with what's possible in Azure (some of which we'll explore in later chapters), let's look at some basics. These next few pages may seem like familiar considerations and options you have with on-premises VMs. If so, great! If this is new, don't worry; a lot of the management is abstracted in Azure, and things like virtual networks are typically created and configured once and then left alone. We'll go into each area in more depth in the coming chapters, so take a deep breath and go one step at a time.

2.1.1 Regions and availability options

Azure is divided into regions around the world, with each region having one or more data centers. These data centers provide the core compute, storage, and network resources to run your applications and workloads. Azure runs in more than 50 regions, with the list growing every few months. With so many regions, the idea is that you can deploy applications close to your employees or customers. This regional availability reduces latency and improves the end-user experience.

An Azure region may not offer every single service that's available in Azure. With hundreds of services available, the most common set of core services is usually everywhere, but new or niche services typically roll out over time. As you plan your applications in Azure, check out product availability by region at https://azure.microsoft.com/global-infrastructure/services.

In chapter 8, we'll look at some of the high-availability options such as Availability Sets and Availability Zones. These redundancy options let Azure distribute multiple instances of your VMs or applications within a single data center or across an entire region. This ability lets you define your tolerance to maintenance updates or hardware failure. In the early chapters of this book, you'll typically create only one or two VMs, so don't worry about these availability options just yet.

2.1.2 VM images

To create a VM, you need a starting point. Usually, this starting point comes down to the choice of operating system: Windows or Linux. Next comes the choice of what version of Windows to use (such as Windows Server 2016 or 2019) or what Linux distribution to use (such as Ubuntu, Red Hat Enterprise Linux, or SUSE).

An *image*—a preconfigured OS bundle with basic configuration options applied—is that starting point. Azure contains hundreds of these prebuilt images in the Azure Marketplace to use when you create VMs. You can often apply existing Windows licenses, depending on your current licensing model, or can opt in to additional support from Canonical to run Ubuntu Linux or updates from Red Hat, for example.

To keep things simple and short enough that you can complete these lessons in a lunch hour, you'll use these prebuilt images in Azure throughout the book. In the real world, you'll probably want to customize things to fit your business needs and requirements. To do this, you'll often create your own VM images. The workflow to create and manage the VMs is the same as with the Azure Marketplace images, but often, building your own images takes a lot of planning and then hours of configuring, generalizing, and capturing your own images ahead of time.

Try it now

Here are a few ideas to think about as you plan applications in Azure. They sound basic, and a lot of the time, you may make these decisions automatically, without much thought. But it's still important to understand your application needs before you start to build and run applications!

- What regions should your application run in? Do you have a large concentration of users in a specific region? How will you provide redundancy?

  If you're building internal applications, run them in the Azure region nearest to your users. If you have a major office in Houston, Texas, for example (maybe you like rocket ships!), run your Azure applications and VMs in the South Central United States.

  If you're building external applications, do you anticipate having customers from certain regions? This configuration may require multiple instances deployed in different regions (and also provide high availability). We'll get to this configuration in chapter 12.
- Do you need to provide a lot of VM customizations? How long does it take to test and validate all those changes? What business needs drive them?

  In a traditional on-premises environment, a lot of time is often spent creating preconfigured images for deployments. In the cloud, try to minimize this time. The prebuilt Azure images include the latest security updates; they're tested for you and then geographically replicated for the fastest deployment times.

www.dbooks.org

If you do create your own images, use features like Azure Shared Image Gallery to distribute and replicate those images as needed (https://docs.microsoft.com/azure/virtual-machines/windows/shared-image-galleries).

2.1.3 VM sizes

There are various families of VM sizes in Azure. These families contain groups of similar virtual hardware types that are targeted for certain workloads. The sizes are sometimes updated as new hardware and workload offerings become available, but the core families remain constant. The family types are as follows:

- *General purpose*—Great for development and testing or for low-use production databases and web servers
- *Compute optimized*—High-performance CPUs, such as for production application servers
- *Memory optimized*—Larger memory options, such as when you need to run big databases or tasks that require a lot of in-memory data processing
- *Storage optimized*—Low-latency, high-disk performance for disk-intensive applications
- *GPU*—NVIDIA-based graphics-specialized VMs, if you need graphics rendering or video processing
- *High-performance compute*—Lots of everything: plenty of CPU, memory, and network throughput for the most demanding workloads

Just how big of a VM can you create in Azure? Things are constantly improving, but at the time of writing the largest VM you can create is an Mv2-series (part of the memory-optimized family) with 208 virtual CPUs and 5.7 TiB of memory. That should make for a decent Minecraft server, don't you think?!

The key thing to learn here is that the number of VMs and amount of CPU and memory you can request in Azure are limited only by your budget. You'd likely struggle to create VMs of this size in the traditional on-premises world.

When you create a VM in the Azure portal or by using the CLI or PowerShell, you must choose what size VM to use. A common VM size, such as D2s_v3, is often used as a default to get you started. This is probably way too much power for a basic web server in this chapter, but it's quick to create the VM and install the required packages!

The Azure portal lets you filter based on a rough size (such as small, medium, or large) or a specific family (such as the general-purpose or memory-optimized VMs). An estimated monthly cost is also shown to give you an idea of how expensive each VM is. Pay attention to the costs, as they can add up quickly! Within reason, you can usually change the VM size after the VM is up and running, though the VM must shut down and reboot to complete the process.

VM cost savings

The VMs created by default are often overpowered for what you need, but they're quick to deploy and use, which helps cut down on how much time you spend installing packages on your lunch break.

In the real world, pay attention to the memory, CPU, and storage demands of your VMs. Create appropriately sized VMs. This approach is the same as in the on-premises world, where you can end up with VMs that have much more memory or many more virtual CPUs assigned than they need.

There's also a special type of VM in Azure: the *B-series*. These VM sizes use burstable CPU and memory resources, and you can bank credits for unused compute resources. If you want to save your Azure credits, you can choose this series of VMs for the book exercises. They come with a lower price point and are great for scenarios in which you don't always need a lot of CPU and memory resources. Take care, though: depending on the size of the B-series VM you create, it may have less CPU and memory than something like the D2s_v3 series, so it will run a little slower.

2.1.4 Azure storage

Storage for VMs in Azure is straightforward. How many disks do you want, how big, and what type? The first two really aren't Azure-specific, so we'll skip them. These types of storage are available:

- *Premium SSD (solid-state drive) disks*—Use low-latency, high-performance SSDs and are perfect for production workloads. You should use mostly this type to get the best performance for your applications.
- *Standard SSD disks*—Use standard SSDs and deliver consistent performance compared with hard disk drives (HDDs). This type is great for development and testing workloads or for budget-conscious, low-demand production uses, such as web servers.
- *Standard HDD disks*—Use regular spinning disks and are ideal for infrequent data access, such as archive data or backups. This type is not recommended for running application workloads.

You don't need to dig much deeper into the specifics of storage to create a quick web server. You'll learn more in chapter 4, including about Ultra disks that are only for attached data disks. For now, it's enough to know that when you pick a VM size, that size helps define what type of storage you use.

The virtual disks you use, regardless of type, are called Azure *managed disks*. These managed disks allow you to create a VM and attach additional data disks without worrying about underlying storage accounts, resource limits, or performance quotas. Managed disks are also automatically encrypted at rest; there's nothing you need to configure to secure your data! Again, chapter 4 covers all this and more. For now, you can usually let Azure create the most appropriate disk based on the VM size you select.

www.dbooks.org

Try it now

To check your knowledge, work through the following questions:

- For most production workloads, what type of disk provides the best performance?

  A premium SSD disk is usually what you should run for production workloads. This type is often the default choice when you create a VM. Standard SSD disks are an okay second choice, and ultra SSDs should be only used on very disk-intensive applications that require low latency. Although there's a little bit of cost savings with standard HDDs, performance often is great, just as with on-premises virtual environments.
- What VM family is a good choice for a database server?

  A memory-optimized VM is a good fit, as databases often need a larger amount of memory than CPU resources. Always try to estimate resource needs and then monitor performance after deployment. Don't be afraid to switch to a different VM size to provide the desired performance.

2.1.5 Virtual networking

It sounds obvious, but a VM needs network connectivity if you want anyone to reach your applications. For a basic web server, you need both a virtual network and external connectivity. Chapter 5 covers core Azure networking in detail, and chapter 9 gets into how to distribute traffic to multiple VMs by using load balancers. Things get really cool in chapter 11, with Azure DNS and global routing of end users with Traffic Manager. I won't make you a network engineer, but you're going to learn a lot of Azure networking in this book!

To get started with the basics needed for this chapter, a virtual network in Azure is made up of the same core features as a regular physical network:

- An address space and a subnet mask, such as 10.0.0.0/16
- One or more subnets, which you can use to divide external, database, or application traffic, for example
- Virtual network interface cards (NICs) that connect VMs to a given subnet
- Virtual IP addresses that are assigned to resources such as a virtual NIC or load balancer

You can create a VM that's only attached to a virtual network without providing external connectivity, which may be the case for backend database or application servers. To connect to these VMs for administration and maintenance, you can create a virtual private network (VPN) connection, or you can use a private, dedicated connection to Azure from your on-premises networking equipment. In Azure, this dedicated connection is called *ExpressRoute*.

The basic web server you'll build in this chapter requires a specific type of virtual IP address: a public IP address. This public IP address is assigned to the virtual NIC and allows external traffic to reach your VM. Then you can control the flow of traffic to your VM with NSGs (network security groups). Think of a regular firewall that you use to open or close various ports and protocols; in Azure, network security groups lock down traffic by default and allow only the specific traffic that you define. Common traffic to allow would be HTTP or HTTPS on TCP ports 80 and 443. Remote management using remote desktop protocol (RDP) or Secure Shell (SSH) can also be opened, with care, which you'll do later in this chapter to see how to connect and install some packages.

2.2 Creating an SSH key pair for authentication

In the end-of-chapter lab exercise, you'll create what you're probably already familiar with: a Windows Server VM. This type of VM uses password-based authentication. A lot of applications in the cloud run on Linux; in fact, more than half of the VMs in Azure run Linux. You usually don't use password-based authentication with Linux; instead, you use SSH and a public-key pair. To start expanding your skills, the basic web server in this chapter runs Linux, so you need to get comfortable with how to create and use SSH. You don't *need* Linux experience to work in the cloud, but I highly recommend that you learn some of the basics!

SSH key pairs

SSH is a protocol used to communicate securely with remote computers and is the most common way to log in to Linux VMs. It's similar to using an RDP connection to a Windows VM, except that in Linux, the whole SSH session is typically console-based. With public-key cryptography, you can use a digital key pair to authenticate you with a remote Linux VM.

An SSH key pair has two parts: a public key and a private key. The public key is stored on your Linux VM in Azure. You keep a copy of the private key. When you need to log in to your Linux VM, the public key on the remote VM is matched with the private key you keep locally. If the key pairs match, you're logged in to the VM. There's a little more to the process than that, but at its core, public-key cryptography is a great means to verify identity.

I'd like you to get into the habit of using SSH keys to log in to Linux VMs. SSH keys are a lot more secure than passwords because, among other things, they aren't susceptible to brute-force password attacks. You should always focus on security as a central concept, especially in the cloud.

Try it now

Create an SSH public-key pair, using the Azure Cloud Shell:

www.dbooks.org

1 Open a web browser to https://portal.azure.com. Sign in to the Azure account you created in chapter 1 and then select the Cloud Shell icon at the top of the dashboard, as shown in figure 2.2. You can also open Cloud Shell directly at https://shell.azure.com.



Figure 2.2 Select and launch Cloud Shell in the Azure portal by selecting the shell icon.

2 The first time you open Cloud Shell, it takes a few moments to create persistent storage that's always then connected to your sessions. This storage allows you to save and retrieve scripts, configuration files, and SSH key pairs. Accept any prompts to allow this storage to be created.

3 If necessary, choose Bash from the drop-down menu in the top-left corner of Cloud Shell. PowerShell support is also available, though we'll focus mostly on Bash and the Azure CLI throughout the book.

4 To create a key pair, enter the following command:

```
ssh-keygen
```

5 Accept the default prompts by pressing the Enter key. In a couple of seconds, you have an SSH public-key pair that you can use with all your VMs! The `ssh-keygen` command defaults to a 2,048-bit-length key and uses the RSA version 2 protocol. This is a good balance of security and is recommended for most use cases. Figure 2.3 shows an example of a completed SSH key pair in Cloud Shell.

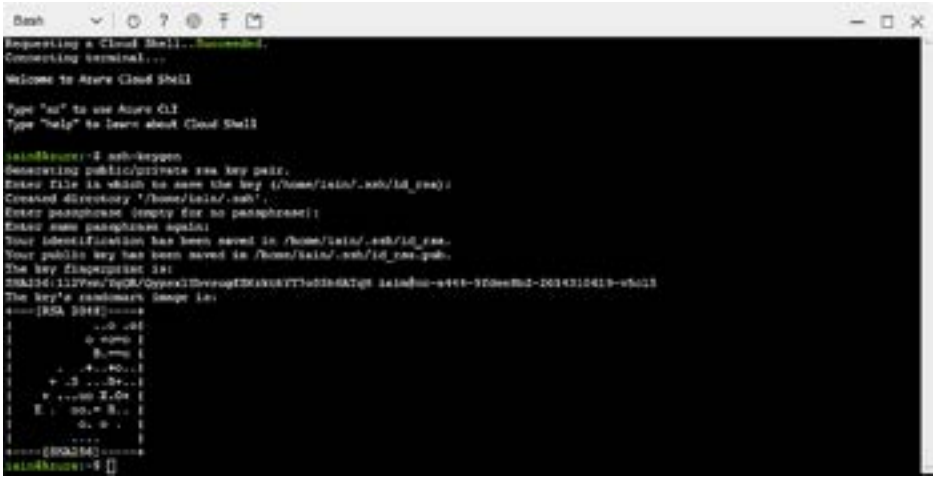


Figure 2.3 An SSH key pair created in the Azure Cloud Shell with the `ssh-keygen` command

6 To view your public key and use it with a VM, enter the following command:

```
cat .ssh/id_rsa.pub
```

7 Select the output, and copy it to a simple text file on your computer. You'll use this public key to create a VM in section 2.3; this VM is referenced from the Azure CLI throughout the rest of the book. Usually, you don't need to copy and paste the whole key each time, but it's good to see what's happening at first. This information isn't super-secret, so using Notepad or TextEdit to create and save a copy of the key is fine. Be careful when copying the output of the public key, because it's sensitive to additional whitespace or missing characters. An example of a complete SSH public key is as follows:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDPGaOBsfhJJOHAWAv+RLLR/vdUTzS9HOIj
➥JyzWWLsnu0ESH2M6R+YYPPNXv9X7dmVyM1zCXXEaLucpnyFjevbwPedxTgifyxgCFTgylr1
➥kg7o4EyCTGBGhTA+hSHuhXGXa12KPdKWehsPwHMa6Hs8fbt/in9Z1k2ZAwvbT+LWPcmJgNO
➥FuolIHOsOEeoQQqdXLrGa7NU/3fzSXdT9Y2BT1KLINc4KnwdOuONddLw3iANvK+Gkwax8iK
➥7IicKMoammwvJUCRf+MTEK9pZ84tfsc9qOIAdhrCCLbQhtoWjZpIwYnFk+SNBE8bZZtB8b2
➥vkDFNZlA5jcAd6pUR3tPuL0D iain@cc-a444-9fdee8b2-2014310619-v5cl5
```

TIP Cloud Shell is browser-based, so the keyboard shortcuts for copy and paste may work a little differently than you're used to. Ctrl-Insert and Shift-Insert should copy and paste, rather than the regular Ctrl-C and Ctrl-V.

2.3 Creating a VM from your web browser

Now that you know a little of the theory of Azure VMs and have created an SSH key pair, you're ready to jump in and create a VM. I'm going to get you started and then let you configure the VM based on what you've just learned, so you'll definitely need to pay attention!

The Azure CLI and Azure PowerShell tools are incredibly powerful, but a big strength of Azure is how much time has gone into building a great portal experience. The Azure portal is a web-based graphical tool that lets you see how all the different components come together and do a quick visual check to ensure that all is well. The portal includes a couple of unique things that the other tools don't provide, and it's fast to use because you don't have to install anything.

Try it now

Creating a VM in Azure gives you a lot of defaults you can use to reduce the number of choices you have to make. For this exercise, look at the resources Azure is going to create based on what you learned in section 2.2 for things like network and storage:

1 In the Azure portal (https://portal.azure.com), select Create a Resource in the top-left corner of the dashboard. Popular resources should be listed, including the most recent Long Term Support (LTS) version of Ubuntu (as of this writing, it's Ubuntu Server 18.04 LTS).

www.dbooks.org

2 Select the LTS version.

You can also search the marketplace across the top of the window or browse the list of high-level services (such as Compute and Networking) to get an idea of what else is available for your own future needs. Try to stick with Ubuntu Server 18.04 LTS so that you can follow one of the upcoming exercises to install the web-server components.

3 Create a resource group for your web server.

When you create resources in Azure, they're logically contained in a resource group that you define. These groups typically contain like-minded resources for your applications. Chapter 7 gets into ways to plan and manage applications by using resource groups.

For now, I suggest naming resource groups by chapter to help you organize things as you go. Name the resource group in this exercise `azuremol-chapter2`, for example.

4 Give your VM a name, such as webvm, and then pick a region close to you. Don't worry about infrastructure redundancy for now.

Look at the options for the VM image, just to get a feel for the other options, but for this exercise, stick with Ubuntu Server 18.04 LTS. The default VM size is fine for this exercise, but again, look to see what's available and how you query for different sizes and what hardware they run. See how the sizes align with the VM families you looked at earlier in this chapter.

5 Make sure that you're using SSH public-key authentication and then provide a username, such as azuremol. You'll use this username to sign into the VM in the next exercise.

6 Copy and paste the SSH public key you created in the previous section. Again, make sure that there's no additional whitespace or formatting when you copy and paste the public key. The SSH key should be on one line. Even word wrap in Notepad can cause problems! The Azure portal validates the key before you can proceed.

7 To connect to the VM in the next exercise and install the web-server components, open SSH on port 22.

Opening SSH on a public VM isn't great security practice. Chapter 16 looks at how to open and restrict access automatically by using just-in-time VM access.

Look at some of the other ports you can open here. HTTP and HTTPS are common ports to open, and you're meant to be building a web server in this chapter, right? Don't cheat and open those ports just yet; I want to introduce you to the Azure CLI in the next exercise, where you allow HTTP traffic.

Connect securely by using a bastion host

In real-world scenarios, you shouldn't open remote management ports for SSH or RDP to the public internet. Seriously, just don't! Follow the best practices that you should use in the non-cloud, on-premises world, such as connecting only when necessary and limiting remote access to a specific set of management addresses.

(continued)

A common way to provide remote access is to use a bastion host, or a jump box. In this kind of setup, you don't connect directly to application servers from your laptop or desktop. Instead, you connect to a dedicated bastion host and then connect to the server you need to manage. This approach locks down access to a limited set of addresses and provides a secure way to allow remote management.

Azure Bastion (https://docs.microsoft.com/azure/bastion) provides a managed approach to this need for secure remote connection. You create an Azure Bastion host in a dedicated subnet and then use this host to connect to VMs that run your applications. Those VMs don't need to be publicly accessible. You can do everything through the Azure portal without opening network ports for SSH or RDP. The bastion host itself is managed for you in terms of security updates and network security group rules.

8 Look at some of the other VM configuration options for storage and networking to familiarize yourself with the options, though you can leave everything as default for now.

9 There are also some cool management options, such as enabling Auto-shutdown, Backups, and Diagnostics, which are covered in chapters 12 and 13. For now, turn off things like boot diagnostics and OS guest diagnostics, as you need to create and configure a storage account for them to work.

10 When you're all set, review and create your basic VM.

2.4 Connecting to the VM and installing the web server

When your VM is up and running, you can use the SSH key you created earlier to log in to your VM. Then you can begin to install and configure the web server, and you can do it all through Cloud Shell.

2.4.1 Connecting to the VM with SSH

This section looks at how you can get the connection details for your VM quickly.

Try it now

If Linux is new to you, don't worry! Follow the next few steps to log in to your VM:

1 In the Azure portal, browse to and select Virtual Machines on the navigation bar on the left side of the screen. It takes a couple of minutes to create the VM from the preceding exercise, so select the Refresh button until VM status shows *Running*. When ready, choose your VM and select Connect, as shown in figure 2.4.

www.dbooks.org

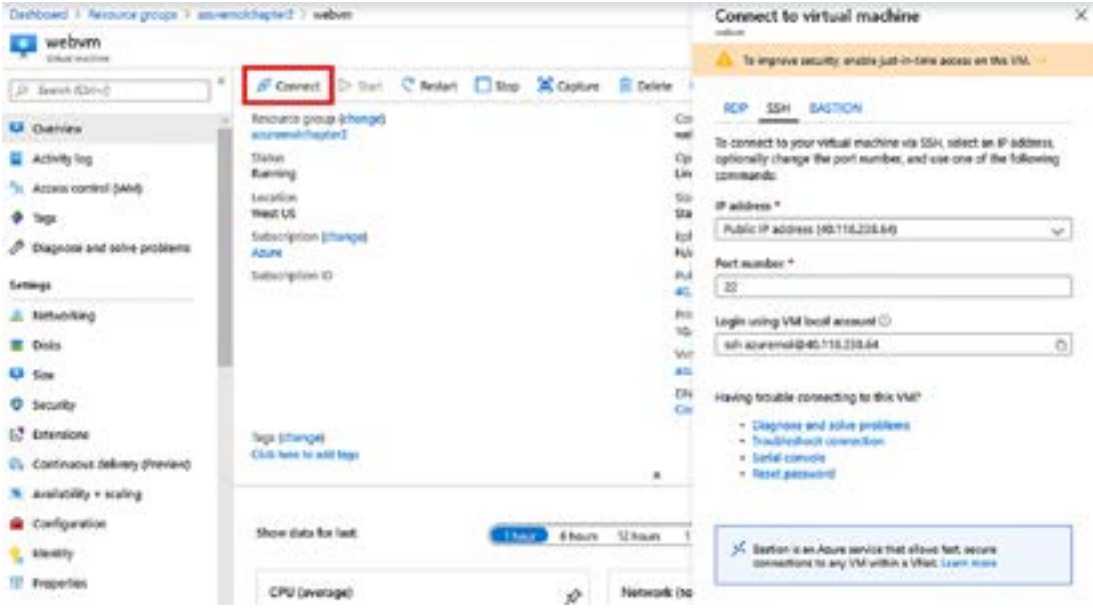


Figure 2.4 Select your VM in the Azure portal and then select Connect to generate the SSH connection information.

With a Linux VM, you're shown the SSH command that includes your name and public IP address. Copy this SSH connection command, such as `ssh azuremol@104.209.208.158`.

On a Windows VM, the Connect button downloads an RDP connection file to your computer that's prepopulated with the public IP address of your VM.

2 If necessary, open Cloud Shell again. If you're going to be switching between Cloud Shell and the portal, you can minimize Cloud Shell to keep it available in the background.

3 Paste the SSH command into Cloud Shell and then press Enter. The SSH key you created earlier is automatically used to authenticate.

The first time you connect to a VM with SSH, it prompts you to add it to a list of trusted hosts. This is another layer of security that SSH provides. If someone tries to intercept the traffic and direct you to a different remote VM, your local SSH client knows that something has changed and warns you before connecting.

Accept the prompt to store the remote VM connection. Figure 2.5 shows the SSH connection process in the Azure Cloud Shell.

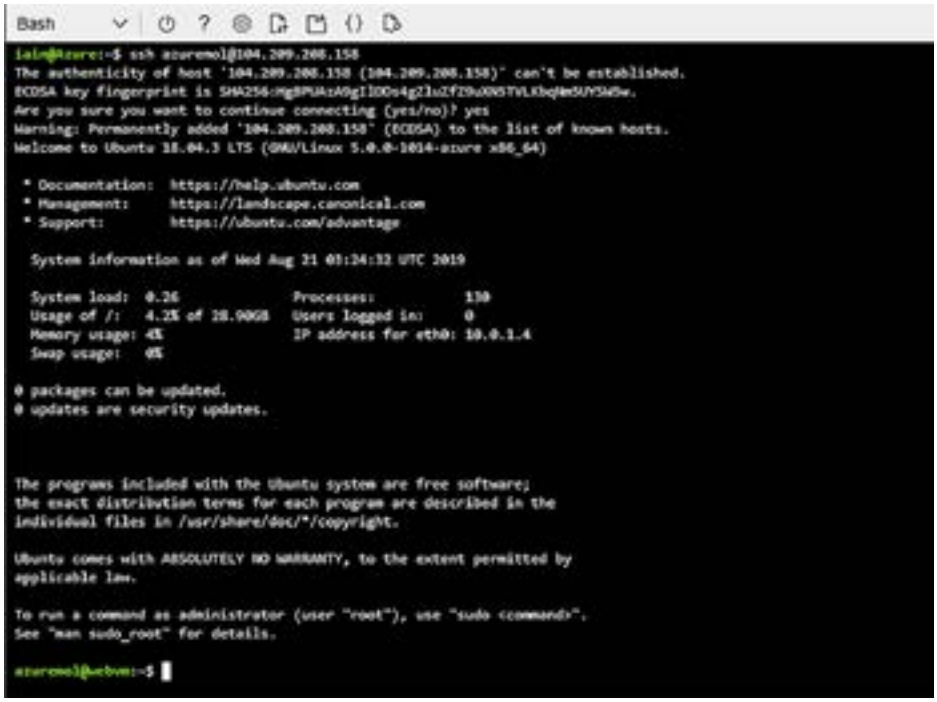


Figure 2.5 Use the connection string shown in the Azure portal to create an SSH connection to your VM from Cloud Shell.

At this point, you're either home away from home or the Linux prompt is totally foreign. Don't worry. You don't need to know a huge number of Linux commands, and every command is explained as we go along. That said, I highly recommend that you learn at least some basic Linux administration skills. A lot of the cloud is based on Linux systems, and there's a big move toward containers and microservices for application development and management. If you're an old-school Windows admin, welcome! There's something lined up for you at the end of the chapter, so bear with it.

2.4.2 Installing the web server

Create a VM? Check. Connect to the VM with SSH? Check. Now you can install the packages for a web server and get ready to see it in action.

Azure supports many Linux distributions (*distros*). Package-management tools and configuration file locations vary a little among distros. You're going to use Ubuntu in this book because it's one of the most popular and well-documented Linux distros for cloud computing. If you get stuck along the way, you should be able to find plenty of documentation to help, starting at https://help.ubuntu.com. If you want to use a different distribution that you're already comfortable with, feel free to use that! Otherwise, stick with Ubuntu.

www.dbooks.org

Try it now

From your SSH session to the VM, install the web server packages with APT:

1. In Ubuntu, you install packages with an Advanced Packing Tool (APT)—a super-powerful package-management tool that automatically installs any additional packages it needs. All you need to do is say "Install a web server," and APT installs all the required components.

   For this example, install the LAMP web stack. This is probably the most common set of web components: Linux, Apache (a web server), MySQL (a database server), and PHP (a web programming language):

   ```
   sudo apt update && sudo apt install -y lamp-server^
   ```

   The first command updates the available packages, which is good practice to make sure you can install the latest and greatest packages. When that command finishes, run the next command with `&&`. Why not just start a new line for the next command? The `&&` runs the next command only if the preceding command was successful. If there was no network connectivity for apt to get the latest packages, for example (humor me, I know you must have network connectivity to connect in the first place!), there's no point in running the `install` command.

   If the update command is successful, apt determines what additional packages it needs and begins to install `lamp-server`. Why is there a caret symbol at the end (`^`)? That symbol tells `apt` to install the entire set of packages that make up the LAMP server, not just a single package named `lamp-server`.

2. The installer may prompt you for a password or default to using an empty MySQL password. That's not very secure, and for real production use, you need to specify a strong password. In chapter 15, you'll get really cool and store a strong, secure password in Azure Key Vault that's automatically injected into this MySQL install wizard.

   It takes a minute or so to install all the packages for your LAMP web stack; then you're finished.

3. Type `exit` to log out of your VM and return to the Cloud Shell prompt.

That's it! Your web server is up and running, but you won't be able to access it in a web browser just yet. To do that, you need to allow web traffic to reach the VM.

2.5 Allowing web traffic to reach the VM

Your web server is up and running, but if you enter the public IP address of your VM in a web browser, the web page doesn't load. Why? Remember the network security groups discussed briefly in section 2.1.5? When you created the VM, a network security group was created for you. A rule was added to allow remote management—in this case, SSH.

The rest of the VM is locked down. To allow visitors to access your web server over the internet, you need to create a rule in the network security group that allows web traffic. Otherwise, no one can order pizzas!

2.5.1 Creating a rule to allow web traffic

This section mixes things up a little by using the Azure CLI to create a rule for web traffic. You could have opened this HTTP port in the portal when you created the VM, but then you would have missed half the fun!

The Azure CLI is available in Cloud Shell. There's nothing you need to install. Chapter 5 covers virtual networking and network security groups in more depth; for now, you can check out how quick and powerful the Azure CLI is with just one command.

Try it now

Open the Azure Cloud Shell, and follow these steps to see the Azure CLI in action:

1. If you closed your Cloud Shell window, open it again from the Azure portal. Make sure that the Bash shell loads, not PowerShell. If necessary, switch to the Bash version.
2. To see the Azure CLI and installed modules, type `az --version`. A list of modules and version numbers is shown. What's great about Cloud Shell is that it always has the latest and greatest version available.

NOTE If you're observant, you may have noticed the command output information about the version of Python. Why is this information important? Python is a powerful, popular programming language. The Azure CLI is written in Python, which is part of what makes it cross-platform and available for you to install locally on any computer if you don't want to always use Cloud Shell. To keep up with Microsoft's drive to contribute to the open source community, the Azure CLI is made available on GitHub for anyone to make contributions, suggestions, or report problems (https://github.com/Azure/azure-cli).

3. To open a port, specify the VM name and its resource group, along with the port number. For web traffic, you need to open port 80. Enter the resource group (`-g`) and VM name (`-n`) you specified when you created your VM:

```
az vm open-port -g azuremolchapter2 -n webvm --port 80
```

2.5.2 Viewing the web server in action

Now that you have a port open to your VM, see what happens when you try to access it in a web browser:

1. In the Azure portal, select your VM, if you navigated away from it. The public IP address is listed in the top-right corner of the VM overview page.

www.dbooks.org

2 Select the address, and copy it.

3 In your web browser, open a new tab or window, and paste in the public IP address. The default Apache website loads, as shown in figure 2.6. Okay, it doesn't look like a pizza store, but you have the foundation ready to bring in your code and start to build your application!



Figure 2.6 To see your web server in action and view the default Apache 2 page, enter the public IP address in a web browser.

2.6 Lab: Creating a Windows VM

The preceding sections walked through installing the LAMP stack on an Ubuntu Linux VM. This platform is a common one for websites, but you may want some love and attention if you have a Windows background! Your development teams or business decision makers may want to use .NET, for example. Even so, you can run .NET Core on Linux VMs, so don't let the language drive your decision.

From what you learned in the step-by-step example, try to create a VM that runs Internet Information Services (IIS). Here are some hints:

- You need a VM that runs Windows Server 2019.
- You use RDP, not SSH, so expect a slightly different connection experience.
- In Server Manager, look for an Add Roles and Features option.

- You need to install the web server (IIS).
- Don't forgot to open a network port for HTTP traffic on TCP port 80. You can use the portal if you want.

2.7 Cleaning up resources

As you create resources in Azure, the billing meter starts to spin. You're charged by the minute, so it's wise to form good habits and not leave resources such as VMs running when you're done with them. You have two ways to stop the billing charges for running a VM:

- *Deallocate a VM.* You can select the Stop button in the portal to stop and deallocate a VM, releasing all held compute and network resources.
- *Delete a VM.* This option is rather obvious. If there's nothing left in Azure, there's nothing to pay for. Make sure that you're finished with the VM before you delete it. There's no Undo button in Azure!

I recommend that you create a resource group for each application deployment as you start to build things in Azure. As you walk through the exercises in this book, that's what you'll do. If you name your resource groups by chapter, such as `azuremolchapter2`, it'll be easier to keep track of your resources and what to delete. This practice makes cleanup a little easier, because you can delete the entire resource group at the end of each chapter. Choose Resource Groups from the navigation menu on the left side of the screen; open each resource group you've created in this chapter; and then select Delete Resource Group, as shown in figure 2.7. To confirm, you're prompted for the resource group's name.

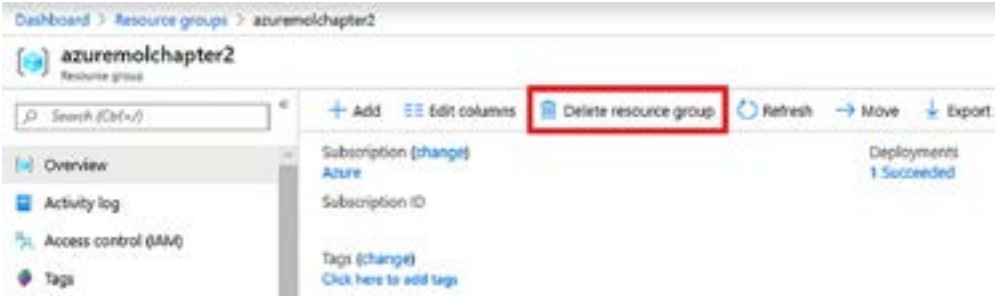


Figure 2.7 To save costs, delete resource groups when you no longer need them.

If you get into the habit of deleting resources when you're done with them, you can comfortably make it through this book on those free Azure credits. At the very least, deallocate your VM at the end of each lesson so that you can resume the next day and stop the clock on the billing.

www.dbooks.org

2.8 Houston, we have a problem

Sometimes, you'll run into problems in Azure. There—I said it. Usually, the Azure platform is good about issues that come up as you create resources:

- The Azure CLI or Azure PowerShell reports back as you run commands, so it should be obvious when something goes wrong. Azure PowerShell typically uses nice, calm, red text to get your attention.
- The Azure CLI can be a little more cryptic because it usually includes the actual responses to the underlying REST API calls from the server. If this is new, it can take a few successes and failures to understand what's going wrong. The helpful part of getting the REST responses is that you can copy and paste the error messages into your favorite search engine and usually get solid results to help you troubleshoot.

Take a REST? We just got started!

When you open a web page in your browser, your computer is communicating with a web server using HTTP. I can almost guarantee that you've seen a 404 error message on a website before. That message means that the web page couldn't be found. Other common errors you may have seen are 403 (you don't have permission to view the page) and 500 (the server encountered an error).

Even when things go well, under the hood, your browser receives code 200 messages when the page loads fine or code 301 messages if a page has been redirected to a new location. You don't need to understand and keep track of all these codes; they're just standard ways that HTTP facilitates communication between computers.

Earlier, this chapter discussed how to create and manage Azure resources through the web portal, CLI, or PowerShell. All the Azure services are accessed by Representational State Transfer (REST) application programming interfaces (APIs).

If this is new to you, REST APIs are a (somewhat) standardized way of exposing services via HTTP. You use standard HTTP requests such as `GET` and `POST` to request information or make a change, and when the platform accepts and processes the request, you receive a status message. Azure has a well-defined set of REST APIs.

You don't need to understand what all this means. Just be aware that when you see an error message, it's not always in the most human-readable, helpful format. Sometimes, you get the raw HTTP response from the REST API that you must decipher by yourself. Again, paste this error into your favorite search engine. There's a good chance that someone has encountered the problem and provided a more human-readable reason for what went wrong and what you need to correct.

The most common problems with VMs occur when you connect to your VM. You could be connecting for remote administration with SSH or RDP, or trying to access your applications through a web browser or desktop client. These issues are often

network related. I don't get into totally blaming the network folks until chapter 5, so here are a couple of things to check:

- Can you connect to any other Azure VMs or applications running in Azure? If not, something local to your network is probably preventing access.

  If you can connect to other Azure resources, make sure that you opened the network security group rules (section 2.5). Chapter 5 digs into these rules.
- For authentication problems, try the following:
  - Confirm that you have the correct SSH keys. Azure should tell you when you create the VM whether the public key is invalid, but if you have more than one private key, make sure that you're using the correct one.
  - For RDP issues, try to connect to localhost\<*username*> and enter your password. By default, most RDP clients try to present local credentials or network credentials, which your VM won't understand.

www.dbooks.org

3 Azure Web Apps

In chapter 2, you created a virtual machine (VM) and manually installed packages to run a basic web server. You could build an online pizza store with this VM if you were hungry to get started. One of the biggest use cases for Azure VMs is to run web applications, typically at scale. Web applications are a comfortable workload for VMs. Comfortable is nice, if you also like the maintenance that goes with managing all those VMs—you know, fun things like software updates, security patches, centralized logging, and compliance reports. What if you could get all the power of a secure web server to run your web applications, including the ability to automatically scale to meet demands, but without the need to create and manage all those VMs? Let me introduce you to the Azure Web Apps service.

In this chapter, we'll compare the Infrastructure as a Service (IaaS) approach of VMs and web servers to the Platform as a Service (PaaS) approach. You'll learn the benefits of Azure Web Apps as you create a web application and see how to work with its development and production releases. Then you'll learn how to deploy your web app automatically from a source control, such as GitHub. This workflow is shown in figure 3.1. Azure Web Apps allows you to deploy and run your online

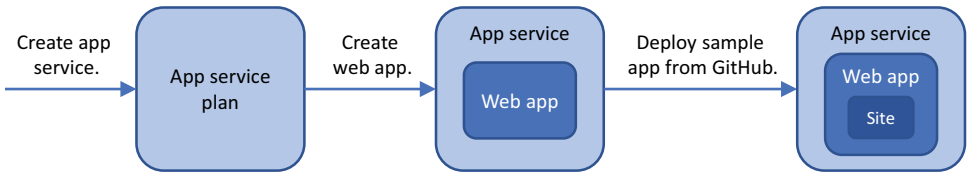


Figure 3.1 In this chapter, you'll create an app service plan and a basic web app and then deploy a website from GitHub.

pizza store in a matter of minutes, without the need to install and configure a VM and web-server packages.

3.1 Azure Web Apps overview and concepts

With Azure Web Apps, you start to dip your toes into the wonderful world of PaaS solutions. If you think cloud computing is all about VMs, you should probably reset that idea a little. At the start of this book, I talked about buying computer power and focusing on your applications and customers. As you move from IaaS solutions, such as VMs, and drift toward PaaS solutions, such as web apps, your applications and customers become the focus.

To run web applications on IaaS VMs requires management of the OS, application updates, security and traffic rules, and configuration of the whole system. With Web Apps, you upload your web application, and all those management tasks are taken care of for you. Now you can focus on improving the application experience for your customers or improving availability with scaling options and traffic management.

Does that mean you should never run VMs to host a web application? Probably not. There are valid reasons to run the entire application stack and configure it yourself, such as if you need very specific application support or language run time. But Web Apps can provide many of the use cases for running web applications.

3.1.1 Supported languages and environments

What kind of flexibility does Web Apps offer in terms of programming languages you can use to build your web application? Quite a lot! There are two primary platforms for running Web Apps: Windows and Linux. You can run .NET Core, Node.js, Python, Java, Ruby, and PHP web apps natively on both Windows and Linux web apps instances. On Windows, you can also run the full .NET framework. If you want to be really cool and run your web application in containers, there's also Web Apps for Containers, which lets you run native Docker containers for Linux. We'll dive more into containers and Docker in chapter 19. For now, understand that your options are covered with Web Apps!

When may Web Apps not make sense? Not all application languages are supported by Web Apps. Say you really want to torture yourself with a web application written in Perl. In that scenario, you'd likely fall back to running on IaaS VMs that you manage yourself, because there's no support for Perl in Web Apps. But Web Apps arguably supports the most common web programming languages that you'd want to use. You should probably look at a newer version of your app than one written in Perl, too.

Web Apps provides support not only for various languages, but also for various versions of those languages. Take PHP, for example. Typically, you can select three or four versions of PHP to best support your application. And best of all, you don't have to worry about the dependencies on the underlying web server to support it all, as you would if you managed an IaaS VM yourself. Python is another example of differences between the stable 2.7 and 3.6 (and later) versions, as shown in figure 3.2.

www.dbooks.org

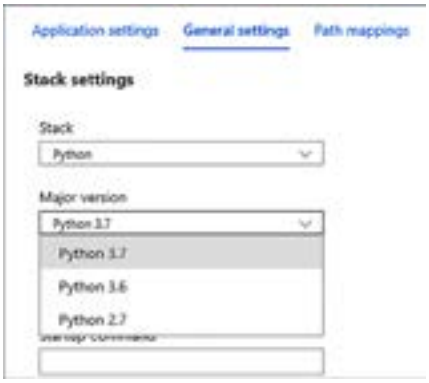


Figure 3.2 Select a specific version of a language in the Web Apps application settings.

Web Apps stays up to date on security fixes, too. But don't expect an older version of PHP or Python to continue to be supported indefinitely. There will be a cutoff on supported older versions at a certain point. Again, that may be a time when you fall back to running IaaS VMs yourself if your app needs an older language version. But if you need to run an older version of a given language to support a legacy application, don't get sucked into a constant-maintenance approach. Always look to move those legacy apps to more modern supported platforms.

3.1.2 Staging different versions with deployment slots

Deployment slots provide a staged environment for your web application. You can push new versions of your app to a deployment slot and get them running, using environmental variables or database connections, without affecting the live site. When you're happy with how things look and feel in a deployment slot, you can switch this version to the live site in an instant. Then the previously live site switches to a deployment slot of its own, providing an archived version, or you can flip the app back to production if necessary.

The number of available deployment slots varies based on the tier of web app you select. A larger number of deployment slots enables different developers to use multiple staged versions as they stage and test their own updates.

3.1.3 App service plans

Web Apps is part of the wider App Service family in Azure. Azure App Service also includes Mobile Apps, API Apps, and Logic Apps. All but Logic Apps are available in every region that Azure runs in. Here is a great resource to check out Azure service availability by region: https://azure.microsoft.com/regions/services. Many services are available globally.

When you need to create an App Service resource, such as a web app, you create or use an existing service plan. The service plan defines the amount of resources available

to you, how much automation is available to scale and back up your web app, and how highly available to make your site with staging slots and Traffic Manager (a way to geographically route traffic to the closest instance for a user, which we'll look at in chapter 11). As with anything, you get what you pay for. Your application and business needs should guide you as to the amount of resources required and what additional features are needed. Each service tier builds on the features of the lower tiers, generally adding more storage and available resources.

The four main service plan tiers are as follows:

- *Free/Shared*—Uses a shared infrastructure; offers minimal storage, and has no options for deploying different staged versions, routing of traffic, or backups. The Shared tier allows you to use a custom domain and charges for this domain.
- *Basic*—Provides dedicated compute resources for your web app, and allows you to use SSL and manually scale the number of web app instances you run. The free/shared and basic tiers provide a good environment for you to test the Web Apps service, but I wouldn't recommend running any actual production or development workloads. The performance isn't a limiting factor, but you miss some of the automated features, such as backups and scaling.
- *Standard*—Adds daily backups, automatic scale of web app instances, and deployment slots, and allows you to route users with Traffic Manager. This tier may be suitable for low-demand applications or development environments in which you don't need a large number of backups or deployment slots.
- *Premium*—Provides more frequent backups, increased storage, and a greater number of deployment slots and instance scaling options. This tier is ideal for most production workloads.

The case for isolation

With PaaS solutions such as Web Apps, the infrastructure is intentionally abstracted. As the names of some of the service plan tiers imply, web apps run across a shared platform of available resources. That's not at all to say that web apps are insecure and that others can view your private data! But compliance or regulatory reasons may require you to run your applications in a controlled, isolated environment. Enter *App Service Environments*: isolated environments that let you run App Service instances such as web apps in a segmented part of an Azure data center. You control the inbound and outbound network traffic, and can implement firewalls and create virtual private network (VPN) connections back on your on-premises resources.

All these infrastructure components are still largely abstracted with App Service environments, but this approach provides great balance when you want the flexibility of PaaS solutions but also want to retain some of the more fine-grained controls over the network connections traffic flow.

You can do quite a lot with the Free and Basic tiers, although for production workloads, you should probably use the Standard or Premium tier. This chapter's example

www.dbooks.org

uses the Standard tier so that you can see all the available features. When you use Web Apps with your own applications, you can decide how many of these features you need and select the most appropriate service plan tier accordingly.

3.2 Creating a web app

With a little theory under your belt, take a look at a web app in action. There are a couple of steps to getting an application running. First, you create the basic web app and see the default site in your browser. Then, you use a sample web page from GitHub and push that to Azure. Maybe your web developers have started to build a frontend for your online pizza store, so you have a basic site ready to upload.

> **NOTE** If you've never used Git before, don't worry. You don't need to understand what Git is doing at this point, and there's room at the end of the chapter to play around and explore a little. *Learn Git in a Month of Lunches,* by Rick Umali (https://www.manning.com/books/learn-git-in-a-month-of-lunches), is an excellent intro to using Git if you want to learn a little more, and it's available to read for free on the Manning liveBook platform.

3.2.1 Creating a basic web app

Just as I did in chapter 2, I'm going to give you some rough guidance along the way, but see whether you can apply some of the theory on application run times and app service plans to create a web app. If you're not sure about some options, it's safe to accept the defaults for now.

> **PaaS, not IaaS**
>
> This web app is a new resource and is separate from VMs like the one you created in chapter 2, which is an IaaS approach to building and running web applications. The PaaS approach is Web Apps. There's no real relationship between the two types. In fact, if you followed the advice in chapter 2 and deleted your VM, this web app runs without a VM in your Azure subscription at all!

> **Try it now**
>
> To create your web app, complete the following steps:

1. Open a web browser to https://portal.azure.com, and log in to your Azure account.
2. In the portal, select Create a Resource in the top-left corner of the dashboard.
3. Choose Web from the list of resources you can create, and then select Web App.
4. To help keep things clean and organized as you did in chapter 2, I suggest that you create a dedicated resource group for your web app, such as azuremolchapter3.

5. For the web app name, enter a globally unique name. This name must be unique, as it creates the URL to your web app in the form http://*<name>*.azurewebsite.net. If you're wondering, yes, you can apply a custom domain name here. For now, use the default azurewebsites.net address.
6. You're going to push some basic HTML code, not a Docker container, but look at all the different run-time stacks that are available. You can change this setting after you've created the web app, but for now, choose an ASP.NET run time that runs on Windows.
7. Let Azure create an app service plan automatically, but change the size to S1 Standard. This tier provides all the core features without providing too many resources for your basic demo website. In real-world deployments, this step is where you could manually create and configure your own app service plans or select an existing plan.
8. When you're ready, review and create your first web app.

It takes a few seconds to create your app service. When you're ready, browse to and select App Services on the navigation bar on the left side of the screen; then choose your web app from the list. On the Overview window of your web app, view and select the web app's URL, such as https://azuremol.azurewebsites.net.

When you select the URL to your web app, a new browser window or tab opens. The default web app page loads, as shown in figure 3.3. It still doesn't look like pizza. . . .

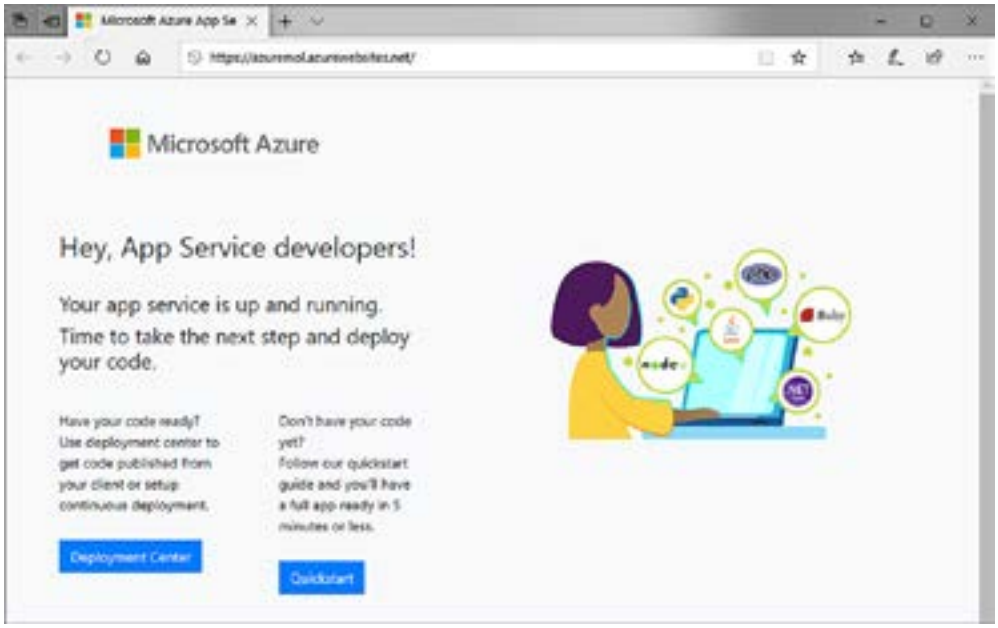


Figure 3.3 To see the default web app page in action, open a web browser to the URL of your site.

www.dbooks.org

3.2.2 Deploying a sample HTML site

You have a web app in Azure, but it's a rather dull, default website. How do you get your own website in Azure? One of the most common cross-platform ways is to use Git. Most application developers and teams use a source control system. Rather than storing files on your computer and saving changes as you go, source control systems keep track of changes and allow you to work with others. You can create test releases that won't affect your production code and revert to earlier versions if problems arise. Git is one of the most common source control systems; GitHub is a cloud-based service that lets you share and contribute code with the rest of the world. Microsoft acquired GitHub in 2018, but there's nothing that forces you to use GitHub with Azure, or vice versa. All the samples in this book are available in GitHub.

For this example, you create a local copy of the static HTML sample site and then push the files to your Azure web app. This workflow is shown in figure 3.4.



Figure 3.4 You create a local copy of the sample files from GitHub with the `git clone` command. To push these local files to your Azure web app, you use `git push`.

Try it now

To get a copy of the sample HTML page from GitHub and push it to your web app, complete the following steps:

1 Open Cloud Shell in the Azure portal, and wait a few seconds for your session to connect. To get started, you need the HTML sample site from GitHub.

To *clone*, or copy, the HTML sample site from GitHub, enter the following command:

```
git clone https://github.com/fouldsy/azure-mol-samples-2nd-ed.git
```

If this is your first time using Git in Cloud Shell, you need to define a couple of settings for Git to understand who you are. For most of the exercises in this book, Git doesn't care who you are, but for use with your own projects and applications, it's a great way to track who performs certain actions in a source

control system. You need to define these settings only once. Enter your own email address and full name in git `config` as follows:

```
git config --global user.email "iain@azuremol.com"
git config --global user.name "Iain Foulds"
```

2 Change to the azure-mol-samples-2nd-ed directory that was created when you cloned the Git repo:

```
cd azure-mol-samples-2nd-ed/03/prod
```

3 To get ready to upload the sample HTML page, you must initialize Git and then add and commit your files. Don't worry too much about the Git commands right now! You need to tell Git what files to track and add, and give yourself a way to track those changes later if needed:

```
git init && git add . && git commit -m "Pizza"
```

4 Now you can get ready to push this HTML sample site to your web app. First, define the deployment credentials. To secure Web Apps when you use a deployment method like Git or FTP, you must provide a username and password. Web Apps can use a set of credentials that are valid across all app service plans in Azure or app-level credentials that apply only to a single app.

In the real world, I recommend that you use app-level credentials to minimize the scope of an attack should one of the credentials be exposed. Azure automatically generates the app-level credentials, but you have to retrieve and assign these credentials each time. To keep things simple, use a defined credential that you can reuse in the next few chapters.

Create the deployment credentials, and specify your own username and secure password. The username must be globally unique. If it helps, add your initials to the username or a naming convention that makes sense for your environment.

```
az webapp deployment user set --user-name azuremol --password @azurem0l!
```

5 Next, you need to get the URL for your web app's Git repository. Enter the web app name (not the username you created in step 4) and resource group that you specified when the web app was created to view the Git repo URL.

How to slash-dot yourself

In the following example and later chapters, the backslash (\) means that the command continues on the next line. It's a way to wrap long lines, and this approach is used in a lot of online samples in which you can copy and paste the commands. You don't have to type the backslashes in this book's examples if you don't want to! Just continue typing the additional parameters as part of one big line.

If you're using the Windows command prompt rather than a Bash shell, don't include the backslashes. If you do, you really won't get the outcome you desire!

www.dbooks.org

```
az webapp deployment source config-local-git \
--name azuremol \
--resource-group azuremolchapter3 -o tsv
```

6 Your web app is configured to work with Git repos, so you need to tell Cloud Shell what that repo is. In Git, you define these locations as *remotes*.

Copy your Git clone URL from step 5 and then set this URL as a destination for the HTML sample site in Cloud Shell with the following command:

```
git remote add azure your-git-clone-url
```

7 To upload, or copy, files with Git, you *push* them. Where does Git push them to? A *remote* like the one you configured in step 6, such as *azure*. The final part of the command is a branch—typically, *master*. A branch in Git is how you keep trawent work-in-progress models. A best practice in production environments is to push to release branches that you can name as you wish, such as *dev* or *staging*. These additional branches allow your production code to run as normal; then you can work on new features or updates safely and without any effect on real workloads that your customers use.

Push the HTML sample site to your web app:

```
git push azure master
```

8 When prompted, enter the password that you created for the deployment credentials. You can copy and paste the password to minimize errors here.

You can see from the output that the existing default web app site page is removed, and the HTML sample site is uploaded and configured to run. Here's some sample output:

```
Counting objects: 3, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 510 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: Updating branch 'master'. remote: Updating submodules.
remote: Preparing deployment for commit id 'dda01e9d86'.
remote: Generating deployment script.
remote: Generating deployment script for Web Site
remote: Running deployment command...
remote: Handling Basic Web Site deployment.
remote: Creating app_offline.htm
remote: KuduSync.NET from: 'D:\home\site\repository' to:
     'D:\home\site\wwwroot'
remote: Deleting file: 'hostingstart.html'
remote: Copying file: 'index.html'
remote: Deleting app_offline.htm
remote: Finished successfully.
remote: Running post deployment command(s)...
remote: Deployment successful.
To https://azuremolikf@azuremol.scm.azurewebsites.net/azuremol.git
  * [new branch]       master -> master
```

To see your updated web app, refresh your site in a web browser or open it again from the Overview window in the Azure portal. It should look like the wonderful example in figure 3.5. Yes, the site is basic, but the workflow for deploying the most basic static HTML site to a complex .NET or Node.js web app is the same!

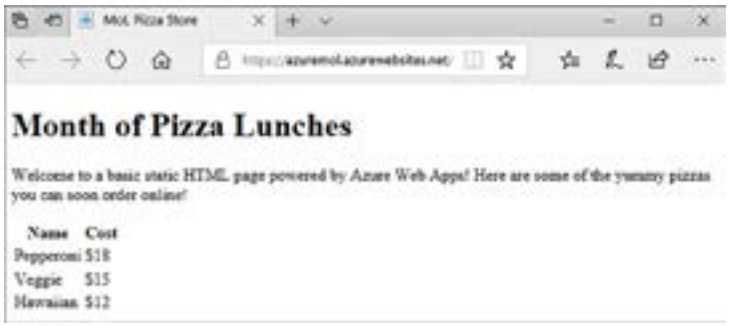


Figure 3.5 Refresh your web browser to see the default web app page replaced by the basic static HTML site from GitHub.

3.3 Viewing diagnostic logs

Now that you've seen how to create a basic web app and deploy a simple HTML site to it, what about general management? If you run into problems, it would be helpful to see the web server or application logs. To help troubleshoot your apps, you can write output from your app to these log files. Log files can be viewed in real time or written to log files and reviewed later.

Your web app largely runs by itself. There's not a lot you can do from a maintenance perspective on the underlying web host. If your application runs into problems, you may want to look at the logs to see what's going on and troubleshoot the issue. With Azure Web Apps, you configure things like the level of log messages to review, where to store the logs, and how long to keep the logs. Figure 3.6 outlines how you generate and view log files with Web Apps.

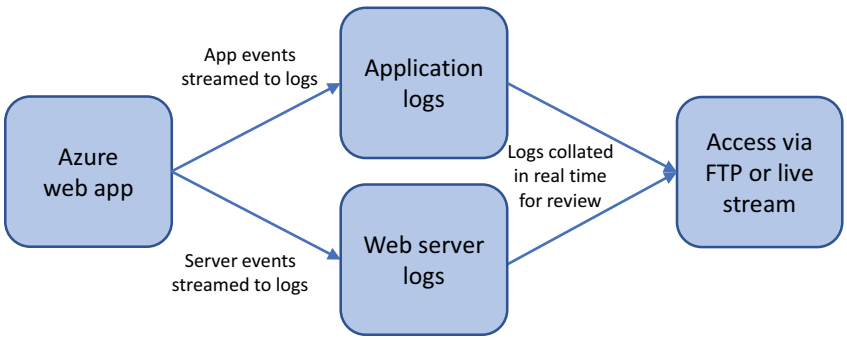


Figure 3.6 Your application can generate application logs and server logs. To review or troubleshoot problems, you can download these logs with FTP or view them in real time.

www.dbooks.org

Try it now

To configure your web app for diagnostic logs, complete the following steps:

1. In the Azure portal, select the web app you created in the preceding exercise.
2. In the Overview window, scroll down to the Monitoring section, and select App Service Logs.
3. Review the available log options, such as the verbosity and whether you want to enable failed request tracing. If you deal with the infrastructure side of Azure, you may need to work with your application developers to determine what logs they need to help troubleshoot problems. Then you can turn on the relevant logging here. Logs can be stored in the local filesystem of the web app or pushed to Azure Storage for processing with another application.
4. For now, turn on Application Logging (Filesystem). Also turn on web-server logging to the filesystem with a retention period of seven days. The default error level may not show anything if everything works right, but take care with changing to Debug or Trace, as your logs can fill up fast and make it hard to actually see what's happening! If you have a problem, gradually increase the log level until you capture enough information to troubleshoot without being overwhelmed by the log data.

If you really want to dig into the data, you can access the logs stored on the filesystem by using FTP. The FTP addresses are shown in the Download Logs section or on the Overview window for the web app. You may be thinking, "FTP is a complicated way to get diagnostic logs. Isn't there an easier way?" Why, yes, there is! In the Azure portal, right where you configured your diagnostic logs, is a Log Stream option. Can you guess what it does? Let me give you a hint: it has something to do with streaming your log files.

If you select this button in the Azure portal, you can choose between Application Logs and Web Server Logs. These logs read from the same diagnostic logs that are written to file. It can take a few minutes for the log data to show in the stream, and what's displayed depends on the log levels you specify and whether your web application generates any application events. For the basic HTML site, the stream is rather boring, but it's a great feature to have in the web browser. Figure 3.7 shows example streaming web server logs in the Azure portal.

Try it now

View the streaming log files in the Azure portal. You may need to refresh the page in your web browser a couple of times to generate activity in the logs.

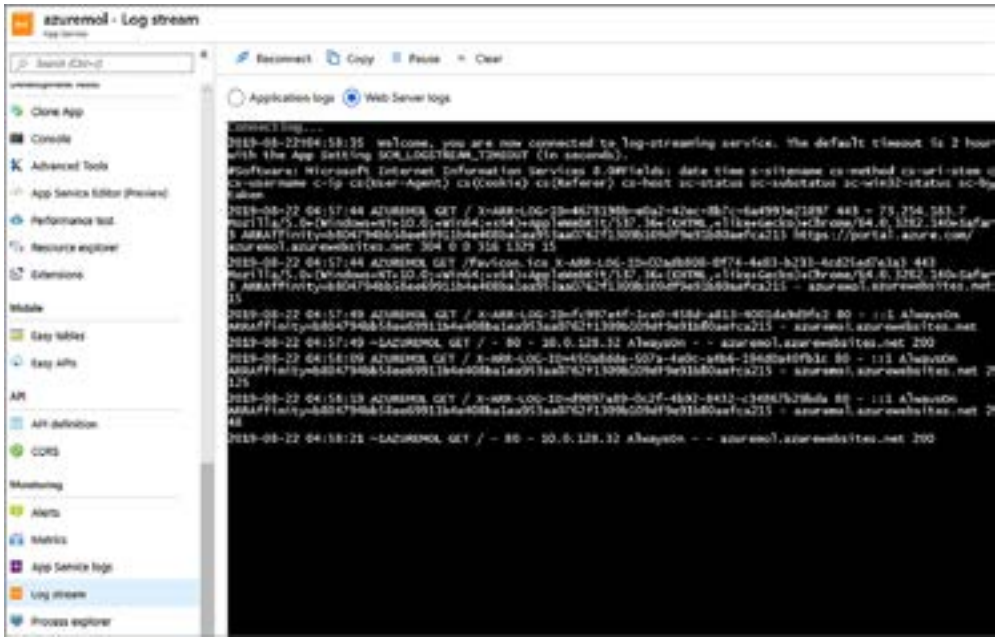


Figure 3.7 You can view the Web Apps web server log streams of live logs from your application to help verify and debug application performance. The console box on the right side on the screen shows the real-time streaming logs from your web app.

As you get more comfortable with Azure and use the Azure CLI or Azure PowerShell module, you can stream logs with these tools. Developers can also enable remote debugging with Visual Studio or configure Application Insights to allow a web application to provide telemetry to additional services for monitoring and diagnostics. The key takeaway here is that as you move toward PaaS solutions like web apps, you can still obtain crucial diagnostics logs and application data to troubleshoot and monitor the performance of your web application.

3.4 Lab: Creating and using a deployment slot

You've walked through creating a simple HTML site and pushing the page to Azure Web Apps with Git. What if you now want to add some new pizza styles and view them before you make the site live for customers to order? Check out how to use a deployment slot to provide somewhere to upload your changes, review them, and then swap them to production:

1. In your web app, choose Deployment Slots. Add a deployment slot named Dev, but don't clone any settings from the existing deployment slot.
2. When you're ready, select the staging slot from the list. The portal shows the same configuration options and logging options as the production slot, which

www.dbooks.org

shows how you can change settings in this deployment slot without affecting the live site.

3 This time, explore the options in the Azure portal for Deployment Center. You want to use Local Git for source control that uses the App Service build service for the staging slot. This happened behind the scenes when you used the Azure CLI in the earlier exercise, but you have other options in terms of where you can deploy your code from and what service builds and creates that deployment.

4 When you're done, copy the Git Clone Uri, such as https://azuremol-dev.scm.azurewebsites.net:443/azuremol.git. Note how the Git repo includes the `-dev` for the staging slot.

A sample development site is included in the GitHub samples you cloned earlier.

5 In the Azure Cloud Shell, change to the development directory as follows:

```
cd ~/azure-mol-samples-2nd-ed/03/dev
```

6 As before, initialize, add, and commit your changes in Git with the following commands:

```
git init && git add . && git commit -m "Pizza"
```

7 Create a link to the new Git repository in your staging slot with `git remote add dev`, followed by your staging slot's Git deployment URL.

8 Use `git push dev master` to push your changes to the deployment slot.

9 Select the URL to your staging slot from the Azure portal Overview window. The change isn't a big one, sure, but the page title lets you know that you're seeing the development version.

10 In the Azure portal for the web app, what do you think happens if you select the Swap button, as shown in figure 3.8? Try it, and then refresh the main page, such as https://azuremol.azurewebsites.net, in your web browser.

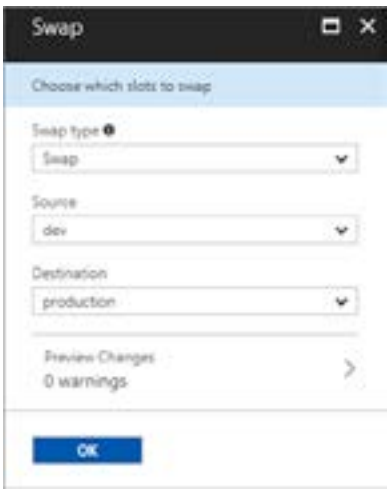


Figure 3.8 When you swap slots for a web app, you pick the source and destination instances to change. You can also preview the new look before you make the changes live.

Deployment slots, behind the scenes

When you swap slots, what was live in the *production* slot is now in the *dev* slot, and what was in *dev* is now live in *production*. Not all settings can swap, such as SSL settings and custom domains, but for the most part, deployment slots are a great way to stage and validate content before it goes live to your customers. You can also perform a swap with *preview*, which gives you the opportunity to make sure the swapped content works correctly before it's publicly live in production.

For production use in DevOps workflows, you can also configure Auto Swap. Here, when a code commit is noted in source control such as GitHub, it can trigger a build to an Azure Web Apps deployment slot. When that build is complete and the app is ready to serve content, the deployment slots swap automatically to make the code live in production. You typically use this workflow with a test environment to review the code changes first, not to publish live straight to production!

www.dbooks.org

4 Introduction to Azure Storage

We can be certain of one thing in the IT world: when things go wrong, storage and networking are inevitably to blame. I say this as someone who wore the hat of a SAN admin in one of my past lives! I was best friends with the networking team. I'm mostly joking (about being best friends), but it doesn't matter how well an application is designed and written: the foundational infrastructure pieces must be in place to support it. Bear with me in the next couple of chapters as I explore Azure Storage and Azure Networking. You may tempted to brush over these services to get to the cool stuff in the later chapters, but there's a lot of value in spending some time exploring and learning these core services. It won't make your food taste any better, but it may help your customers as they order their yummy pizzas for delivery!

This chapter looks at the different types of storage in Azure and when to use them. I also discuss redundancy and replication options for the Azure Storage service, and how to get the best performance for your applications.

4.1 Managed Disks

Years ago, server storage was expensive, slow, and overly complicated. It wasn't uncommon for a storage vendor to sell you hardware that cost hundreds of thousands of dollars and took days, or even weeks, for an army of consultants and engineers to configure. As virtualization began to take root in the data center and VMware and Hyper-V became more accepted, storage often became the bottleneck. And that's to say nothing of firmware mismatches between storage adapters in the server and on the storage array, redundant network paths failing back and forth, and solid-state disks (SSDs) being considered to be the only way to gain performance.

Has Azure magically fixed all these storage problems? Of course not! But it abstracts away 95% of these worries and leaves you to focus on building and

creating awesome experiences for your customers. This chapter covers the final 5% that you need to be aware of.

The Azure Managed Disks service simplifies the approach to VM storage. Managed disks abstract away a lot of the behind-the-scenes work to give you . . . well, a disk. That's all you really need to care about for VMs: how big and how fast they are, and what they connect to. Throughout the book, and in all your own real-world deployments, you should always use managed disks for VMs. Managed disks are the default option, and there aren't a lot of good reasons to change that behavior.

Before managed disks, you had to create a uniquely named storage account, limit the number of virtual disks you created in each, and move custom disk images manually to create VMs in different regions. These types of disks are known as *unmanaged disks* or *classic disks*. The Managed Disks service removes the need for a storage account, limits you to "only" 50,000 disks per subscription, and lets you create VMs from a custom image across regions. You also gain the ability to create and use snapshots of disks, automatically encrypt data at rest, and use disks up to 64 TiB.

Why is this important? If you run across old documentation or blog posts, they may have you create a storage account for your VMs. Stop right there! Yes, you can convert VMs from unmanaged disks to managed disks, but if you have a clean slate, look to begin each project with managed disks from the start. The use case for unmanaged disks is more to maintain backward compatibility with existing deployments, although I argue that you should look to convert those workloads to managed disks!

4.1.1 OS disks

Remember that line earlier about how, if you wanted the best performance, you had to buy SSDs? There's no magic way to get around that requirement in Azure. Sorry. The truth is that SSDs greatly outperform regular spinning disks. There are physical limits to how fast those spinning disks can . . . well, spin. The engineers at Microsoft haven't been able to bend the laws of physics just yet! There are still use cases for regular spinning disks, like low-cost archival storage, and just as in regular storage arrays, the latest technologies can provide good performance from a pool of spinning disks.

The first and main choice you need to make for an Azure VM is what type of storage to use:

- *Premium SSD disks*—Use high-performance SSD disks for optimal performance, greater IOPS, and low latency; recommended storage type for most production workloads.
- *Standard SSD disks*—Use standard SSDs and deliver consistent performance compared with hard disk drives (HDDs). These disks are great for development and testing workloads, or budget–conscious and low-demand production use.
- *Standard HDD disks*—Use regular spinning disks for more infrequent data access, such as archives and backups.

The VM size you choose helps determine what type of storage you can select. Back in chapter 2, when you created a VM, you picked a size that gave you a VM quickly. The

www.dbooks.org

default was likely something like a D2S_v3 series VM, which gave you access to premium SSD disks. How can you tell which VMs can access premium SSD disks? Look for an *s*, for *S*SD, in the VM size. There are a couple of exceptions to the rule, but that's a good pattern to follow. The following examples help you identify which VMs can access premium disks and which VMs can access standard SSDs or HDDs:

- D2S_v3, F*s*, GS, and L*s* series VMs can access premium SSD disks.
- D, A, F, and M series VMs can access only standard SSD or HDD disks.

If you select a VM size that can use premium SSD disks, you're under no obligation to do so. You could create and use standard SSD or HDD disks. By choosing premium SSD disks, you future-proof the application and give yourself the option to use high-performance SSDs as you need them without resizing your VMs and incurring a little downtime in the process. All VM sizes can use standard SSD disks.

Ephemeral OS disk

There's a special type of OS disk called an *ephemeral disk*. It's still a managed disk, of sorts, but it's local to the underlying Azure host. This fact makes an ephemeral disk really fast, with low latency.

As the data isn't written out to a remote storage array, the data may not persist during VM reboots if you move to a different underlying host. Ephemeral disks are great for stateless workloads that can handle booting up with a clean image each time and don't need to store data locally to access across reboots.

Only certain VM sizes support ephemeral disks, but there's no additional cost to using them, and they're available in all regions. You lose some functionality for things like Azure Site Recovery and Azure Disk Encryption (chapters 13 and 14, respectively), but if you want high speed, low-latency storage, check out ephemeral disks.

Try it now

How can you tell what VM sizes are available to you? In the Azure portal, open Cloud Shell. Enter the following command (feel free to use your own region):

```
az vm list-sizes --location eastus --output table
```

Remember that any size with an s gives you access to premium SSD disks.

4.1.2 Temporary disks and data disks

Now that you've figured out what level of performance you need for your applications, I'll discuss another couple of puzzle pieces. Disks are connected in two ways:

- *Temporary disks*—Every VM automatically has local SSD storage attached from the underlying host that offers a small amount of high-performance storage. Take great care how you use this temporary disk! As the name implies, this disk may not persist with the VM. If the VM moves to a new host in a maintenance event,

a new temporary disk will be attached, and any data you stored there will be lost. The temporary disk is designed to be a scratch space or application cache.

- *Data disks*—Any disks you specifically create and attach to the VM act as you'd expect in terms of partitions, filesystems, and persistent mount points. Data disks are reattached as the VM moves around the Azure data center, and they're where the bulk of your applications and data should be stored. You can still use storage spaces or software RAID to pool data disks at the VM level for even greater performance.

There's a specific type of data disk that you can attach to a VM if you need maximum performance and low latency: ultra disks. These disks are a step above premium SSD disks and are available only for data disks. Ultra disks are designed for large databases and data-intensive workloads like SAP HANA. How fast are we talking about? At the time of writing, ultra disks can be up to 64 TiB in size and provide up to 160,000 IOPS per disk with a maximum throughput of 2,000 MBps.

4.1.3 Disk-caching options

It's also important to consider the OS disk that comes with the VM. When you create a VM, you always get at least one disk: the disk where the OS itself is installed. It's tempting to use that disk to install your applications or write log files to it. Unless you run a small proof-of-concept deployment, don't run your applications on the OS disk! There's a good chance that you won't get the performance you desire.

Disks in Azure can have a caching policy set on them. By default, the OS disk has *read/write* caching applied. This type of caching typically isn't ideal for application workloads that write log files or databases, for example. Data disks, by contrast, have a default cache policy of *none*. This is a good policy for workloads that perform a lot of writes. You can also apply a *read-only* cache policy, which is better suited for application workloads that primarily read data off the disks.

In general, always attach and use data disks to install and run your applications. Even the default cache policy of none likely offers better performance than the read/write cache policy of the OS disk.

4.2 Adding disks to a VM

In this section, you'll see how to add disks to a VM as you create it. In chapter 2, you created a VM with the Azure portal. This time, use the Azure CLI to create a VM. The Azure CLI provides a quick way to create a VM and attach a data disk at the same time.

Try it now

To create a VM and see data disks in action, complete the following steps:

1. In the Azure Cloud Shell, create a resource group with `az group create`, providing a name for the resource group along with a location:

```
az group create --name azuremolchapter4 --location eastus
```

www.dbooks.org

2. Create a VM with the `az vm create` command. The final parameter, `--data-disk-sizes-gb`, lets you create a data disk along with the VM. In the end-of-chapter lab, you can connect to this VM and initialize the disks.
   - You can create a Linux or Windows VM for this exercise. If you're comfortable with Linux or want to learn how to initialize and prepare a disk for Linux, use the following command to create an Ubuntu LTS VM:

```
az vm create \
  --resource-group azuremolchapter4 \
  --name storagevm \
  --image UbuntuLTS \
  --size Standard_B1ms \
  --admin-username azuremol \
  --generate-ssh-keys \
  --data-disk-sizes-gb 64
```

   - If you're more comfortable with Windows, use the following command to create a Windows Server 2019 VM. Then you can use RDP to connect to the VM to configure the disks later:

```
az vm create \
  --resource-group azuremolchapter4 \
  --name storagevm \
  --image Win2019Datacenter \
  --size Standard_B1ms \
  --admin-username azuremol \
  --admin-password P@ssw0rd! \
  --data-disk-sizes-gb 64
```

   - It takes a few minutes to create the VM. The VM already has one data disk attached and ready for use.

What if you want to add another data disk after a few weeks or months? Use the Azure CLI again to see how to add a disk quickly. The process is the same for a Linux or Windows VM. All you do is tell Azure to create a new disk and attach it to your VM.

Try it now

Add an additional data disk to your VM as shown next.

Create a new disk with the `az vm disk attach` command. Provide a name and size for the disk. Remember the earlier discussion of standard and premium disks? In the following example, you create a premium SSD disk:

```
az vm disk attach \
    --resource-group azuremolchapter4 \
    --vm-name storagevm \
    --name datadisk \
    --size-gb 64 \
    --sku Premium_LRS \
    --new
```

Do you recognize the last part of that storage type? *LRS* means *locally redundant storage*. We'll look at redundancy options in section 4.3.3.

In two commands, you created a VM with the Azure CLI that included a data disk and simulated how to attach an additional data disk later. But just because you attached these disks doesn't mean you can write data to them immediately. As with any disk, be it a physical disk attached to an on-premises server or a virtual disk attached to a VM, you need to initialize the disk and then create a partition and filesystem. You can do that in the optional exercise in the end-of-chapter lab.

4.3 Azure Storage

Storage may not seem to be an obvious topic to examine for building and running applications, but it's a broad service that covers a lot more than you may expect. The Azure Storage service offers much more than just somewhere to store files or virtual disks for your VMs.

Take a look at what your fictional pizza company may need to build an app that processes orders from customers for takeout or delivery. The app needs a data store that holds the available pizzas, list of toppings, and prices. As orders are received and processed, the app needs a way to send messages among the application components. Then the frontend website needs mouthwatering images to show customers what the pizzas look like. As you can see in figure 4.1, Azure Storage has a variety of storage features and can cover all three of these needs.

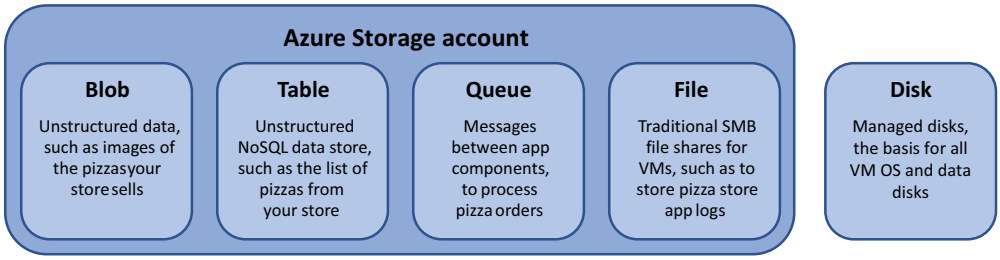


Figure 4.1 An Azure Storage account allows you to create and use a wide variety of storage features, way beyond just somewhere to store files.

- *Blob storage*—For unstructured data such as media files and documents. Applications can store data in blob storage, such as images, and then render them. You could store images of your pizzas in blob storage.
- *Table storage*—For unstructured data in a NoSQL data store. As with any debate on SQL versus NoSQL data stores, plan your application and estimate the performance requirements for processing large amounts of data. You could store the list of pizzas on your menu in table storage. Section 4.3.1 explores NoSQL in more detail.

www.dbooks.org

- *Queue storage*—For cloud applications to communicate among various tiers and components in a reliable, consistent manner. You can create, read, and delete messages that pass between application components. You could use queue storage to pass messages between the web frontend when a customer makes an order and the backend to process and bake the pizzas.
- *File storage*—For a good, old-fashioned Server Message Block (SMB) file share, accessible by both Windows and Linux/macOS platforms; often used to centralize log collection from VMs.

Azure Storage for VMs is straightforward. You create and use Azure Managed Disks, a type of virtual hard disk (VHD) that abstracts away a lot of design considerations around performance and distributing the virtual disks across the platform. You create a VM, attach any managed data disks, and let the Azure platform figure out redundancy and availability.

4.3.1 Table storage

Let's discuss a couple of types of data storage. First is *table storage*. Most people are probably familiar with a traditional SQL database such as Microsoft SQL Server, MySQL, or PostgreSQL. These are *relational databases*, made up of one or more tables that contain one or more rows of data. Relational databases are common in application development and can be designed, visualized, and queried in a structured manner—the *S* in *SQL* (for Structured Query Language).

NoSQL databases are a little different. They don't follow the same structured approach, and data isn't stored in tables in which each row contains the same fields. There are different implementations of NoSQL databases; examples include MongoDB and CouchDB. The touted advantages of NoSQL databases are that they scale horizontally (meaning that you can add more servers rather than adding more memory or CPU), can handle larger amounts of data, and are more efficient at processing those large datasets.

How the data is stored in a NoSQL database can be defined in a few categories:

- *Key-value*, such as Redis
- *Column*, such as Cassandra
- *Document*, such as MongoDB

Each approach has pros and cons from a performance, flexibility, or complexity viewpoint. An Azure storage table uses a key-value store and is a good introduction to NoSQL databases when you're used to an SQL database such as Microsoft SQL or MySQL.

You can download and install the Azure Storage Explorer at https://azure.microsoft.com/features/storage-explorer if you like to visualize the data. You don't need to do this right now. Storage Explorer is a great tool for learning what tables and queues look like in action. In this chapter, I don't want to take you too far down the rabbit hole of NoSQL databases; chapter 10 explores some cool NoSQL databases in depth

with Azure Cosmos DB. In fact, in the following exercise, you use the Cosmos DB API to connect to Azure Storage and create a table. The use of Azure tables is more an introduction to NoSQL databases than a solid example of production use.

For now, run a quick sample app to see how you can add and query data, just as you'd do with an actual application. These samples are basic but show how you can store the types of pizzas you sell and how much each pizza costs. Rather than use something large, like Microsoft SQL Server or MySQL, use a NoSQL database with Azure table storage.

Try it now

To see Azure tables in action, complete the following steps:

1. Open the Azure portal in a web browser, and then open Cloud Shell.
2. In chapter 3, you obtained a copy of the Azure samples from GitHub. If you didn't, grab a copy as follows:

   ```
   git clone https://github.com/fouldsy/azure-mol-samples-2nd-ed.git
   ```

3. Change into the directory that contains the Azure Storage samples:

   ```
   cd ~/azure-mol-samples-2nd-ed/04
   ```

4. Install a couple of Python dependencies, if they aren't already installed. Here, you install the `azurerm` package, which handles communication that allows you to create and manage Azure resources, and two `azure` packages, which are the underlying Python SDKs for Azure Cosmos DB and Storage:

   ```
   pip install --user azurerm azure-cosmosdb-table azure-storage-
   ➥queue==2.1.0
   ```

   What does `--user` mean when you install the packages? If you use the Azure Cloud Shell, you can't install packages in the core system. You don't have permissions. Instead, the packages are installed in your user's environment. These package installs persist across sessions and let you use all the neat Azure SDKs in these samples.

5. Run the sample Python application for tables. Follow the prompts to enjoy some pizza:

   ```
   python storage_table_demo.py
   ```

Snakes on a plane

Python is a widely used programming language that's often used in "Intro to Computer Science" classes. If you work mainly on the IT operations or administration side of things, think of Python as a powerful scripting language that works across OSes.

www.dbooks.org

Python isn't just for scripting; it can also be used to build complex applications. As an example, the Azure CLI that you've been using is written in Python.

I use Python for some of the samples in this book because they should work outside Cloud Shell without modification. macOS and Linux distros include Python natively. Windows users can download and quickly install Python and then run these scripts locally. Python is great for those who have little to no programming experience, as well as for seasoned developers who are familiar with other languages. The Azure documentation for Azure Storage and many other services provides support for a range of languages, including .NET, Java, and Node.js. You're not limited to using Python as you build your own applications that use tables.

The Quick Python Book, 3rd edition, by Naomi Ceder (http://mng.bz/6QZA), can help you get up to speed if you want to learn more. There's also a video-based course for *Get Programming with Python in Motion*, by Ana Bell (http://mng.bz/oPap).

4.3.2 Queue storage

Azure tables are cool when you start to dip your toes into the world of cloud application development. As you begin to build and manage applications natively in the cloud, you typically break an application into smaller components, each of which can scale and process data on its own. To allow these components to communicate and pass data back and forth, some form of message queue is typically required. Enter Azure Queues.

The Azure Queues service allows you to create, read, and then delete messages that carry small chunks of data. These messages are created and retrieved by different application components as they pass data back and forth. Azure Queues won't delete a message until an application has finished processing the message data.

Try it now

To see Azure Queues in action, run the following Python script from the same azure-samples/4 directory. Follow the prompts to see messages written, read, and deleted from the queue:

```
python storage_queue_demo.py
```

Continue the example application that handles pizza orders. You may have a frontend application component that customers interact with to order their pizzas and then a message queue that transmits messages to a backend application component that processes those orders. As orders are received, messages in the queue can be visualized as shown in figure 4.2.

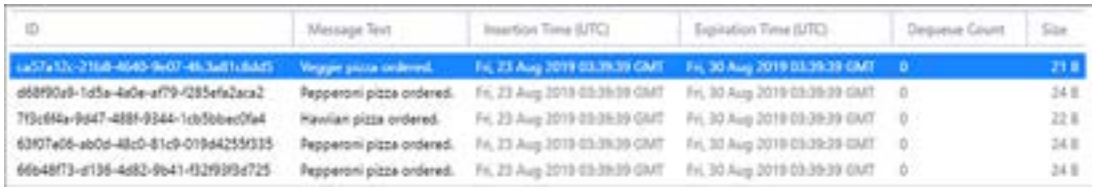


Figure 4.2 Messages are received from the frontend application component that details what pizza each customer ordered in the `Message Text` property.

As the backend application component processes each pizza order, the messages are removed from the queue. Figure 4.3 shows what the queue looks like when you have a veggie pizza in the oven and that first message is removed.

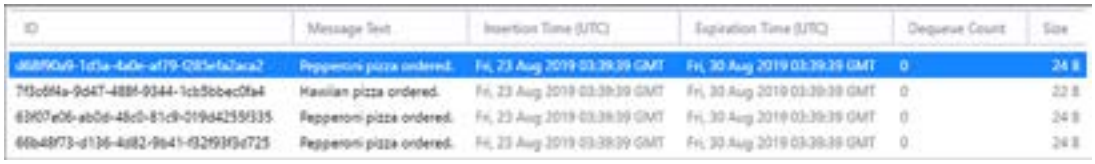


Figure 4.3 As each message is processed, it's removed from the queue. The first message shown in figure 4.2 was removed after it was processed by the backend application component.

4.3.3 Storage availability and redundancy

Azure data centers are designed to be fault-tolerant, with redundant internet connections, power generators, multiple network paths, storage arrays, and so on. But you still need to do your part when you design and run applications. With Azure Storage, you choose what level of storage redundancy you need. This level varies for each application and depending on how critical the data is. Here are the available storage-redundancy options:

- *Locally redundant storage (LRS)*—Your data is replicated three times inside the single data center in which your storage account was created. This option provides redundancy in the event of a single hardware failure, but if the entire data center goes down (rare, but possible), your data goes down with it.
- *Zone-redundant storage (ZRS)*—The next level up from LRS replicates your data three times across two or three data centers in a region (when multiple data centers exist in a region) or across regions. ZRS is also available across availability zones, which we'll explore in chapter 7.
- *Georedundant storage (GRS)*—With GRS, your data is replicated three times in the primary region in which your storage is created and then replicated three times in a paired region. The paired region is usually hundreds or more miles away. West US is paired with East US, for example; North Europe is paired with West

www.dbooks.org

Europe, and Southeast Asia is paired with East Asia. GRS provides a great redundancy option for production applications.

- *Read-access georedundant storage (RA-GRS)*—This option is the premium data-redundancy option. Your data is replicated across paired regions as in GRS, but you also have read access to the data in that secondary zone.

4.4 Lab: Exploring Azure Storage

Here's a chance to test your skills. Pick one of the following tasks to complete for your lab exercise.

4.4.1 VM-focused

If you want to log in to a VM and see that the process to initialize a disk and create a filesystem is the same as any other VM you've worked with, try one of these exercises:

1. Log in to the VM you created in section 4.2. Depending on your choice, you'll connect with SSH or RDP.
2. Initialize the disk, and create a partition.
   - In Linux, the flow is `fdisk`, `mkfs`, and then `mount`.
   - In Windows, use whatever sequence you're comfortable with—probably Disk Management > Initialize > Create Volume > Format.

4.4.2 Developer-focused

If you're more of a developer and don't want to figure out initializing data disks on a VM, go back to Cloud Shell, and explore the two Python demos that use tables and queues. Even if you're new to Python, you should be able to follow along with what's going on:

- Think of some scenarios in which you could implement tables or queues in your own applications. What would it take to build cloud-native applications with individual application components that could use queues, for example?
- Modify one of the samples that interests you to create an additional pizza menu item (if a table) or create a new pizza order message (if a queue).

5 Azure Networking basics

In chapter 4, you explored the Azure Storage service. One of the other core services for cloud applications is Azure Networking. Azure has a lot of powerful network features to secure and route your traffic on a truly global scale. These features are designed to help you focus on how to build and maintain your apps, so you don't have to worry about details like IP addresses and route tables. If you build and run an online store to handle pizza orders, it must securely transmit the customer data and process payment transactions.

This chapter examines Azure virtual networks and subnets, and discusses how to create virtual network interfaces. To secure and control the flow of traffic, you create network security groups and rules. If networking is new to you, or if it's been a while since you had to work with IP addresses and network cards, this chapter may take a little longer to work through. It has a lot of Try It Now exercises. It's worth your while to take the time to understand this chapter, however, as networking is key to many of the services in Azure.

5.1 Virtual network components

Think of how many cables are behind your computer desk or in your entertainment center. Now think of all the cables required to connect the computers on a given floor of an office building. What about the entire office building? Have you ever been in a data center or seen photos of one? Try to imagine how large the Azure data centers are. Now try to imagine dozens of Azure data centers all around the world. Math isn't my strong point, so I can't calculate how many miles and miles of network cables are used to carry all the traffic in Azure!

Network connectivity is a crucial part of modern life. In Azure, the network is central to how everything communicates. For all the thousands of physical network

www.dbooks.org

devices and miles of network cables that connect everything in an Azure data center, you work with *virtual* network resources. How? Software-defined networks. When you create a VM or a web app, a technician doesn't have to run around the Azure data center to physically connect cables for you and assign IP addresses (although that would be funny to watch!). Instead, all the network resources that define your entire network environment are logically handled by the Azure platform. Figure 5.1 shows the virtual network components you'll build as you work through this chapter.

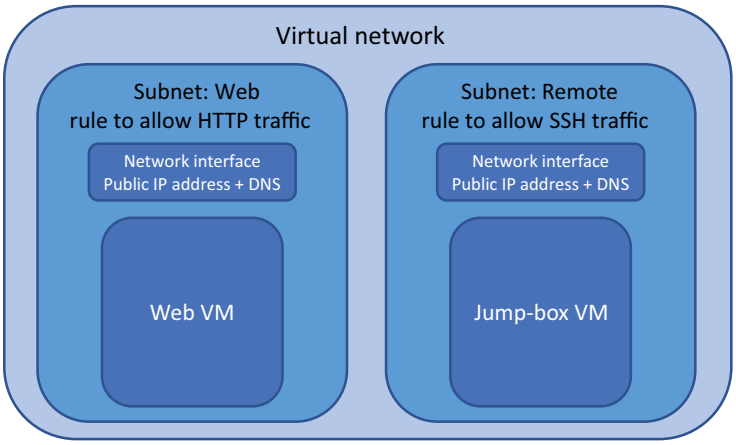


Figure 5.1 Software-defined network connections in Azure

Some of the network components are abstracted if you use PaaS resources. The main components that you use for VMs are as follows:

- Virtual networks and subnets (including IP address pools)
- Virtual network interface cards
- One or more public IP addresses
- Internal DNS name and optional public DNS names for external name resolution
- Network security groups and rules, which secure and control the flow of network traffic the way a regular firewall does

5.1.1 Virtual networks and subnets

When you created a VM in chapter 2, you didn't have to adjust any network parameters. The Azure platform can create these resources for you with default names and IP address scopes. In this section, you'll create the network resources ahead of time and see how they come together for a VM.

Try it now

Networking is often easier to visualize when you see it in action. You're going to use the Azure portal to get started, which ends up taking quite a few separate steps, but you'll see the power of the Azure CLI later in the chapter.

Don't worry too much about how to use your own address spaces or custom DNS names right now. To build out your virtual network and subnet, complete the following steps:

1. Open the Azure portal, and select Create a Resource in the top-left corner of the dashboard.
2. Select Networking from the list of Marketplace services, and then choose Virtual Network.
3. Enter a name for the virtual network, such as `vnetmol`.
4. To give yourself a little more room to play with, change the address space to `10.0.0.0/16`.

IP address ranges

Virtual networks span a certain range of IPs—an address space. If you've ever seen an IP address, you may have noticed the subnet mask, which is often something like `255.255.255.0`. This subnet mask is often used in a short form that specifies how big that range is, such as `/24`.

The Azure portal defaults to a `/24` address space. You want to increase the number of additional IP ranges here without too much network knowledge, so you increase the address space to `/16`. You don't give this type of IP address straight to a VM; in the next step, you'll create a subnet that covers a smaller section of this address space.

If network address spaces are totally foreign to you, don't worry. For the most part, you won't deal with them on a daily basis. Sensible Azure governance may work the same way it does in your existing on-premises IT world; one group of folks may manage the Azure virtual networks, and you drop your applications into a precreated space.

5. Create a resource group, such as `azuremolchapter5`, and then select an Azure region close to you.
6. Provide a subnet name, such as a `websubnet`, and enter the subnet address range `10.0.1.0/24`. This address range is part of the wider virtual network you specified earlier. Later, you'll add another subnet.
7. Look at some of the other options, such as distributed denial of service (DDoS) protection, service endpoints, and Azure Firewall. Leave the defaults for now, but I hope this example gives you some hints about what's possible beyond a basic virtual network.
8. When you're ready, create the virtual network and subnet.

www.dbooks.org

5.1.2 Virtual network interface cards

Now that you've created a virtual network and subnet, you need to connect a VM. Just as you do with a regular desktop PC, laptop, or tablet, you use a network interface card (NIC) to connect to the virtual network. And no, there's no free Wi-Fi! But there are VM sizes in Azure that currently provide up to eight NICs with speeds of up to 32 Gbps. Even if I were good at math, I could tell you that these figures add up to some serious bandwidth!

You may wonder why you'd create each of these resources ahead of time. You can do all this when you create a VM. That's true, but take a step back and think about network resources as long-lived resources.

Network resources exist separately from VM resources and can persist beyond the lifecycle of a given VM. This concept allows you to create the fixed network resources and create, delete, and create again a VM that maintains the same network resources, such as IP addresses and DNS names. Think of a lab VM or a development-and-test environment. You can reproduce the exact same environment quickly because only the VM changes.

Try it now

To create a NIC, complete the following steps:

1. In the Azure portal, select Create a Resource in the top-left corner of the dashboard.
2. Search for and select Network Interface, and then select Create.
3. Provide a name for your network interface, such as `webvnic`; then select the virtual network and subnet you created in the previous exercise.
4. I talked about long-lived resources earlier; now you can see how they work. Create a static IP address assignment that uses the address `10.0.1.4`.

TIP Why .4? What about the first three addresses in the address space? Azure reserves the first three IP addresses in each range for its own management and routing. The first usable address you can use in each range is .4.

5. Don't create a network security group for now; you'll come back to this in a few minutes. If you're one of the cool kids who knows all about IPv6, you can check the Private IP Address (IPv6) box and provide a name; otherwise, stick with IPv4.
6. Select the existing resource group from the previous exercise, and then choose to create the NIC in the same region as the virtual network.
7. When you're ready, create the NIC.

Role separation in Azure

You don't have to create other compute resources within the same resource group as your virtual network. Think back to the concept of Azure governance, discussed earlier. You may have a group of network engineers who manage all the virtual network resources in Azure. When you create resources for your applications, such as VMs, you create and manage them in your own resource groups.

Later chapters discuss some of the security and policy features in Azure that allow you to define who can access and edit certain resources. The idea is that if you don't know, or don't want to know, a great deal about the network resources, you can connect to what's given to you, and that's it. The same applies to other engineers or developers; they may be able to see your application resources but not edit or delete them.

This kind of governance model in Azure is nice, but take care to avoid the trap of working in silos. In large enterprises, it may be inevitable that you're constrained along department lines. But one of the big advantages of cloud computing providers like Azure is speeding time to deployment of applications, because you don't have to wait for physical network resources to be cabled up and configured. Plan to have the Azure network resources created and configured, and you should be able to create and manage your application resources seamlessly.

5.1.3 Public IP address and DNS resolution

No one can access your resources yet, because no public IP addresses or DNS names are associated with them. Again, follow the principle of long-lived resources to create a public IP address and public DNS name and then assign them to your network interface.

Try it now

To create a public IP address and DNS entry for your network interface, complete the following steps:

1. In the Azure portal, select Create a Resource in the top-left corner of the dashboard.
2. Search for and select Public IP Address, and then select Create.
3. Create a basic SKU and IPv4 address. Standard SKUs and IPv6 addresses are for use with load balancers (chapter 8). Don't worry too much about the differences right now.
4. Enter a name, such as webpublicip, that uses a dynamic assignment.

IP address assignment types

A dynamic assignment allocates a public IP address when the VM is started. When the VM is stopped, the public IP address is deallocated. There are a couple of important points here:

www.dbooks.org

- You won't have a public IP address until you assign it to a VM and start it.
- The public IP address may change if you stop, deallocate, and start the VM.

A *static* assignment is a public IP address allocated without an associated VM, and that address won't change. This assignment is useful when you're using an SSL certificate mapped to an IP address or a custom DNS name and record that point to the IP address.

Right now, you're using a single VM. For production use, you'll ideally run your application on multiple VMs with a load balancer in front of them. In that scenario, the public IP address is assigned to the load balancer and typically creates a static assignment at that point.

5 Enter a unique DNS name. This name forms the fully qualified domain name (FQDN) for your resource that's based on the Azure region you create it in. If you create a DNS name called azuremol in the East US region, for example, the FQDN becomes azuremol.eastus.cloudapp.azure.com.

DNS entries

What about a custom DNS name? The default FQDN isn't exactly user friendly! Use a static public IP address, and then create a CNAME record in your registered DNS zone. You retain control of the DNS record and can create as many entries as you wish for your applications.

As an example, in the manning.com DNS zone, you might create a CNAME for azuremol that points to a static public IP address in Azure. A user would access azuremol.manning.com to get your application. This address is a lot more user-friendly than webmol.eastus.cloudapp.azure.com!

6 Select the existing resource group from the previous exercise, and then choose to create the public IP address in the same region as the virtual network.

7 When you're ready, create the public IP address.

8 Associate the public IP address and DNS name label with the network interface you created in section 5.1.2. Browse to and select Resource Group on the navigation bar on the left side of the Azure portal. Then choose the resource group in which you created your network resources, such as azuremolchapter5.

9 Select your public IP address from the list of resources, and then choose Associate.

10 Choose to associate with a network interface (but notice what else you can associate the public IP address with); then choose the network interface you created, such as webvnic.

After a few seconds, the public IP address window updates to show that the IP address is now associated with your network interface. If you selected Dynamic as the

assignment type, the IP address is still blank, as shown in figure 5.2. Remember that a public IP address is allocated when an associated VM is powered on.

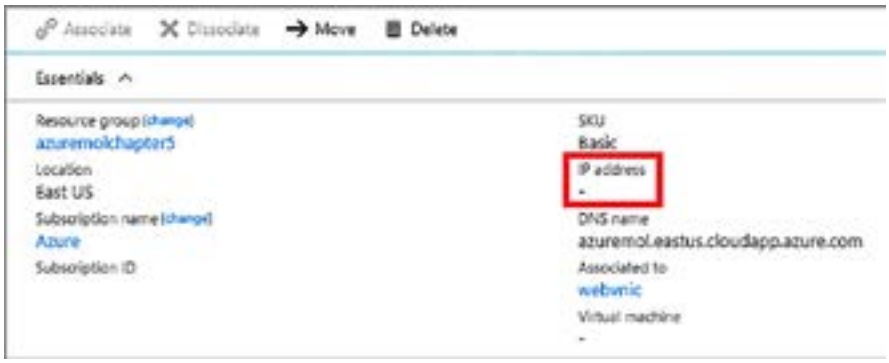


Figure 5.2 The public IP address is associated with a network interface. With a dynamic assignment, no public IP address is shown until a VM is created and powered on.

5.2 Securing and controlling traffic with network security groups

Pop quiz time: Should you connect a VM to the internet without a firewall to control and restrict the flow of traffic? If you answered, "Sure, why not?" maybe you should take the rest of your lunch break to read a little about network security on the Wild Wide Web!

I hope your answer was a resounding, "*No!*" Unfortunately, there's too much potential for your VM to come under an automated cyberattack soon after you turn it on. You should always follow best practices to keep the OS and application software up to date, but you don't even want the network traffic to hit your VM if it's not necessary. A regular macOS or Windows computer has a built-in software firewall, and every (competent) on-premises network I've seen has a network firewall between the internet and the internal network. In Azure, firewall and traffic rules are provided by network security groups.

5.2.1 Creating a network security group

In Azure, an NSG logically applies a set of rules to network resources. These rules define what traffic can flow in and out of your VM. You define what ports, protocols, and IP addresses are permitted, and in which direction. These groups of rules can be applied to a single network interface or an entire network subnet. This flexibility allows you to finely control how and when the rules are applied to meet the security needs of your application.

Figure 5.3 shows the logic flow of an inbound network packet as it passes through an NSG. The same process would apply for outbound packets. The Azure host doesn't

www.dbooks.org

differentiate between traffic from the internet and traffic from elsewhere within your Azure environment, such as another subnet or virtual network. Any inbound network packet has the inbound NSG rules applied, and any outbound network packet has the outbound NSG rules applied.

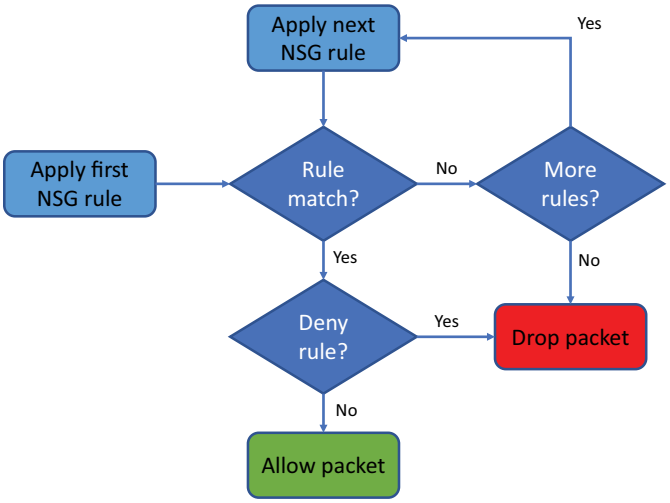


Figure 5.3 Inbound packets are examined, and each NSG rule is applied in order of priority. If an Allow or Deny rule match is made, the packet is either forwarded to the VM or dropped.

Here's what happens to each network packet:

1. The first NSG rule is applied.
2. If the rule doesn't match the packet, the next rule is loaded until there are no more rules. Then the default rule to drop the packet is applied.
3. If a rule matches, check whether the action is to deny the packet. If so, the packet is dropped.
4. Otherwise, if the rule is to allow the packet, the packet is passed to the VM.

Next, you'll create an NSG so these concepts start to make sense.

Try it now

To create a network security group, complete the following steps:

1. In the Azure portal, select Create a Resource in the top-left corner of the dashboard.
2. Search for and select Network Security Group, and then select Create.
3. Enter a name, such as `webnsg`, and choose to use the existing resource group.

That's it! The bulk of the configuration for an NSG comes when you create the filtering rules. Section 5.2.2 discusses how you do that and put your NSG to work.

5.2.2 Associating a network security group with a subnet

The NSG doesn't do much to protect your VMs without any rules. You also need to associate it with a subnet, the same way you associated your public IP address with a network interface earlier. You'll associate your NSG with a subnet first.

Try it now

To associate your virtual network subnet with your network security group, complete the following steps:

1. Browse to and select Resource Group on the navigation bar on the left side of the Azure portal. Then choose the resource group you created your network resources in, such as azuremolchapter5.
2. Select your NSG, such as webnsg.
3. On the left side, in the Settings options, are Network Interfaces and Subnets. Choose Subnets.
4. Select the Associate button; select the virtual network and network subnet you created earlier; and then select OK to associate your NSG with the subnet.

   The flexibility of NSGs means that you can associate multiple subnets, across various virtual networks, with a single NSG. The mapping is one-to-many, which allows you to define core network security rules that apply to a wide range of resources and applications.

   Now you can see what your NSG looks like and what default rules are applied.
5. On the left side of your NSG, select Inbound Security Rules. No security rules are listed—at least, none that you've created.
6. Select Default Rules to see what the Azure platform creates for you, as shown in figure 5.4.



Figure 5.4 Default security rules are created that permit internal virtual network or load-balancer traffic but deny all other traffic.

www.dbooks.org

Three default rules have been created for you. These rules are important to understand:

- *AllowVnetInBound*—Allows any traffic that's internal to the virtual network. If you have multiple subnets in your virtual network, the traffic isn't filtered by default and is allowed.
- *AllowAzureLoadBalancerInBound*—Allows any traffic from an Azure load balancer to reach your VM. If you place a load balancer between your VMs and the internet, this rule ensures that the traffic from the load balancer can reach your VMs, such as to monitor a heartbeat.
- *DenyAllInBound*—The final rule that's applied. Drops any inbound packets that make it this far. If there are no previous Allow rules, this rule drops all traffic by default. All you need to do is allow any specific traffic you want; the rest is dropped.

The priority of an NSG rule is important. If an Allow or Deny rule is applied, no additional rules are applied. Rules are applied in ascending numerical priority order; a rule with a priority of 100 is applied before a rule with a priority of 200, for example.

As in previous discussions of the governance of Azure resources, these NSG rules may already be created for you and applied to a given subnet. You create your VMs and run your applications, and someone else manages the NSGs.

It's important to understand how the traffic flows in case something goes wrong. A couple of tools in Azure can help you determine why traffic may not reach your application when you think it should!

5.2.3 Creating network security group filtering rules

Now that you have your NSG associated with the network subnet and we've reviewed the default rules, create a basic NSG rule that allows HTTP traffic.

Try it now

To create your own rules with the network security group, complete the following steps:

1. To create an NSG rule from the previous Azure portal window, select Add in the Inbound Security Rules section.
2. You have two options for creating rules: Basic and Advanced. To create prebuilt rules quickly, select Basic at the top of the window.
3. Choose HTTP from the Service drop-down menu. Many default services are provided, such as SSH, RDP, and MySQL. When you select a service, the appropriate port range is applied—in this case, port 80. The default action on basic rules allows the traffic.

4. A priority value is assigned to each rule. The lower the number, the higher the priority. Accept the default low priority, such as 100.
5. Accept the default name or provide your own; then select OK.

5.3 Building a sample web application with secure traffic

So far, you've created a virtual network and subnet. Then you created a network interface and associated a public IP address and DNS name label. You created an NSG and applied it to the entire subnet, and you created an NSG rule to allow HTTP traffic. You're missing one thing: the VM.

5.3.1 Creating remote access network connections

In production, you shouldn't open remote access, such as SSH or RDP, to VMs that run your applications. Typically, you have a separate jump-box VM that you connect to from the internet, and you access additional VMs over the internal connection. So far, you've created all the virtual network resources in the Azure portal. Let's switch over to the Azure CLI to see how quickly you can create these resources from the command line.

Try it now

You created the first NSG in the Azure portal. To create another NSG with the Azure CLI, complete the following steps:

1. Select the Cloud Shell icon at the top of the Azure portal dashboard. Make sure that the Bash shell opens, not PowerShell.
2. Create an additional NSG in the existing resource group. As in earlier chapters, the backslashes (\) in the following command examples are to help with line breaks; you don't have to type them if you don't want to. Provide a name, such as remotensg:

```
az network nsg create \
  --resource-group azuremolchapter5 \
  --name remotensg
```

3. Create an NSG rule in the new NSG that *allows* port *22*. Provide the resource group and NSG you created in the previous step along with a name, such as allowssh:

```
az network nsg rule create \
  --resource-group azuremolchapter5 \
  --nsg-name remotensg \
  --name allowssh \
  --protocol tcp \
  --priority 100 \
  --destination-port-range 22 \
  --access allow
```

4. Create a network subnet for your remote VM. Provide a subnet name, such as remotesubnet, along with an address prefix inside the range of the virtual

www.dbooks.org

network, such as `10.0.2.0/24`. You also attach the NSG you created in step 3 to the subnet, such as remotensg:

```
az network vnet subnet create \
  --resource-group azuremolchapter5 \
  --vnet-name vnetmol \
  --name remotesubnet \
  --address-prefix 10.0.2.0/24 \
  --network-security-group remotensg
```

Three commands are all it takes to create a subnet, create an NSG, and create a rule. Can you start to see the power of the Azure CLI? Azure PowerShell is equally powerful, so don't feel like you must create all resources in the Azure portal. As you move forward in the book, you'll use the Azure CLI rather than the portal in most cases.

5.3.2 Creating VMs

With all the network components in place, you're ready to create two VMs. One VM is created in the subnet that allows HTTP traffic so that you can install a web server. The other VM is created in the subnet that allows SSH so that you have a jump box to further secure your application environment and begin to replicate a production deployment. Figure 5.5 reminds you what you're building.

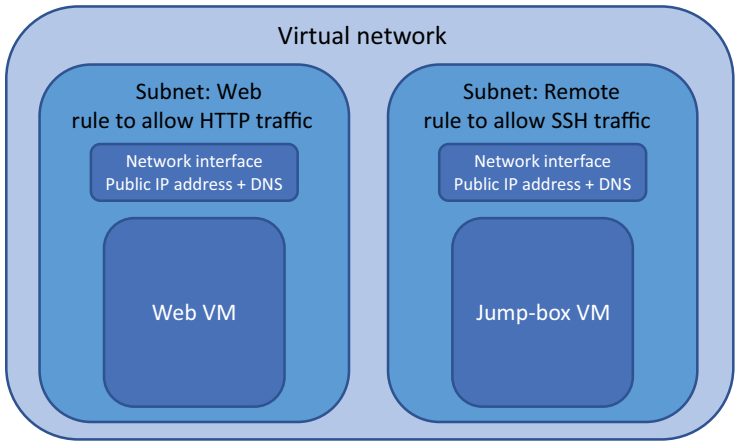


Figure 5.5 You're bringing together two subnets, NSGs, rules, network interfaces, and VMs. This example is close to a production-ready deployment in which one VM runs the web server and is open to public traffic, and another VM in a separate subnet is used for remote connections to the rest of the application environment.

When you create a VM, you can provide the virtual network interface that you created earlier. If you didn't specify this network resource, the Azure CLI creates a virtual network, subnet, and NIC for you, using built-in defaults. That's great for creating a VM

quickly, but you want to follow the principle of using long-lived network resources that another team may manage and in which you'll create your VMs.

Try it now

To use the Azure CLI to create your web server and jump-box VMs, complete the following steps:

1. Create the first VM for your web server, and provide a name, such as webvm. Attach the network interface, such as webvnic, and enter an image, such as UbuntuLTS. Provide a username, such as azuremol. The final step, `--generate-ssh-keys`, adds to the VM the SSH keys you created in chapter 2:

```
az vm create \
  --resource-group azuremolchapter5 \
  --name webvm \
  --nics webvnic \
  --image UbuntuLTS \
  --size Standard_B1ms \
  --admin-username azuremol \
  --generate-ssh-keys
```

2. Create the second VM for the jump box. This example shows how you can use an existing subnet and NSG, and let the Azure CLI create the network interface and make the appropriate connections. You create a public IP address, such as remotepublicip, as part of this command:

```
az vm create \
  --resource-group azuremolchapter5 \
  --name remotevm \
  --vnet-name vnetmol \
  --subnet remotesubnet \
  --nsg remotensg \
  --public-ip-address remotepublicip \
  --image UbuntuLTS \
  --size Standard_B1ms \
  --admin-username azuremol \
  --generate-ssh-keys
```

The output from both commands shows a public IP address. Make a note of these IP addresses. In the next exercise, if you try to SSH to your first VM for the web server, it fails. Why? You can SSH to the remote VM because you created an NSG rule to allow only HTTP traffic to the web VM.

5.3.3 Using the SSH agent to connect to your VMs

I need to introduce one piece of magic with SSH that allows you to use your jump box correctly and connect to the web VM over the Azure virtual network: the *SSH agent*. This agent applies only to Linux VMs, so if you mainly work with Windows VMs and Remote

www.dbooks.org

Desktop Protocol connections, don't worry if the SSH talk is new. You can create RDP connections to Windows VMs from your jump box with the local remote credentials or with domain credentials if you configure the server appropriately.

An SSH agent can store your SSH keys and forward them as needed. Back in chapter 2, when you created an SSH public-key pair, I talked about the public and private keys. The private key is something that stays on your computer. Only the public key is copied to the remote VMs. Although the public key was added to both VMs you created, you can't just SSH to your jump box and then SSH to the web VM. Why? That jump box doesn't have a copy of your private key. When you try to make the SSH connection from the jump box, it has no private key to pair up with the public key on the web VM for you to authenticate.

The private key is something to safeguard, so you shouldn't take the easy way out by copying the private key to the jump box. Any other users who access the jump box could potentially get a copy of your private key and then impersonate you anywhere that key is used. Here's where the SSH agent comes into play.

If you run the SSH agent in your Cloud Shell session, you can add your SSH keys to it. To create your SSH connection to the jump box, you specify the use of this agent to tunnel your session. This technique allows you to effectively pass through your private key for use from the jump box without ever copying the private key. When you SSH from the jump box to the web VM, the SSH agent tunnels your private key through the jump box and allows you to authenticate.

Try it now

To use SSH with your jump-box VM, complete the following steps:

1. In Cloud Shell, start the SSH agent as follows:

   ```
   eval $(ssh-agent)
   ```

2. Add the SSH key you created in chapter 2 to the agent as follows:

   ```
   ssh-add
   ```

3. SSH to your jump-box VM. Specify the use of the SSH agent with the `-A` parameter. Enter your own public IP address that was shown in the output when you created the jump-box VM:

   ```
   ssh -A azuremol@<publicIpAddress>
   ```

4. This is the first time you've created an SSH connection to the jump-box VM, so accept the prompt to connect with the SSH keys.
5. Remember how you created a static private IP address assignment for the web VM in section 5.1.2? This static address makes it a lot easier to SSH to it. SSH to the web VM as follows:

   ```
   ssh 10.0.1.4
   ```

6 Accept the prompt to continue the SSH connection. The SSH agent tunnels your private SSH key through the jump box and allows you to successfully connect to the web VM. Now what? Well, you have a lab to see this work!

5.4 Lab: Installing and testing the LAMP web server

You've already done the hard work throughout the chapter. This quick lab reinforces how to install a web server and lets you see the NSG rule on your VM in action:

1 *Install a basic Linux web server.* Think back to chapter 2 when you created an SSH connection to the VM and then installed the LAMP web server package with `apt`. From the SSH connection to your web VM created in section 5.3.2, install and configure the default LAMP web stack.

2 *Browse to the default website.* When the LAMP web stack is installed, open your web browser to the DNS name label you entered when you created a public IP address in section 5.1.3. In the example, that was azuremol.eastus.cloudapp.azure.com. You can also use the public IP address that was output when you created the web VM. Remember, though: that public IP address is different from the jump-box VM you SSHed to!

www.dbooks.org

Part 2

High availability and scale

Okay, let's start to have some fun! Now that you understand the core resources in Azure, you can dive into areas such as redundancy, load balancing, and geographical distribution of applications. This part is where things get exciting, and the topics you learn about should start to show solutions and best practices that you can use in real-world deployments. Azure has some awesome features to replicate data globally, distribute customer traffic to the closest instance of your application, and automatically scale based on demand. These features are the power of cloud computing and where you bring true value to your work.

www.dbooks.org

6 Azure Resource Manager

Most days, you want to spend as little time as possible on how you deploy an application environment and get on with actual deployment. In many IT environments, there's a movement toward development and operations teams that collaborate and work closely together, with the *DevOps* buzzword being thrown around a lot at conferences and in blogs.

There's nothing inherently new or groundbreaking about the DevOps culture, but often, different teams didn't work together as they should. Modern tools have spurred the DevOps movement, with continuous integration/continuous delivery (CI/CD) solutions that can automate the entire deployment of application environments based on a single code check-in by a developer. The operations team is usually the one that builds and maintains these CI/CD pipelines, which allows much quicker tests and deployments of application updates for developers.

The Azure Resource Manager deployment model is central to how you build and run resources, even though you probably haven't realized it yet. Resource Manager is an approach to building and deploying resources as much as the automation processes and templates that drive those deployments. In this chapter, you'll learn how to use Resource Manager features such as access controls and locks, consistent template deployments, and automated multitier rollouts.

6.1 The Azure Resource Manager approach

When you created a VM or web app in previous chapters, you first created a resource group as the core construct to hold all your resources. A resource group is central to all resources: a VM, web app, virtual network, or storage table can't exist outside a resource group. But the resource group is more than just a place to organize your

resources—a lot more. This section looks at the underlying Azure Resource Manager model and shows why it's important as you build and run applications.

6.1.1 Designing around the application lifecycle

Ideally, you won't build an application and never maintain it. You usually have updates to develop and deploy, new packages to install, new VMs to add, and additional web app deployment slots to create. You may need to make changes in the virtual network settings and IP addresses. I mentioned in previous chapters that your virtual networks in Azure may be managed by a different team. You need to start to think about how you run on a large, global scale, and in terms of the application lifecycle and management.

You have a couple of main approaches for grouping resources in Azure:

- *All resources for a given application in the same resource group*—As shown in figure 6.1, this approach works well for smaller applications and for development and test environments. If you don't need to share large networking spaces and can manage storage individually, you can create all the resources in one place, and then manage updates and configuration changes in one operation.

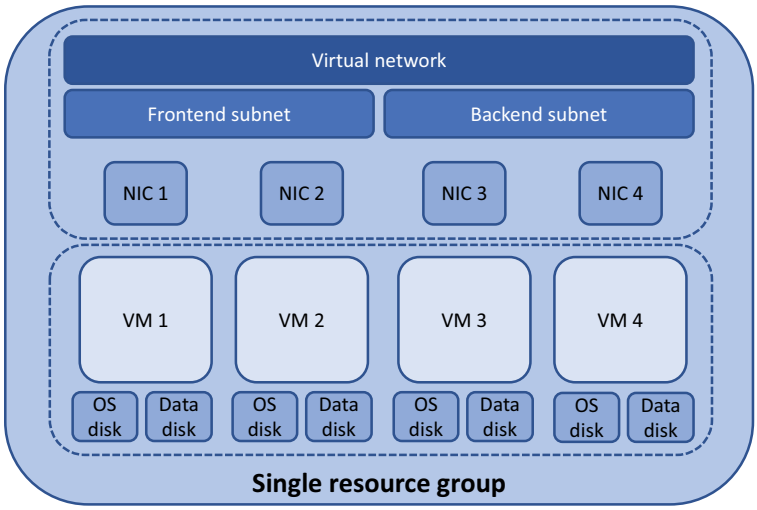


Figure 6.1 One way to build an application in Azure is to create all the resources related to that application deployment in the same resource group and manage them as one entity.

- *Like-minded resources grouped by function in the same resource group*—As shown in figure 6.2, this approach is more common in larger applications and environments. Your application may exist in a resource group with only the VMs and supporting application components. Virtual network resources and IP addresses may exist in a different resource group, secured and managed by a different group of engineers.

www.dbooks.org

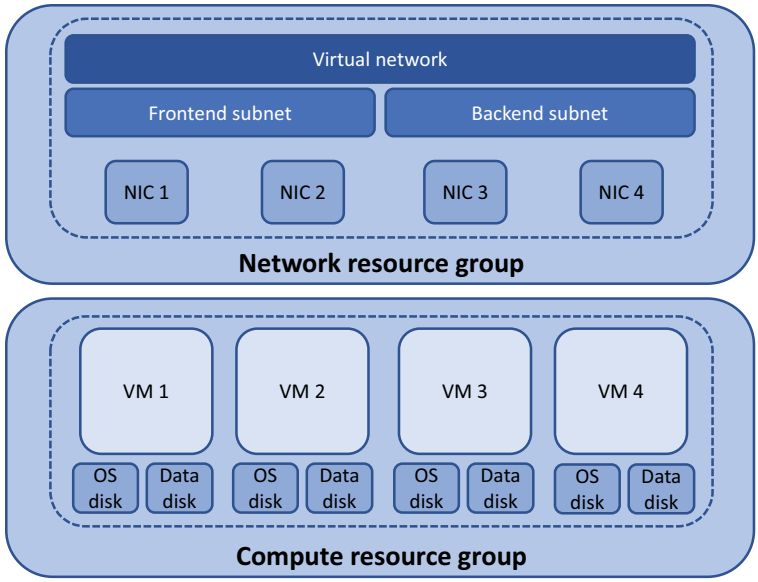


Figure 6.2 An alternative approach is to create and group resources based on their role. A common example is that all core network resources are in a separate resource group from the core application compute resources. The VMs in the compute resource group can access the network resources in the separate group, but the two sets of resources can be managed and secured independently.

Why are there different approaches? The answer isn't all down to job security and the lovely silos some teams like to work in. It's about how you need to manage the underlying resources. In smaller environments and applications where all the resources exist in the same resource group, you're responsible for everything in that environment. This approach is also well suited to development and test environments in which everything is packaged together. Any changes you make in the virtual network affect only your application and resource group.

The reality is that networks don't change often. The address ranges are often well defined and planned so that they can coexist across Azure and office locations around the world. Logically, it often makes sense to place the network components in their own resource group. The network is managed separately from the application. Storage may be managed and updated separately in the same way. There's nothing inherently wrong with dividing resources in this way as long as the IT staff doesn't get stuck in a silo mentality, resulting in a lack of cooperation.

For your applications, the division of resources can also be a benefit, in that you're largely free to make changes and updates as you wish. Precisely because you don't have the network components in your resource group, you don't need to worry about them when you make application updates.

6.1.2 Securing and controlling resources

Each resource can have different security permissions applied to it. These policies define who can do what. Think about it—do you want an intern restarting your web app or deleting the VM data disks? And do you think that your good buddies over in the network team want you to create a new virtual network subnet? Probably not. In Azure, there are four core roles you can assign to resources, much like file permissions:

- *Owner*—Complete control, basically an administrator
- *Contributor*—Full management of the resource except making changes in the security and role assignments
- *Reader*—Ability to view all information about the resource but make no changes
- *User access administrator*—Ability to assign or remove access to resources

Role-based access control (RBAC) is a core feature of Azure resources that automatically integrates with the user accounts across your subscriptions. Think of file permissions on your normal computer. The basic file permissions are read, write, and execute. When these permissions are combined, you can create different sets of permissions for each user or group on your computer. As you work with network file shares, permissions are common tools for controlling access. RBAC in Azure works along the same lines to control access to resources, just like file permissions on your local computer or network share (figure 6.3).

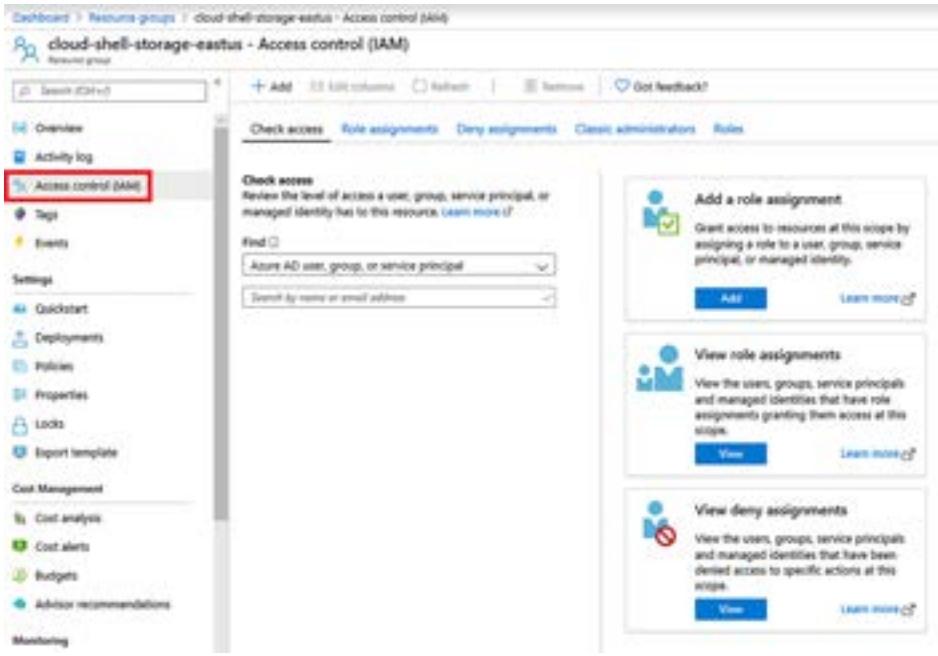


Figure 6.3 The access control for each Azure resource lists the current assignments. You can add assignments or select Roles to see information about what permission sets are available.

www.dbooks.org

Try it now

Open the Azure portal in a web browser, and then select any resource you have, such as cloud-shell-storage. Choose the Access Control (IAM) button, as shown in figure 6.3. Review the current role assignments. Look at how to add a role assignment, and explore all the available role assignments. The information icon for each role shows what permissions are assigned.

As you explore the available roles, you may notice several resource-specific roles, including the following:

- Virtual Machine Contributor
- Website Contributor
- Network Contributor

Can you guess what those roles mean? They take the core platform Contributor role and apply it to a specific resource type. The use case here goes back to that concept of how you manage like-minded resources. You might be assigned the Virtual Machine Contributor or Website Contributor role. Then any VMs or web apps created in that resource group would be available for you to manage. But you couldn't manage network resources, which may be in a different resource group entirely.

6.1.3 Protecting resources with locks

The permissions-based approach of RBAC is great for limiting who can access what. But mistakes can still happen. There's a reason why you typically don't log on to a server as a user with administrative, or root, permissions. One wrong keystroke or mouse click, and you could mistakenly delete resources. Even if you have backups (You do have backups, right? And you test them regularly?), it's a time-consuming process that may mean lost productivity or revenue to the business. In chapter 13, you'll learn more about the ways the Azure Backup, Recovery, and Replication services protect your data.

Another feature built into the Resource Manager model is resource locks. Each resource can have a lock applied that limits it to read-only access or prevents delete operations. The delete lock is particularly helpful, because it can be all too easy to delete the wrong resource group. When you start a delete operation, there's no going back or canceling the operation after the Azure platform has accepted your request.

For production workloads, I suggest that you implement locks on your core resources to prevent deletes. These locks are only at the Azure resource and platform levels, not for the data within your resources. You could delete files within a VM or drop a table in a database, for example. The Azure resource locks would apply only if you tried to delete the entire VM or Azure SQL database. The first time a lock kicks in and prevents the wrong resource group from being deleted, you'll thank me!

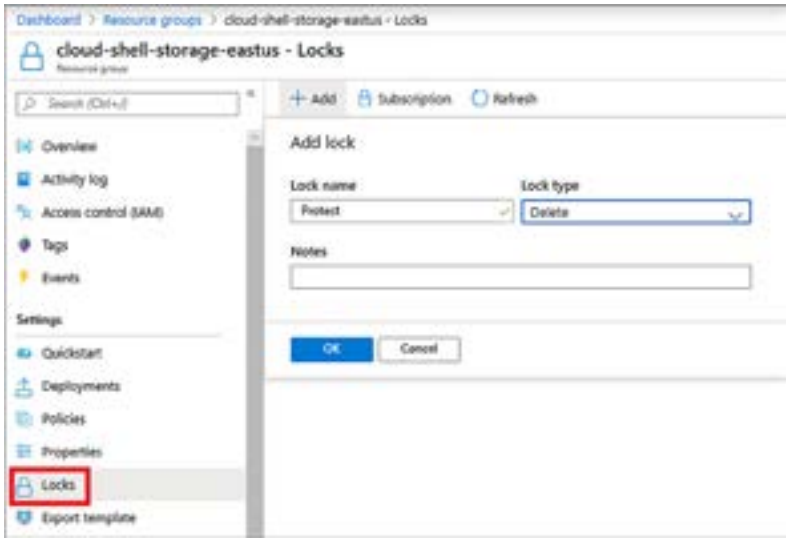


Figure 6.4 Create a resource lock in the Azure portal.

Try it now

To see Azure resource locks in action, as shown in figure 6.4, complete the following steps:

1. Open the Azure portal in a web browser, and select any resource group you have, such as cloud-shell-storage.
2. Choose Locks on the left side of the portal.
3. Enter a Lock Name, such as Protect; choose Delete from the Lock Type drop-down menu; and choose OK. Your new lock appears in the list.
4. Select Overview for the resource group, and then try to delete the resource group. You need to enter the resource group name to confirm that you want to delete it (which is also a good mental prompt to make sure that you have the right resource to delete!).
5. When you choose the Delete button, review the error message that's displayed to see how your lock prevented Azure from deleting the resource.

6.1.4 Managing and grouping resources with tags

One final feature in the Azure Resource Manager model that I want to bring up is *tags*. There's nothing new or special about how you tag resources in Azure, but this management concept is often overlooked. You can apply tags to a resource in Azure that describe properties such as the application it's part of, the department responsible for it, or whether it's a development or production resource.

www.dbooks.org

You can target resources based on tags to apply locks or RBAC roles, or to report on resource costs and consumption. Tags aren't unique to a resource group and can be reused across your subscription. Up to 50 tags can be applied to a single resource or resource group, so you have a lot of flexibility in how you tag and then filter tagged resources.

Try it now

To see Azure resource tags in action, complete the following steps:

1. Open the Azure portal in a web browser, and then select any resource, such as cloud-shell-storage. Although you can tag a resource group itself, don't pick a resource group for this exercise.
2. With your resource selected, choose the Tags button, as shown in figure 6.5.
3. Enter a Name, such as `workload`, and a Value, such as `development`.
4. Select Save.
5. Open Cloud Shell.
6. To filter resources based on tags, use `az resource list` with the `--tag` parameter. Use your own name and value as follows:

```
az resource list --tag workload=development
```



Figure 6.5 You can create up to 50 name:value tags for each Azure resource or resource group.

6.2 Azure Resource Manager templates

So far, you've created a small number of Azure resources at a time. To do this, you used the Azure portal or Azure CLI. Although I haven't shown you Azure PowerShell, I did talk about it in chapter 1, and it's available in Cloud Shell. Maybe you've tried it without

me. That's okay; I don't feel left out! As I mentioned in chapter 1, Azure has tools that let you choose what's most comfortable for you and the environment you work in.

The downside of using the portal or CLI or PowerShell commands is that you must click a bunch of buttons in the web browser or type lines of commands to build your application environment. You could create scripts to do all these things, but then you'd have to build logic to handle the creation of multiple resources at the same time or the order to create resources in.

A script that wraps Azure CLI or PowerShell commands starts to move in the right direction in terms of how you should build and deploy application environments—not just in Azure, but across any platform. There's a move toward infrastructure as code (IaC), which is nothing new if you've been around IT for a while. All it means is that you don't rely on a human to type commands and follow a set of steps; rather, you programmatically create your infrastructure from a set of instructions. Manual deployments introduce a human element that can often lead to minor misconfigurations and differences in the final VMs, as shown in figure 6.6.

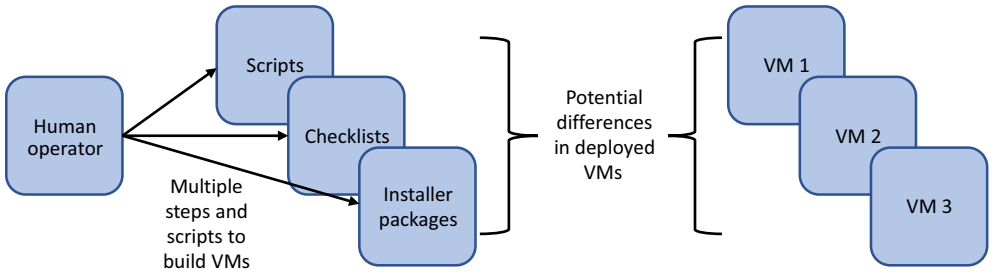


Figure 6.6 Humans make mistakes, such as mistyping a command or skipping a step in a deployment. You can end up with slightly different VMs at the end of the output. Automation is often used to remove the human operator from the equation and instead create consistent, identical deployments every time.

Even when you have scripts, you still need someone to write them, maintain them, and keep them updated as new versions of the Azure CLI or PowerShell modules are released. Yes, there are sometimes breaking changes in the tools to accommodate new features, although they're rare.

6.2.1 Creating and using templates

Resource Manager templates can help reduce human error and reliance on manually written scripts. Templates are written in JavaScript Object Notation (JSON), an open standard and cross-platform approach that allows editing them in a basic text editor. With templates, you can create consistent, reproducible deployments that minimize errors. Another built-in feature of templates allows the platform to understand dependencies and can create resources in parallel where possible to speed deployment time. If you create three VMs, for example, there's no need to wait for the first VM to

www.dbooks.org

finish deploying before you create the second; Resource Manager can create all three VMs at the same time.

As an example of dependencies, if you create a virtual NIC, you need to connect it to a subnet. Logically, the subnet must exist before you can create the virtual NIC, and the subnet must be part of a virtual network, so that network must be created before the subnet. Figure 6.7 shows the chain of dependencies in action. If you try to write a script yourself, you must carefully plan the order in which resources are created, and even then, you must build in logic to know when the parent resources are ready and you can move on to the dependent resources.

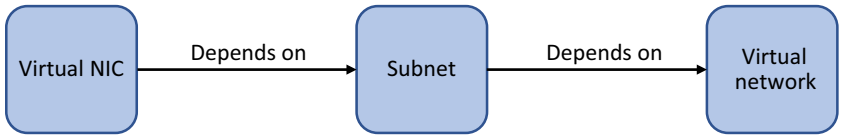


Figure 6.7 Azure Resource Manager handles dependencies for you. The platform knows the order in which to create resources and has awareness of the state of each resource without the use of handwritten logic and loops like those you must use in your own scripts.

Want to know something cool? You've already used Resource Manager templates—in chapter 2 and in the very first VM you created. As you create a VM in the portal or the Azure CLI, under the hood, a template is programmatically created and deployed. Why? Well, why reinvent the wheel and go through the process of building all that logic for the deployments? Let Azure Resource Manager do it for you!

Here's what a section of a Resource Manager template looks like. The following listing shows the section that creates a public IP address, just as in earlier examples when you created a VM.

Listing 6.1 Creating a public IP address in a Resource Manager template

```
{
       "apiVersion": "2019-04-01",
       "type": "Microsoft.Network/publicIPAddresses",
       "name": "publicip",
       "location": "eastus",
       "properties": {
         "publicIPAllocationMethod": "dynamic",
         "dnsSettings": {
           "domainNameLabel": "azuremol"
    }
  }
},
```

Even if JSON is new to you, it's written in a (somewhat) human-readable format. You define a resource type—in this example, `Microsoft.Network/publicIPAddresses`. Then you provide a name, such as `publicip`, and a location, such as `eastus`. Finally,

you define the allocation method—`dynamic`, in this example—and a DNS name label, such as `azuremol`. These parameters are the same ones you provided when you used the Azure portal or CLI. If you use PowerShell, guess what? You're prompted for the same parameters.

The difference with the template is that you didn't have to enter any information. All the information was in the code. "Great," you might think, "but what if I want to use different names each time?" As with a script, you can assign names dynamically by using parameters and variables:

- *Parameters* are values that you're prompted for. They're often used for user credentials, the VM name, and the DNS name label.
- *Variables* can be preassigned values, but they're also adjusted each time you deploy the template, such as the VM size or virtual network name.

Try it now

To see a complete Resource Manager template, open a web browser to the GitHub repo at http://mng.bz/QyWv.

6.2.2 Creating multiples of a resource type

As you build your templates, try to think ahead about how you may need to grow your applications in the future. You may need only a single VM when you first deploy your application, but as demand for the application grows, you may need to create additional instances.

In a traditional scripted deployment, you create a `for` or `while` loop to create multiple resource types. Resource Manager has this functionality built in! There are more than 50 types of functions in Resource Manager, just like in most programming and scripting languages. Common Resource Manager functions include `length`, `equals`, `or`, and `trim`. You control the number of instances to create with the `copy` function.

When you use the `copy` function, Resource Manager creates the number of resources you specify. Each time Resource Manager iterates over the create operation, a numerical value is available for you to name resources in a sequential fashion. You access this value with the `copyIndex()` function. The example in listing 6.1 created a single public IP address. The example in listing 6.2 uses the same `Microsoft.Network/publicIPAddresses` resource provider type but creates two public IP addresses. You use copy to define how many addresses you want to create and `copyIndex()` to name the addresses sequentially.

Listing 6.2 Creating multiple public IP addresses with `copy`

```
{
        "apiVersion": "2019-04-01",
        "type": "Microsoft.Network/publicIPAddresses",
        "name": "[concat('publicip', copyIndex())]",
```

www.dbooks.org

```
        "copy": {
          "count": 2
        }
        "location": "eastus",
        "properties": {
          "publicIPAllocationMethod": "dynamic",
        }
},
```

You also use the `concat` function to combine the public IP address name and the numerical value of each instance you create. After this template is deployed, your two public IP addresses are called `publicip0` and `publicip1`. These names aren't super-descriptive, but this basic example shows how you can use a numbering convention as you create multiple resources with the `copy` function.

6.2.3 Tools to build your own templates

So I'll confess: although Resource Manager templates are neat and among the main ways I suggest that you build and deploy applications in Azure, you still need to write the templates. A couple of tools simplify this task for you, and hundreds of sample templates are available from Microsoft and third parties. In fact, one of the best ways to learn how to create and use templates is to examine the quick-start templates Microsoft makes available in its samples repo at https://github.com/Azure/azure-quickstart-templates.

If you want to roll up your sleeves and start to write you own templates, I recommend two main tools. The first is Visual Studio Code, a free, open source, multiplatform editor (https://code.visualstudio.com). Along with some built-in functionality such as source control and GitHub integration, extensions are available that can automatically build the different sections, or providers, for the resources to build up a template, as shown in figure 6.8. If you download and install VS Code, choose View > Extensions, and then search for *Azure*.



Figure 6.8 Many extensions are available in Visual Studio Code to improve and streamline the way you create and use Azure Resource Manager templates.

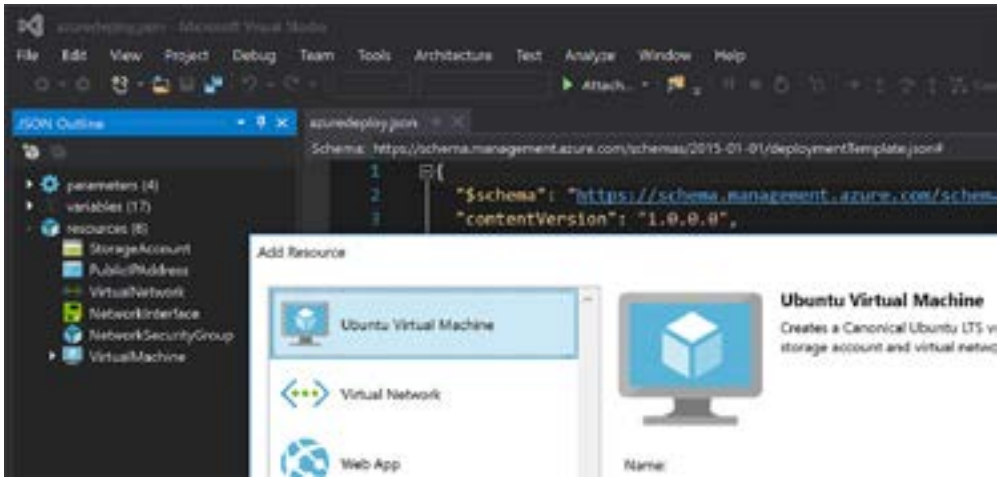


Figure 6.9 With Visual Studio, you can graphically build templates and explore JSON resources.

A more graphical way to build Azure Resource Manager templates is to use the full Visual Studio editor, shown in figure 6.9. There are versions for both Windows and macOS, but you need a separate license to use the editor. A Community Edition is available, but take care if you build templates within your company: you typically need a licensed version. Consult your license experts, because Visual Studio targets application developers.

You can, of course, use a basic text editor. Part of the reason why Azure Resource Manager templates are written in JSON is that it removes the need for any special tools. There's a learning curve to working with JSON, which is why I recommend that you explore the quick-start templates in the Azure samples repo. Take care with indentation, trailing commas, and the use of parentheses, brackets, and braces!

Life on Mars

There are third-party tools and other ways to use templates in Azure. HashiCorp provides many open source tools and solutions for cloud computing, one of which is Terraform. With Terraform, you define all the resources you want to build in much the same way that you do a native Azure Resource Manager template. You can define dependencies and use variables, too. The difference is that Terraform is technically cross-provider. The same constructs and template approach can be used across Azure, Google Cloud, AWS, and vSphere, for example. The difference is the provisioners that you use for each resource.

Is it truly a "one template for any provider" approach? No, not at all. Terraform is also an application that parses your template and then communicates with the relevant cloud provider, such as Azure. You get zero editing capabilities, let alone graphical tools, to build your template. You pick an editor and write the template by hand.

www.dbooks.org

Again, the best way to learn Terraform is to explore its documentation and example templates.

The reason why I bring up this topic relates to the concept of choice in Azure. If you find Azure Resource Manager templates written in JSON a little too cumbersome, explore a product like Terraform instead. But don't give up on template-driven Resource Manager deployments. To achieve those reproducible, consistent deployments at scale, templates are the best approach, so find a good template-driven approach that works for you.

6.2.4 Storing and using templates

So you love the idea of Azure Resource Manager templates, and you've installed Visual Studio or Code to write your own templates. How do you store and deploy them? In the end-of-chapter lab, you deploy a template from the Azure samples repository on GitHub. This repository is public, and you may not want to make your application templates available to the entire world.

There are a couple of common methods for storing Resource Manager templates privately:

- Use a private repository or network file share within your organization.
- Use Azure Storage to centrally store and secure templates for deployment.

There's no right or wrong way to store and deploy templates. You have the flexibility to use whatever processes and tools are already in place. The advantage of using a repository is that you typically have some form of version control, so you can ensure consistent deployments and review the history of your templates if necessary. The only limitation is that when you deploy the template, you need to provide the appropriate credentials to access the shared location. This authentication process can vary, such as providing a username or access token as part of the URL to a template in a repository or providing a shared access signature (SAS) token if you use Azure Storage.

Public repositories such as GitHub can also be great ways to learn and share. I do suggest that you keep your production templates stored privately, but if you create a neat template for a lab environment or to try some new features, sharing on GitHub gives a little something back to the IT community and may help others who want to do the same deployments that you do. And as you begin to build your own templates, be sure to check out what templates already exist so that you don't start from scratch and reinvent the wheel every time!

6.3 Lab: Deploying Azure resources from a template

All this theory about deployment models and approaches is great, but you'll (ideally) start to see the benefits and efficiency when you use templates for real:

1. Go to the Azure quick-start samples on GitHub (https://github.com/Azure/azure-quickstart-templates), and find one that interests you. A good place to start is a simple Linux or Windows VM.

2 Built into the GitHub samples are buttons that deploy straight to Azure. When you find a template you like, select Deploy to Azure, as shown in figure 6.10, and follow the steps in the portal. The process is much the same as that of creating a VM, but only a few prompts are required to complete the required parameters. All the other resources are created for you and abstracted away.

3 The final step in deploying your template is accepting the license agreement and then choosing Purchase. You're creating Azure resources when you deploy a template, so choosing Purchase means that you agree to pay for the costs of those Azure resources.

One of the basic templates, such as a simple Linux or Windows VM, costs about the same as any other VM you've created so far. Make sure that you delete the resource group when your deployment is finished, just as you'd clean up after any other exercise.

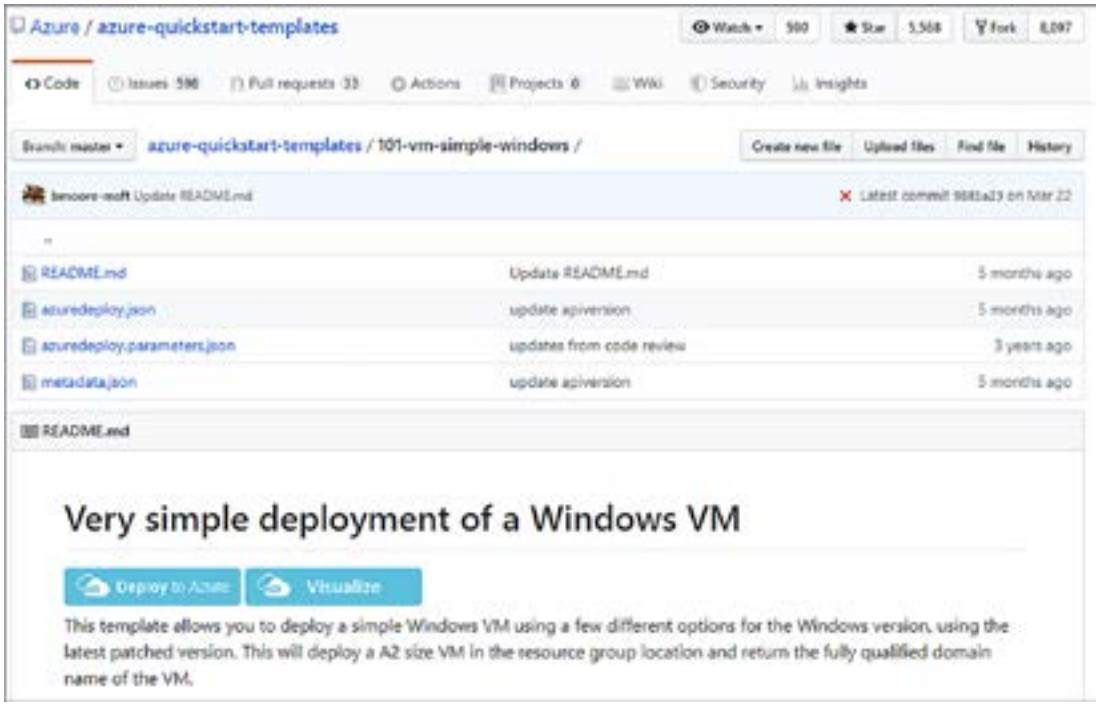


Figure 6.10 For each Resource Manager template in the GitHub sample repo, there's a Deploy to Azure button. If you select this button, the Azure portal loads, and the template is loaded. You're prompted for some basic parameters, and the rest of the deployment is handled by the template.

www.dbooks.org

Parameters in templates

As discussed in section 6.2.1, you can use parameters and variables in your templates. Remember, parameters are values that you're prompted for, and variables are dynamic values that can be applied throughout a template. The values you're prompted for (parameters) vary from template to template. So depending on which quick-start template you select, you may be prompted for one or two values, or you may have to provide seven or eight.

As you design your templates, try to anticipate how you and other users may want to reuse the template as you deploy applications. You can provide a default value and limit what values are allowed. Take care with these default and allowable values, though; otherwise, you may constrain users too much and force them to create their own templates. Where possible, try to build reusable core templates that have enough flexibility.

4 When your template has deployed, go back to GitHub, and examine the azuredeploy.json file. This file is the Azure Resource Manager template that you used to create and deploy the sample. See whether you can understand the different resource types and configurations that were applied. As you work with more Azure resource types and templates, the JSON format will become easier to understand. Honest!

7 High availability and redundancy

I can't count the number of times that something in IT has failed me. I've had a laptop hard drive crash the day before a conference, a smoking power supply in an email server, and failed network interfaces on a core router. And don't even get me started on OS, driver, and firmware updates! I'm sure that anyone who works in IT would love to share horror stories about situations they've had to deal with—usually problems that happened late at night or at a critical time for the business. Is there ever such a thing as a good failure, and at a nice time?

If you anticipate failures in IT, you learn to plan and design your applications to accommodate problems. In this chapter, you'll learn how to use Azure high availability and redundancy features to minimize disruptions caused by maintenance updates and outages. This chapter builds a foundation for the next two or three chapters as you start to move from an application that runs on a single VM or web app to one that can scale and be globally distributed.

7.1 The need for redundancy

If you want customers to trust you for their important pizza business, you must supply applications that are accessible whenever they need them. Most customers won't look for "hours of operation" on a website, especially if you work in a global environment and customers could be from all over the world. When they're hungry, they want to eat!

Figure 7.1 shows a basic example of an application that runs on a single VM. Unfortunately, this application creates a single point of failure. If that one VM is unavailable, the application is unavailable, which leads to customer unhappiness and hunger.

www.dbooks.org

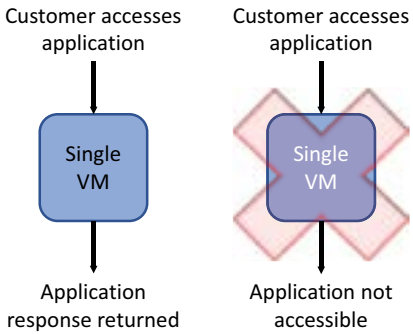


Figure 7.1 If your application runs on a single VM, any outage on that VM causes the application to be inaccessible. This could mean that customers take their business elsewhere or, at the least, aren't satisfied with the service you provide.

If you drive a car, there's a good chance that there's a spare tire in case you have a puncture. If you a use a laptop or tablet, there's a good chance that you plug the device into a charger in case the battery runs out in the middle of work. At home or your apartment, do you have spare light bulbs in case one of the lights goes out? What about a flashlight or candles in case there's a power outage?

Most people like to have some form of redundancy or backup plan, both in day–to-day life and, especially, in IT. If you're ready to switch over to a spare car tire or light bulb, you can handle outages and failures with minimal interruption. If you design and build your applications for redundancy, you provide a high level of availability to your customers that minimizes or even hides any interruptions the application encounters. All Azure data centers are built for high availability. Backup power supplies, multiple network connections, and storage arrays with spare disks are just some of the core redundancy concepts that Azure provides and manages for you. All the redundancy Azure provides may not help if you run your application on a single VM. To give you flexibility and control over how to make your application highly available, two main features for IaaS workloads are available:

- *Availability Zones*—Lets you distribute VMs across physically isolated segments of an Azure region to further maximize your application redundancy. Zones can also provide high availability to network resources such as public IP addresses and load balancers.
- *Availability Sets*—Lets you logically group VMs to distribute them across a single Azure data center and minimize disruption from outages or maintenance updates.

For most new application deployments in Azure, I suggest that you plan to use Availability Zones. This approach offers flexibility in how to distribute your application and provides redundancy to the network resources that are often central to how customers ultimately access the underlying VMs. To see how each of these approaches works, let's discuss them in more depth.

7.2 Infrastructure redundancy with Availability Zones

Availability Zones are physically separate data centers that operate on independent core utilities such as power and network connectivity. Each Azure region that supports Availability Zones provides three zones. You create your resources in and across these zones. Figure 7.2 shows how Azure resources can be distributed across Availability Zones.

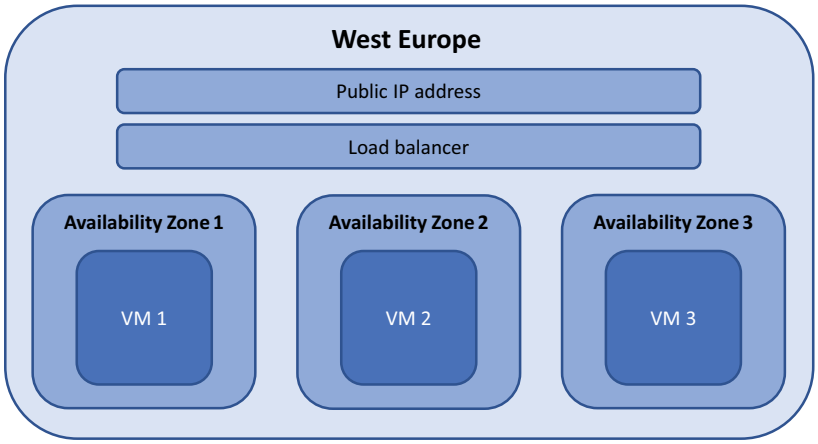


Figure 7.2 An Azure region can contain multiple Availability Zones: physically isolated data centers that use independent power, network, and cooling. Azure virtual network resources such as public IP addresses and load balancers can span all zones in a region to provide redundancy for more than just the VMs.

With Availability Zones, your applications can tolerate an entire Azure data center going offline. Sure, it would take a major event for this situation to occur, but it's still possible!

In large application deployments, you may create more than one VM in each Availability Zone. Multiple VMs in an Availability Zone are automatically distributed across the available hardware within the zone. There's nothing you need to configure or can control. Even if a maintenance update or equipment failure inside a zone were to affect all your VMs that run in the zone, remember that zones are physically isolated from each other; the VMs in another zone would continue to run.

Now, if you feel particularly unlucky, could all your VMs in different zones experience maintenance updates at the same time? Yes, but that's unlikely. Zones within a region have staggered update cycles. Updates are performed across one zone; once they're complete, updates are performed across the next zone. Availability zones provide a high level of abstraction and redundancy, and you should look at your application across the entire deployment, not just where VMs in one zone reside.

The inclusion of the virtual network resources in Availability Zones is a lot more important than it may seem at first. Figure 7.3 shows what would happen if the data

www.dbooks.org

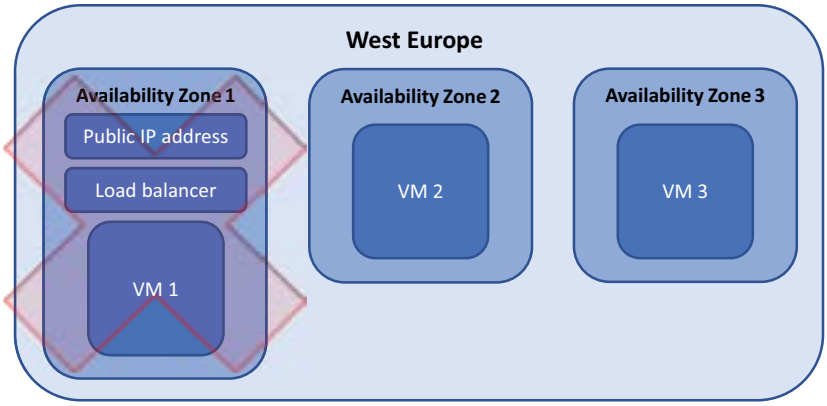


Figure 7.3 When network resources are attached to a single Azure data center, or zone, an outage in that facility causes the entire application to be unreachable by the customer. It doesn't matter that the other VMs continue to run in other zones. Without the network connectivity to distribute traffic from your customers, the whole application is unavailable.

center became unavailable for network resources such as a public IP address and load balancer that run across Availability Zones.

I'll talk more about load balancers in chapter 8, but for now, all you need to understand is that the load balancer distributes traffic across all available VMs that are attached to it. The VMs report their health status at set intervals, and the load balancer no longer distributes traffic to a VM that reports as being unavailable. With a load balancer that works across Availability Zones, an outage in one Azure data center causes those VMs to become unavailable and be taken out of the load-balancer rotation.

A public IP address that spans Availability Zones provides a single entry point for customers to reach your load balancer and then be distributed to an available VM. In an application deployment where that public IP address resides in a single Azure data center, if that data center encounters a problem, no customer can access the public IP address. The customer can't use your application, even if there are VMs available to serve customer requests.

Resources that can use Availability Zones include both zonal services and zone-redundant services:

- *Zonal services* are for things like VMs, a public IP address, or a load balancer. The whole resource itself runs within a given zone and can operate by itself if another zone is unavailable.
- *Zone-redundant services* are for resources that can automatically replicate across zones, such as zone-redundant storage and SQL databases. The whole resource isn't running within a given zone; rather, its data is distributed across zones so that it continues to be available if one zone has a problem.

Availability Zone support is available for more than 20 Azure services across more than ten regions. The number of services and regions that integrate with Availability Zones continues to grow. However, given the region limitations, there may be times when Availability Zone support isn't available for core resources like VMs. In those cases, there's another type of VM redundancy that you can use in any region that we look at in section 7.2.1: Availability Sets.

7.2.1 Creating network resources across an Availability Zone

To start to see some of this availability and redundancy in action, let's create some common resources, such as a public IP address and load balancer, and then VMs. The goal here is to see that you don't have to do much configuration at all to take advantage of Availability Zones in Azure. These are simple examples, but they form the core of most application environments you'd deploy.

Public IP addresses and load balancers can be created in either of two available tiers: basic and standard. The primary difference is that the standard tier allows the network resource to use Availability Zones. By default, a standard public IP address or load balancer is automatically zone redundant. There's no additional configuration for you to complete. The Azure platform centrally stores the metadata for the resource within the region you specify and makes sure that the resource continues to run if one zone becomes unavailable.

Don't worry too much about what happens with the load balancer and network resources right now. Remember what I said at the start; these next two or three chapters build on one another. In chapter 8, we'll dive into load balancers, and all this should start to make more sense.

Try it now

To create network resources that are redundant across Availability Zones, complete the following steps:

1. Select the Cloud Shell icon at the top of the Azure portal dashboard.
2. Create a resource group, such as `azuremolchapter7az`:

```
az group create --name azuremolchapter7az --location westeurope
```

3. Create a standard public IP address in your resource group. By default, a *basic* public IP address would be created and assigned to a single zone. The `--sku standard` parameter instructs Azure to create a redundant, cross-zone resource:

```
az network public-ip create \
  --resource-group azuremolchapter7az \
  --name azpublicip \
  --sku standard
```

www.dbooks.org

4 Create a load balancer that spans Availability Zones. Again, a basic load balancer would be created by default and assigned to a single zone, which isn't the high-availability design you want for your applications. Specify a *standard* SKU to create a zone-redundant load balancer, as follows:

```
az network lb create \
  --resource-group azuremolchapter7az \
  --name azloadbalancer \
  --public-ip-address azpublicip \
  --sku standard
```

7.2.2 Creating VMs in an Availability Zone

To create a VM in an Availability Zone, you specify which zone to run the VM in. To deploy many VMs, you ideally create and use a template. The template defines and distributes the zones for each of the VMs. As customer demand for your online pizza store grows, you can also update the template with the number of VMs you now want and then redeploy the template. The new VMs are distributed across zones for you automatically, and there's no need to manually track which zones the VMs run in. In the end-of-chapter lab, you'll use a template to create and distribute multiple VMs automatically. To see the logical process to specify a zone for a VM, let's create a VM and manually specify the zone.

Try it now

To create a VM in an Availability Zone, complete the following steps:

1 In the Azure portal, select the Cloud Shell icon at the top of the dashboard.

2 Create a VM with the `az vm create` command you've used in previous chapters. Use the `--zone` parameter to specify zone 1, 2, or 3 for the VM to run in. The following example creates a VM named `zonedvm` in zone 3:

```
az vm create \
  --resource-group azuremolchapter7az \
  --name zonedvm \
  --image ubuntults \
  --size Standard_B1ms \
  --admin-username azuremol \
  --generate-ssh-keys \
  --zone 3
```

It takes a few minutes to create the VM. When the process is finished, the output from the command indicates the zone that the VM runs in. You can also view this information with the `az vm show` command:

```
az vm show \
--resource-group azuremolchapter7az \
--name zonedvm \
--query zones
```

> **NOTE** The examples in these "Try it now" exercises are simple but are designed to show you that zones require little configuration to use. You didn't integrate the zone-redundant load balancer and VM, but in chapter 8, you'll build out a more usable application environment that's distributed across Availability Zones. The goal here is to show you that the Azure platform handles the redundancy and distribution of your resources so you can focus on the application itself.

7.3 VM redundancy with Availability Sets

Availability zones are great when designing for redundancy on a wider set of resources that make up your applications and workloads. I recommend that, where possible, you use them for new workloads. However, there are times when you don't necessarily need to make all the resources zone redundant. Or you may want to create VMs in an Azure region that doesn't currently have Availability Zone support.

If you only want to provide redundancy for VMs, Availability Sets have you covered. They're proven, reliable, and available across all regions. Availability Sets contain a logical group of VMs that indicates to the Azure platform that the underlying hardware those VMs run on needs to be carefully selected. If you create two VMs that run on the same physical server, and one server fails, both of those VMs go down. With potentially tens of thousands or more physical servers in an Azure data center, it's highly unlikely that you'd have both of those VMs on the same server, but it's possible! It may be not a failure but a maintenance update that causes the physical server to be briefly unavailable.

What if your VMs run in the same rack, attached to the same storage or networking equipment? You're back to the single point of failure discussed at the start of the chapter.

Availability Sets allow the Azure platform to create your VMs across logical groups called *fault domains* and *update domains*. These logical domains let the Azure platform understand the physical boundaries of hardware groups to make sure your VMs are evenly distributed across them. If one piece of hardware has a problem, only a few VMs in your Availability Set are affected. Or if there are maintenance updates to be applied to the physical hardware, the maintenance affects only a few of your VMs. The relationship of physical hardware to logical fault domains and update domains inside an Availability Set is shown in figure 7.4.

Availability Zones do the same kind of distribution under the hood, but it's abstracted away and isn't exposed. Even with Availability Sets, there's not a lot you can configure. But it's helpful to know what's happening behind the scenes.

7.3.1 Fault domains

A *fault domain* is a logical group of hardware in an Azure data center. It contains hardware that shares power or network equipment. You don't control what these fault domains are, and there's nothing for you to configure at the VM level. The Azure platform tracks what fault domains your VMs are placed in and distributes new

www.dbooks.org

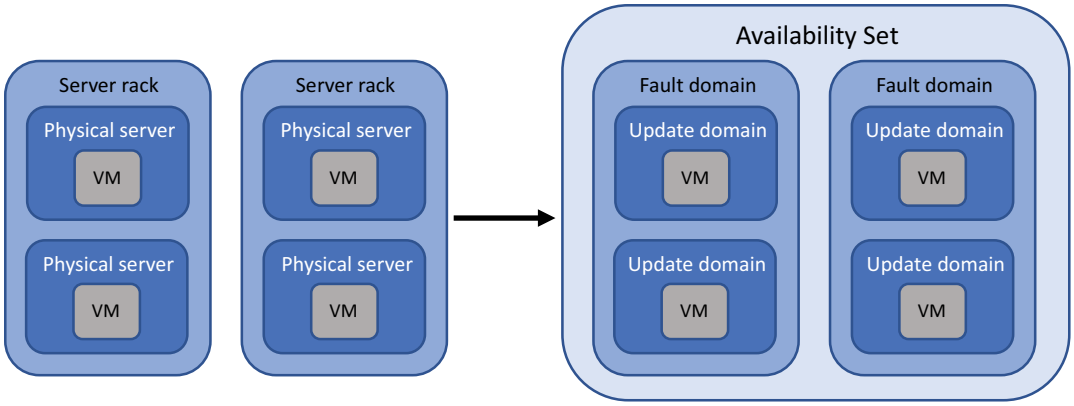


Figure 7.4 Hardware in an Azure data center is logically divided into update domains and fault domains. These logical domains allow the Azure platform to understand how to distribute your VMs across the underlying hardware to meet your redundancy requirements. This example is basic: an update domain likely contains more than one physical server.

VMs across these fault domains so that you always have VMs available if power or a network switch fails.

VMs that use managed disks (remember, *all* your VMs should use managed disks!) also respect logical fault-domain boundaries and distribution. The Azure platform logically assigns storage clusters to fault domains to ensure that as your VMs are distributed across hardware groups, the managed disks are also distributed across storage hardware. There would be no point in VM redundancy across server hardware if there were potential for all the managed disks to end up in one storage cluster! And yes, managed disks can be used with Availability Zones too.

7.3.2 Update domains

Whereas fault domains create a logical group of hardware to protect against hardware failures, update domains protect against routine maintenance. To provide this protection, a fault domain is further logically divided into update domains. Again, there's nothing for you to configure here. Update domains are a way for the Azure platform to understand how it must distribute VMs across your Availability Set.

Azure engineers perform (mostly automated) maintenance and apply updates across all the physical hardware in one update domain, and then perform the same maintenance across all hardware in the next update domain. This maintenance work is staggered across update domains to make sure that the VMs in an Availability Set aren't all running on hardware that undergoes maintenance at the same time. It's the same kind of process we looked at with Availability Zones; the distribution of your resources means that you can't have a scenario in which all the underlying hardware for your resources is being updated at the same time.

There's no relationship between domains across multiple Availability Sets. The physical resources that make up the fault and update domains in one Availability Set may not be the same for a second Availability Set. This awareness means that if you create multiple Availability Sets and distribute your VMs across them, fault domain 1, for example, doesn't always contain the same physical hardware.

7.3.3 Distributing VMs across an Availability Set

Let's go step by step and see how VMs are distributed across the logical fault and update domains that make up an Availability Set. This way, you have multiple VMs that can run your pizza store, and customers won't go hungry!

Try it now

To see Availability Sets in action, complete the following steps to deploy a Resource Manager template:

1. Open a web browser to a Resource Manager template from the GitHub samples repo at https://github.com/fouldsy/azure-mol-samples-2nd-ed/tree/master/07/availability-set, and then select the Deploy to Azure button. You'll use a template in this exercise so that you can deploy VMs quickly and explore how those VMs are distributed across the Availability Set.

   The Azure portal opens and prompts for a few parameters.

2. Choose to create a new resource group, and then provide a name such as `azuremolchapter7`. Select a region, and then provide your SSH key data (you can obtain in this Cloud Shell with cat ~/.ssh/id_rsa.pub).

   The template creates an Availability Set that contains three VMs. These VMs are distributed across the logical fault and update domains. Building on what you learned about Resource Manager in chapter 6, this template uses the `copyIndex()` function to create multiple VMs and NICs.

3. To acknowledge that you wish to create the resources detailed in the template, check the box, "I agree to the terms and conditions stated above," and then select Purchase.

It takes a few minutes to create all three VMs in the Availability Set. Let the deployment continue in the portal while you read the rest of this section.

When the template starts to deploy, an Availability Set is created, and the number of update and fault domains you requested is assigned. The following properties were defined in the sample template:

```
"properties": {
    "platformFaultDomainCount": "2",
    "platformUpdateDomainCount": "5",
    "managed": "true"
}
```

www.dbooks.org

These properties create an Availability Set with two fault domains and five update domains, as shown in figure 7.5, and indicate that the VMs are to use managed disks, so honor the disk distribution accordingly. The region you select for the Availability Set determines the maximum number of fault and update domains. Regions support either 2 or 3 fault domains and up to 20 update domains.

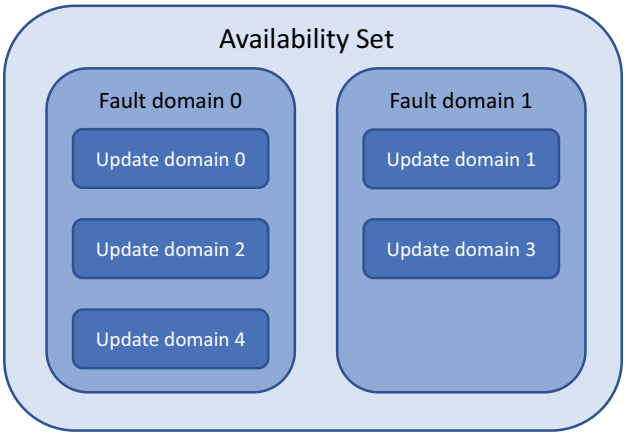


Figure 7.5 The Availability Set that your sample template deploys contains two fault domains and five update domains. The numbering system is zero-based. The update domains are created sequentially across the fault domains.

As you create more VMs in an Availability Set, you need to consider how many update domains to use. Five update domains, for example, mean that up to 20 percent of your VMs may be unavailable due to maintenance:

- Let's say that you have 10 VMs in your Availability Set. Two of those VMs may undergo maintenance at the same time. If you wanted to allow only one VM at a time to undergo maintenance, you'd need to create 10 update domains. The more update domains you create, the longer your application is potentially in a maintenance state.
- Let's continue the previous example of 10 VMs across 10 update domains. Now there's potential for disruption to your applications until all 10 of those update domains have completed their maintenance cycle. If you have only 5 update domains, that maintenance time frame is reduced. It's not necessarily bad to have a longer maintenance period; it's more what your tolerance is for potentially running at less than full capacity.

It's important to remember that these update domains and maintenance cycles are what the Azure platform performs itself. You also need to factor in your own update needs and maintenance windows.

When the first VM is created, the Azure platform looks to see where the first available deployment position would be. This is fault domain 0 and update domain 0, as shown in figure 7.6.

When the second VM is created, the Azure platform looks to see where the next available deployment position would be. This is now fault domain 1 and update domain 1, as shown in figure 7.7.

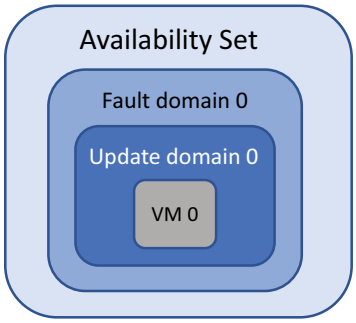


Figure 7.6 The first VM is created in fault domain 0 and update domain 0.

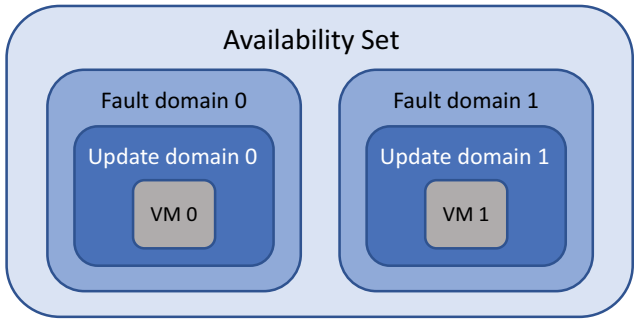


Figure 7.7 With a second VM created, the VMs are evenly distributed across fault and update domains. This is often considered the minimal amount of redundancy required to protect your applications.

Your template creates three VMs, so what do you think happens next? The Azure platform looks again to see where the next available deployment position would be. You created only two fault domains, so the VM is created back in fault domain 0. But the VM is created in a different update domain from the first VM. The third VM is created in update domain 2, as shown in figure 7.8.

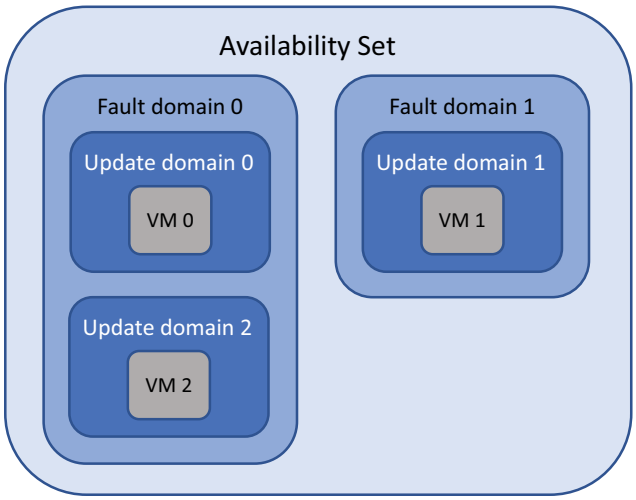


Figure 7.8 The third VM is created back in fault domain 0 but in update domain 2. Although VMs 0 and 2 potentially share hardware failure risk, they're in different update domains and so will not undergo regular maintenance at the same time.

www.dbooks.org

VMs 0 and 2 are in the same fault domain, so potentially, a hardware failure could affect both VMs. But routine maintenance affects only one of those VMs at a time because they're distributed across update domains. If you keep going and create more VMs, the Azure platform will continue to distribute them across different fault and update domains. When all five update domains are used, the sixth VM is created back in update domain 0, and the cycle continues.

7.3.4 View distribution of VMs across an Availability Set

Now that you understand the theory of how VMs are distributed across fault and update domains in an Availability Set, let's check what happened to your Resource Manager template deployment.

Try it now

To see how your VMs are distributed in an Availability Set, complete the following steps:

1. Browse to and select Resource Group from the navigation bar at left in the Azure portal.
2. Choose the resource group you created for your template deployment, such as `azuremolchapter7`.
3. Select your Availability Set from the list of resources, such as `azuremolavailabilityset`.

   The Overview window displays a list of VMs and their associated fault and update domains, as shown in figure 7.9.

NAME	STATUS	FAULT DOMAIN	UPDATE DOMAIN
vm0	Running	1	1
vm1	Running	0	2
vm2	Running	0	0

Figure 7.9 The Availability Set lists the VMs it contains and shows the fault domain and update domain for each VM. This table lets you visualize how the VMs are distributed across the logical domains.

If you're particularly observant, you may notice that the VMs don't line up perfectly with the expected order of fault and update domains. Is there a bug? Probably not. If you examine the example in figure 7.9 and compare it with what the previous concepts told you, you'd expect the VMs to be distributed as shown in table 7.1.

Table 7.1 Availability Set VMs sequentially created and distributed across domains

Name	Fault domain	Update domain
vm0	0	0
vm1	1	1
vm2	0	2

So what went wrong? Nothing. Think back to how Resource Manager creates resources from a template. The Azure platform doesn't wait for the first VM to be created before the second can be created. All three VMs are created at the same time. As such, there may be fractions of a second difference in which VM is associated with an Availability Set first. It doesn't matter what this order is, because you don't control what the underlying fault and update domains represent. That's up to the Azure platform. You just need to make sure that your VMs *are* distributed, not *where*.

No, I must have pretty numbers

If the serial creation behavior of VMs bugs you, and you *must* distribute the VMs in a neat order, you can instruct Resource Manager to create VMs in *serial*, rather than *parallel*. In this mode, the VMs are created one after another, so the deployment time is increased. To enable this serial behavior, use `"mode": "serial"` in your templates as part of the `copyIndex()` function. That should distribute the VMs in a nice, sequential way for you!

7.4 Lab: Deploying highly available VMs from a template

This lab combines and reinforces what you learned in chapter 6 about Azure Resource Manager and templates, with Availability Zones. Take some time to look over the example quick-start template in this exercise to see how you can use logic and functions to distribute multiple VMs across zones. Don't just deploy the template and move on; look at how the template builds on the features introduced in chapter 6!

What's a quota?

In Azure, default quotas on your subscription prevent you from accidentally deploying a bunch of resources and forgetting about them, which would cost you a lot of money. These quotas typically vary by resource type and subscription type and are enforced at the region level. You can see a full list of quotas at http://mng.bz/ddcx.

When you start to create multiple VMs in the next few chapters, you may run into quota issues. You can also encounter quota issues if you haven't deleted resources from previous chapters and exercises. The quotas are a good system that keeps you

www.dbooks.org

aware of your resource usage. The error messages may not be clear, but if you see error text along the lines of

```
Operation results in exceeding quota limits of Core.
Maximum allowed: 4, Current in use: 4, Additional requested: 2.
```

that's a good indication that you need to request an increase in your quotas. Nothing is complicated or unique to Azure. You can view your current quota for a given region as follows:

```
az vm list-usage --location eastus
```

If you have trouble with this lab, delete the first two resource groups you created in this chapter, such as `azuremolchapter7` and `azuremolchapter7az`. If you have a low default quota set, the four VMs across those resource groups may prevent you from successfully completing this exercise.

To request an increase in your quotas for a region, follow the steps outlined at http://mng.bz/Xq2f.

Let's review and deploy a sample template that includes multiple VMs across Availability Zones.

1. In a web browser, open the JSON file at https://github.com/Azure/azure-quickstart-templates/blob/master/201-multi-vm-lb-zones/azuredeploy.json, and search for the following text:

   ```
   Microsoft.Compute/virtualMachines
   ```

   The VMs section looks similar to what you used in chapter 6, but notice the property value for zones. This section combines a few functions available in templates to pick zone 1, 2, or 3 as the VM is created. This way, you don't need to manually track what VM runs in which zone, and how you then deploy additional VMs.
2. In your web browser, search for each of the following to see the sections on the public IP address and load balancer:

   ```
   Microsoft.Network/publicIPAddresses
   Microsoft.Network/loadBalancers
   ```

   Both resources use standard SKU, which provides zone redundancy by default. There's zero additional configuration to make this work! Let's see this in action.
3. In your web browser, open the quick-start template at http://mng.bz/O69a, and select the Deploy to Azure button.
4. Create or select a resource group, and then provide a username and password for the VMs.
5. Enter a unique DNS name, such as azuremol.

6 Choose to create Linux or Windows VMs. Windows VMs take a little longer to create.

7 Specify how many VMs to create, such as 3.

8 Check the box to agree to the terms and conditions for the template deployment, and select Purchase, as shown in figure 7.10.

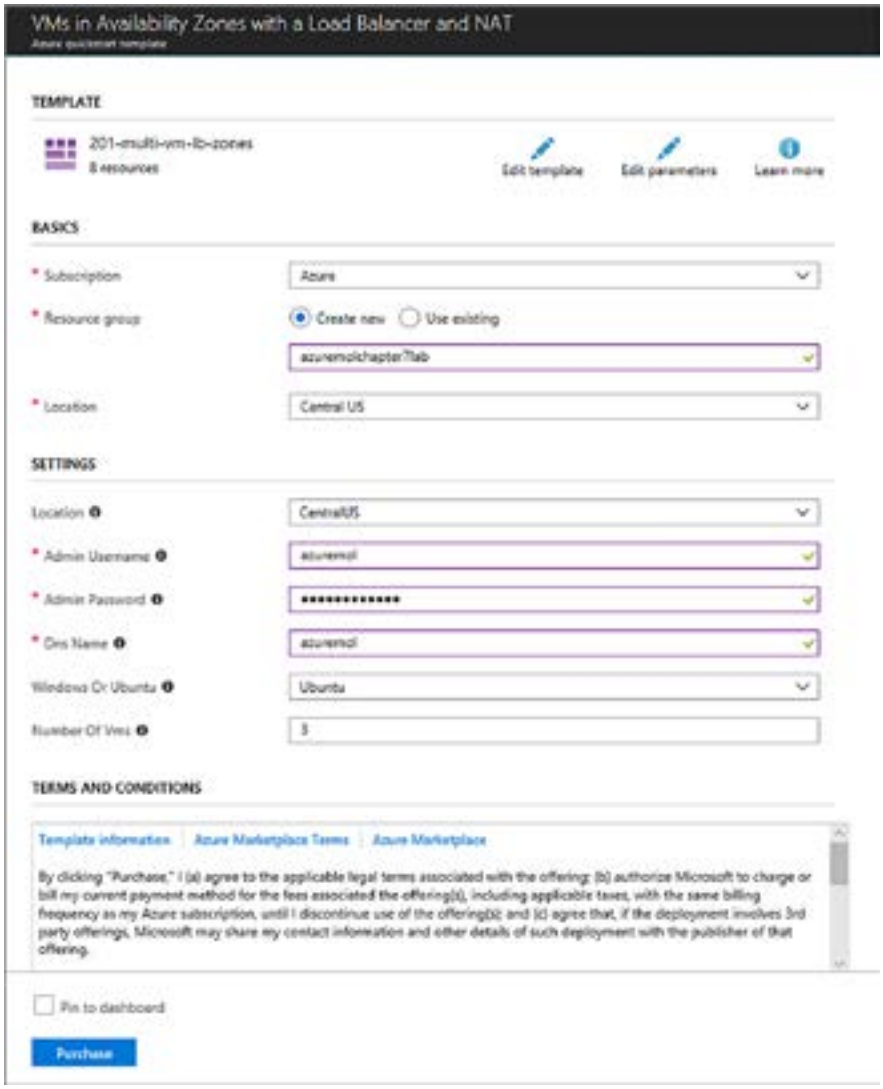


Figure 7.10 To deploy the Availability Zone template in the Azure portal, specify a resource group, username, and password, and then the OS type and the number of VMs you wish to create. The template uses loops, `copyIndex()`, `dependsOn`, variables, and parameters, as covered in chapter 6.

www.dbooks.org

When the VMs have been created, use the Azure portal or the `az vm show` command to see how the VMs were distributed across zones. If you're curious about what the rest of the template does with the network resources, chapter 8 dives deep into load balancers for you!

Cleanup on aisle 3

Remember that at the start of the book, I said to make sure you clean up after yourself to minimize the cost against your free Azure credits. I strongly advise you to delete the resource groups you created in this chapter. The next couple of chapters continue to create multiple VMs and web app instances, so make sure that you keep costs and quotas under control.

Each time you log in to the Azure portal, you should get a pop-up notification that lets you know the status of your Azure credits. If you see your available credit reduced by a large dollar amount from day to day, examine what resource groups you may have forgotten to delete!

8 Load-balancing applications

An important component of highly available applications is how to distribute traffic across all your VMs. In chapter 7, you learned the difference between Availability Zones and Availability Sets, as well as how you can create multiple VMs across Azure datacenters or regions to provide application redundancy. Even if you have all these highly available and distributed VMs, that doesn't help if only one VM receives all the customer traffic.

Load balancers are network resources that receive the incoming application traffic from your customers, examine the traffic to apply filters and load-balancing rules, and then distribute the requests across a pool of VMs that run your application. In Azure, there are a couple of ways to load-balance traffic, such as if you need to perform SSL off-loading on large applications that use encrypted network traffic. In this chapter, you'll learn about the various load-balancer components, and how to configure traffic rules and filters and distribute traffic to VMs. You'll build on the high-availability components from chapter 8 and get ready for chapter 9, which covers how to scale resources.

8.1 Azure load-balancer components

Load balancers in Azure can work at two levels: layer 4, where just the network traffic is examined and distributed (the transport layer, really); and layer 7, where there's an awareness of the application data within the network traffic to help determine the distribution of data. Both levels of load balancer work the same way, as shown in figure 8.1.

www.dbooks.org

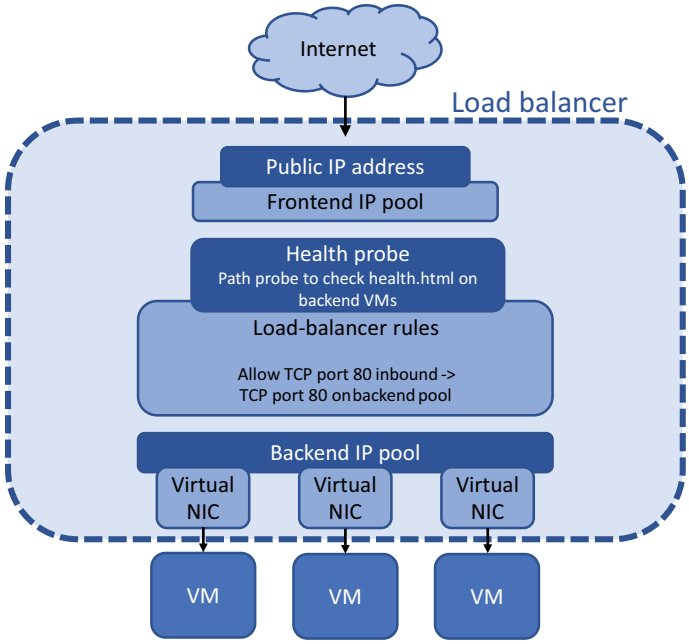


Figure 8.1 Traffic from the internet enters the load balancer through a public IP address that's attached to a frontend IP pool. The traffic is processed by load-balancer rules that determine how and where the traffic should be forwarded. Health probes attached to the rules ensure that traffic is distributed only to healthy nodes. Then a backend pool of virtual NICs connected to VMs receives the traffic distributed by the load-balancer rules.

A load balancer consists of a few main components:

- *Frontend IP pool*—Entry point to the load balancer. To allow access from the internet, a public IP address can be attached to the frontend IP pool. Private IP addresses can be attached for internal load balancers.
- *Health probes*—Monitor the status of attached VMs. To make sure that traffic is distributed only to healthy and responsive VMs, checks are performed on a regular basis to confirm that a VM correctly responds to traffic.
- *Load-balancer rules*—Distribute the traffic to VMs. Each incoming packet is compared with the rules, which define incoming protocols and ports, and then distributed across a set of associated VMs. If no rules match the incoming traffic, the traffic is dropped.
- *Network Address Translation (NAT) rules*—Can route traffic directly to specific VMs. If you want to provide remote access via SSH or RDP, for example, you can define NAT rules to forward traffic from an external port to a single VM.
- *Backend IP pool*—Where the VMs that run your application are attached. Load-balancer rules are associated with backend pools. You can create different backend pools for different parts of your applications.

Azure Application Gateway: advanced load balancing

Azure load balancers can work at the network layer or the application layer. This chapter focuses on the regular Azure load balancer, which works at the network layer (layer 4, or transport protocol). At this layer, the traffic is examined and distributed, but the load balancer has no context of what the traffic means or the applications you run.

Azure Application Gateway is a load balancer that works at the application layer (layer 7). Application Gateway gains insight into the application that runs on the VM and can manage the traffic flows in more advanced ways. One major benefit of Application Gateway is the ability to handle encrypted, HTTPS web traffic.

When you load-balance websites with SSL certificates, you can offload the process that verifies and decrypts the traffic from the web servers. On websites with a lot of SSL traffic, the process to verify and decrypt the traffic can consume a large portion of compute time on the VMs or web apps. Application Gateway can verify and decrypt the traffic, pass the pure web request to the web servers, and then reencrypt the traffic received from the web servers and return it to the customer.

Application Gateway offers some other more advanced load-balancer features, such as the ability to distribute traffic across any IP endpoint rather than just an Azure VM. As you build applications that use more than VMs, these advanced distribution rules may be of use to you. The same core concepts apply as with a regular load balancer, which is what we'll focus on in this chapter so that you understand how it all works in Azure.

8.1.1 Creating a frontend IP pool

In previous chapters, you created VMs that had a public IP address assigned directly to them. Then you used this public IP address to access the VM with a remote connection such as SSH or RDP, or used a web browser to access a website that ran on the VM. When you use a load balancer, you no longer connect straight to the VMs. Instead, to allow traffic to reach your load balancer and be distributed to your VMs, one or more IP addresses must be assigned to the external interface of a load balancer.

Load balancers can operate in either of two modes:

- *Internet load balancer*—Has one or more *public* IP addresses connected to the frontend IP pool. An internet load balancer receives traffic from the internet directly and distributes it to backend VMs. A common example is frontend web servers that customers access directly over the internet.
- *Internal load balancer*—Has one or more *private* IP addresses connected to the frontend IP pool. An internal load balancer works inside an Azure virtual network, such as for backend database VMs. You typically don't expose backend databases or application tiers to the outside world. Instead, a set of frontend web servers connects to an internal load balancer that distributes the traffic without any direct public access. Figure 8.2 shows how an internal load balancer can distribute traffic to backend VMs that are behind a public-facing load balancer and frontend web VMs.

www.dbooks.org

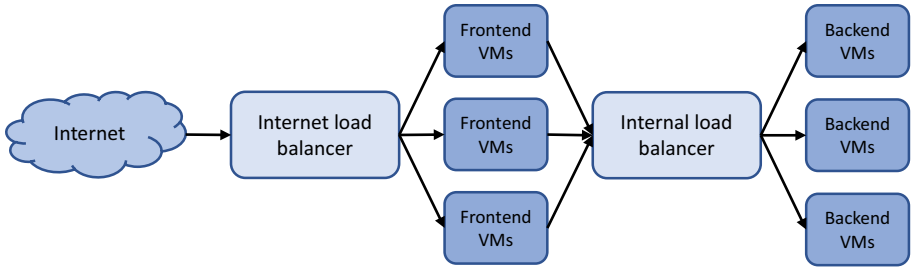


Figure 8.2 An internet load balancer may be used to distribute traffic to frontend VMs that run your website, which then connect to an internal load balancer to distribute traffic to a database tier of VMs. The internal load balancer isn't publicly accessible and can be accessed only by the frontend VMs within the Azure virtual network.

The mode for your load balancer doesn't change the behavior of the frontend IP pool. You assign one or more IP addresses that are used when access to the load balancer is requested. Both IPv4 and IPv6 addresses can be configured for the frontend IP pool, allowing you to configure end-to-end IPv6 communications between customers and your VMs as the traffic flows in and out of the load balancer.

Try it now

To understand how the load-balancer components work together, complete the following steps to create a load balancer and frontend IP pool:

1. Open the Azure portal, and select the Cloud Shell icon at the top of the dashboard.
2. Create a resource group with `az group create`.

   Specify a resource group name, such as `azuremolchapter8`, and a location:

   ```
   az group create --name azuremolchapter8 --location westeurope
   ```

   As you continue to build on chapter 7 and want to use availability zones, take care with the region you select to make sure that availability zone support is available.
3. Create a public IP address with `az network public-ip create`.

   In chapter 7, you learned that availability zones provide redundancy to network resources, so create a standard, zone-redundant public IP address and specify a name, such as `publicip`:

   ```
   az network public-ip create \
     --resource-group azuremolchapter8 \
     --name publicip \
     --sku standard
   ```

To create an IPv6 public IP address, you can add `--version IPv6` to the preceding command. For these exercises, you can use IPv4 addresses.

4 Create the load balancer, and assign the public IP address to the frontend IP pool. To add the public IP address, specify the `--public-ip-address` parameter. If you wanted to create an internal load balancer, you'd use the `--private-ip-address` parameter instead.

As with the public IP address, create a standard, zone-redundant load balancer that works across availability zones:

```
az network lb create \
  --resource-group azuremolchapter8 \
  --name loadbalancer \
  --public-ip-address publicip \
  --frontend-ip-name frontendpool \
  --backend-pool-name backendpool \
  --sku standard
```

We'll dive into what the backend pool is in a few pages.

8.1.2 Creating and configuring health probes

If one of the VMs that run your application has a problem, do you think the load balancer should continue to distribute traffic to that VM? A customer who tries to access your pizza store may get directed to that VM and be unable to order any food! A load balancer monitors the status of the VMs and can remove VMs that have issues. The load balancer continues to monitor the health and adds the VM back into the pool for traffic distribution when the VM is shown to respond correctly again.

A health probe can work in a couple of modes:

- *Port-based*—The load balancer checks for a VM response on a specific port and protocol, such as TCP port 80. As long as the VM responds to the health probe on TCP port 80, the VM remains in the load-balancer traffic distribution. Otherwise, the VM is removed from the load-balancer traffic distribution, as shown in figure 8.3. This mode doesn't guarantee that the VM serves the traffic as expected—only that the network connectivity and destination service returns a response.
- *HTTP path–based*—A custom page, such as health.html, is written and placed on each VM. This custom health check can be used to verify access to an image store or database connection. In this mode, the VM remains in the load-balancer traffic distribution only when the health-check page returns an HTTP code 200 response, as shown in figure 8.4. With a port-based health probe, the actual web server may run but have no database connection. With a custom health-check page, the load balancer can confirm that the VM is able to serve real traffic to the customer.

www.dbooks.org

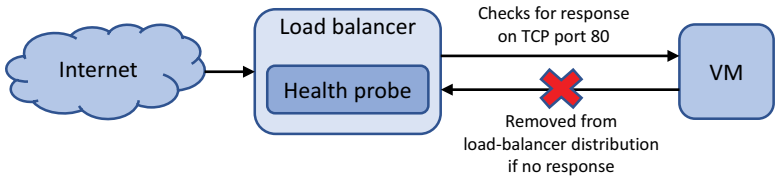


Figure 8.3 A port-based load-balancer health probe checks for a VM response on a defined port and protocol. If the VM doesn't respond within the given threshold, the VM is removed from the load-balancer traffic distribution. When the VM starts to respond correctly again, the health probe detects the change and adds the VM back into the load-balancer traffic distribution.

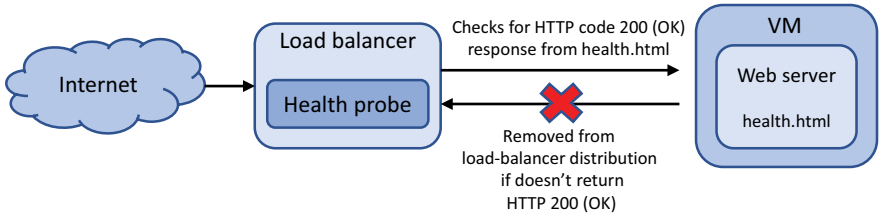


Figure 8.4 A VM that runs a web server and has a custom health.html page remains in the load-balancer traffic distribution, provided that the health probe receives an HTTP code 200 (OK) response. If the web server process encounters a problem and can't return requested pages, those pages are removed from the load-balancer traffic distribution. This process provides a more thorough check of the web server state than port-based health probes.

Additional work is required to create the custom health-check page, but the improved customer experience is worthwhile. The health-check page doesn't have to be complicated. It could be a basic HTML page that's used to confirm that the web server itself can serve pages. Without the health-check page, if the web server process has a problem, the VM would still be available on TCP port 80, so the port-based health probe would believe the VM to be healthy. An HTTP path-based health probe requires the web server to correctly return an HTTP response. If the web server process hangs or has failed, an HTTP response isn't sent, so the VM is removed from the load-balancer traffic distribution.

How often the health probe checks the VM, and what the response is, can also be configured through two parameters:

- *Interval*—Defines how frequently the health probe checks the status of the VM. By default, the health probe checks the status every 15 seconds.
- *Threshold*—Defines how many consecutive response failures the health probe receives before the VM is removed from the load-balancer traffic distribution. By default, the health probe tolerates two consecutive failures before the VM is removed from the load-balancer traffic distribution.

Try it now

To create a health probe for your load balancer as in figure 8.4, complete the following steps:

1. Open the Azure portal, and select the Cloud Shell icon at the top of the dashboard.
2. Specify a name for the health probe, such as `healthprobe`. To set up the health probe for a web server, specify HTTP port 80, and then define a custom health-check page at health.html. In section 8.2, you'll create this health-check page on your VMs. To show how the interval and threshold for the health-probe response can be configured, define an interval of 10 seconds and a threshold of three consecutive failures:

```
az network lb probe create \
  --resource-group azuremolchapter8 \
  --lb-name loadbalancer \
  --name healthprobe \
  --protocol http \
  --port 80 \
  --path health.html \
  --interval 10 \
  --threshold 3
```

After the health probe is created, how do you make it check the status of your VMs? Health probes are associated with load-balancer rules. The same health probe can be used with multiple load-balancer rules. Remember chapter 5, in which you created network security groups (NSGs) and rules? Those NSGs can be associated with multiple VMs or virtual network subnets. A similar one-to-many relationship applies to health probes.

Let's see how to put your health probe to work and create load-balancer rules.

8.1.3 Defining traffic distribution with load-balancer rules

When traffic is directed through the load balancer to the backend VMs, you can define what conditions cause the user to be directed to the same VM. You may want the user to retain a connection to the same VM for the duration of a single session or allow them to return and maintain their VM affinity based on the source IP address. Figure 8.5 shows an example of the default session affinity mode.

In session affinity mode, the flow of traffic is handled by a 5-tuple hash that uses the source IP address, source port, destination IP address, destination port, and protocol type. Basically, for each request a user makes to your web server on TCP port 80, they're directed to the same backend VM for the duration of that session.

What happens if the customer closes their browser session? The next time they connect, a new session is started. Because the load balancer distributes traffic across all healthy VMs in the backend IP pool, it's possible that the user will connect to the same VM again, but the more VMs you have in the backend IP pool, the greater the chance that the user will connect to a different VM.

www.dbooks.org

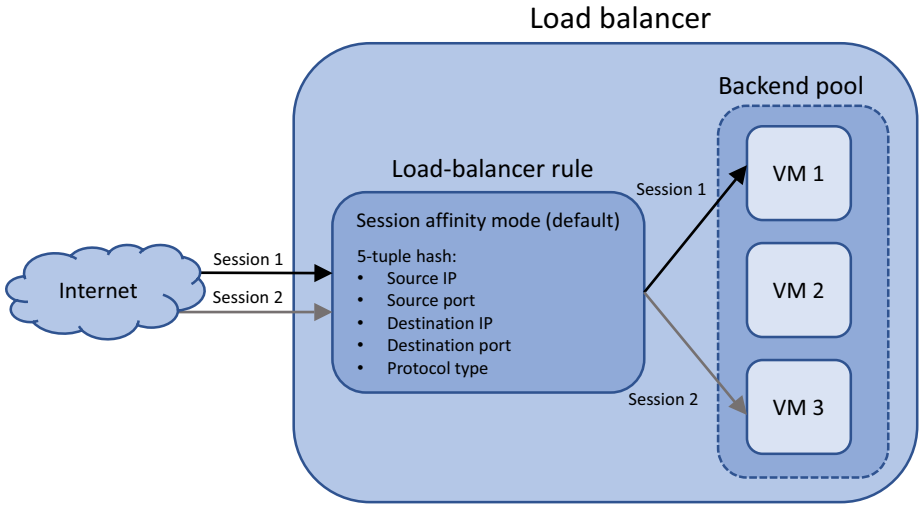


Figure 8.5 In session affinity mode, the user connects to the same backend VM only for the duration of their session.

As the application owner and developer, you may want the user to connect to the same VM as before when they start another session. If your application handles file transfers or uses UDP rather than TCP, for example, you likely want the same VM to continue to process a user's requests. In these scenarios, you can configure the load-balancer rules for source IP affinity. Figure 8.6 shows an example of source IP affinity mode.

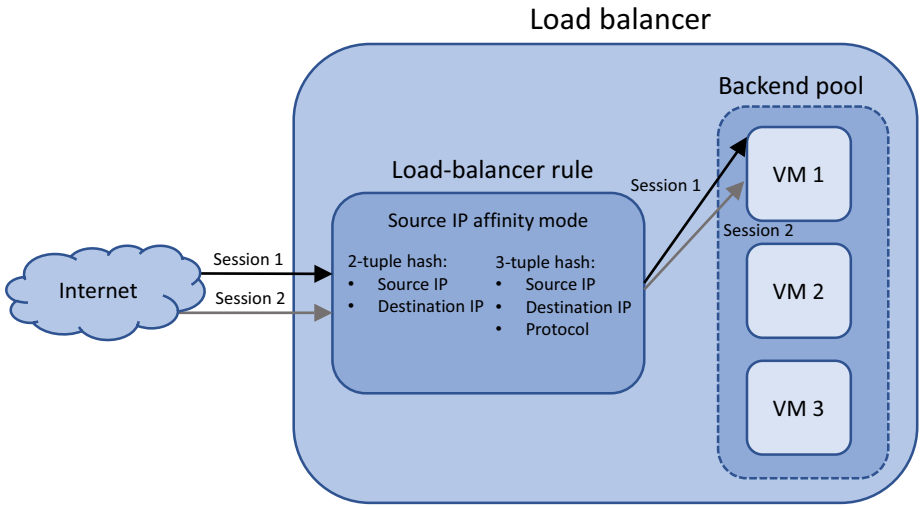


Figure 8.6 When you configure the load-balancer rules to use source IP affinity mode, the user can close and then start a new session but continue to connect to the same backend VM. Source IP affinity mode can use a 2-tuple hash that uses the source and destination IP address or a 3-tuple hash that also uses the protocol.

Try it now

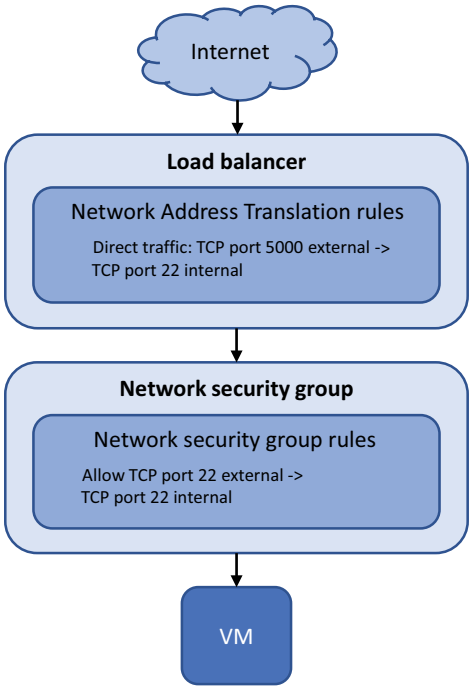
To create a load-balancer rule that uses a health probe, complete the following steps:

1. Open the Azure portal, and select the Cloud Shell icon at the top of the dashboard.
2. To create a load-balancer rule, specify a name for the rule, such as `httprule`.
3. Provide the external port on which traffic is received and the internal port to distribute traffic to. In this basic example, traffic is received on port 80 and then distributed on port 80:

```
az network lb rule create \
  --resource-group azuremolchapter8 \
  --lb-name loadbalancer \
  --name httprule \
  --protocol tcp \
  --frontend-port 80 \
  --backend-port 80 \
  --frontend-ip-name frontendpool \
  --backend-pool-name backendpool \
  --probe-name healthprobe
```

If you run multiple websites on a VM that responds on different ports, a given rule could direct traffic to a specific website on the VM.

8.1.4 Routing direct traffic with Network Address Translation rules



The load-balancer rules distribute traffic across the backend pools of VMs, so there's no guarantee that you can connect to a given VM for maintenance or management purposes. How can you connect to a specific VM that's behind a load balancer? One final part of the load-balancer configuration to look at is Network Address Translation (NAT) rules, which let you control the flow of specific traffic to direct it to a single VM. Figure 8.7 shows how NAT rules forward specific traffic to individual VMs.

Figure 8.7 Traffic in the load balancer is processed by NAT rules. If a protocol and port match a rule, the traffic is forwarded to the defined backend VM. No health probes are attached, so the load balancer doesn't check whether the VM is able to respond before it forwards the traffic. The traffic leaves the load balancer and then is processed by NSG rules. If the traffic is permitted, it's passed to the VM.

www.dbooks.org

NAT rules work alongside NSG rules. The VM can receive the traffic only if there's an NSG rule that allows the same traffic as the load-balancer NAT rule.

Why might you create NAT rules? What if you want to use SSH or RDP to connect to a specific VM (and aren't using Azure Bastion, which I mentioned in chapter 2) or use management tools to connect to a backend database server? If the load balancer distributes traffic across the backend VMs, you'd have try to connect again and again and again, and you still might not connect to the desired VM.

Keeping things secure

We'll dive into some security topics in part 3 of the book, but security should be an ever-present consideration as you build and run applications in Azure. Security shouldn't be something you add later. With the rise of cloud computing and disposable VMs and web apps, it's easy to overlook some basic security best practices. Especially if you work in Azure as part of a wider enterprise subscription, make sure that any resources you create don't accidentally provide a way for attackers to gain access to your infrastructure.

What kind of things are bad? Well, some of the things you've done already in this book! Remote management ports for SSH and RDP shouldn't be opened to the public internet, as you've done; at the very least, you should restrict access from a specific IP address range.

The best practice would be to use a managed service such as Azure Bastion, or manually create one secured VM that has remote management available. As needed, you connect the Azure Bastion host or your one secured VM, and then connect over the internal Azure virtual network to additional VMs. You used this basic jump-box VM approach in chapter 5. This approach minimizes the attack footprint and reduces the need for NSG rules and load-balancer NAT rules. Chapter 16 discusses Azure Security Center and shows how you can dynamically request and open remote-management ports for a specific time period, which is the best of both worlds.

Even if you work in a private Azure subscription that has no connectivity to other Azure subscriptions at school or work, try to minimize how much remote connectivity you provide.

Try it now

To create a load-balancer NAT rule, complete the following steps:

1. Open the Azure portal, and select the Cloud Shell icon at the top of the dashboard.
2. To create a load-balancer NAT rule, define a name, such as `natrulessh`, and the frontend IP pool to use. The NAT rule examines traffic on a given protocol and port, such as TCP port 50001. When there's a rule match, the traffic is forwarded to backend port 22:

```
az network lb inbound-nat-rule create \
  --resource-group azuremolchapter8 \
  --lb-name loadbalancer \
  --name natrulessh \
  --protocol tcp \
  --frontend-port 50001 \
  --backend-port 22 \
  --frontend-ip-name frontendpool
```

At this point, you've created a basic load balancer. Examine how the load-balancer components have come together:

```
az network lb show \
  --resource-group azuremolchapter8 \
  --name loadbalancer
```

A public IP address has been assigned to the frontend IP pool, and you created a health probe to check the status on a custom health page for a web server. A load-balancer rule was created to distribute web traffic from your customers to a backend pool; the rule uses the health probe. You also have a load-balancer NAT rule that permits SSH traffic, but there are no VMs to receive that traffic yet. Your pizza-store customers are hungry, so let's create some VMs that can run your web application and to which the load balancer can distribute traffic!

8.1.5 Assigning groups of VMs to backend pools

The final section of the load balancer defines backend pools that include one or more VMs. These backend pools contain VMs that run the same application components, which allows the load balancer to distribute traffic to a given backend pool and trust that any VM in that pool can respond correctly to the customer request. Figure 8.8 details how the backend pools logically group VMs that run the same applications.

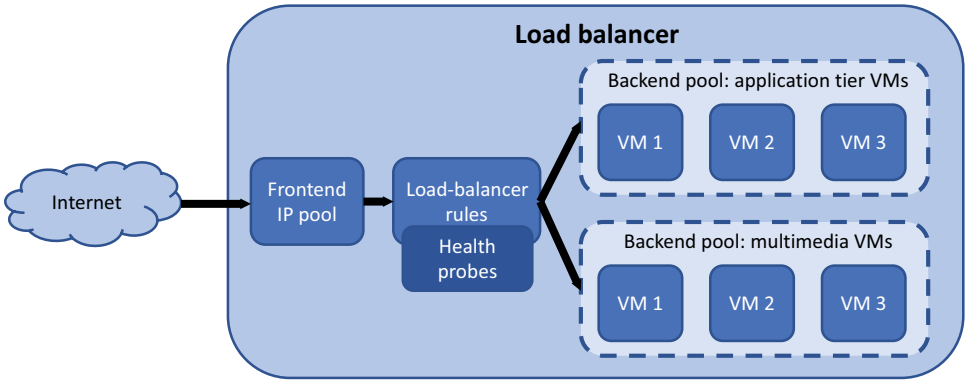


Figure 8.8 One or more backend pools can be created in a load balancer. Each backend pool contains one or more VMs that run the same application component. In this example, one backend pool contains VMs that run the web application tier, and another backend pool contains the VMs that serve multimedia, such as images and video.

www.dbooks.org

You create and use a load balancer with VMs, but everything works at the virtual network level. The frontend IP pool uses public or private IP addresses. The health probe looks at responses on a given port or protocol. Even when an HTTP probe is used, the load balancer looks for a positive network response. Load-balancer rules focus on how to distribute traffic from an external port in the frontend pool to a port on the backend pool.

When you assign VMs to the backend pool that receive traffic distributed by the load balancer, it's the virtual NIC that connects to the load balancer. The VM happens to attach to the virtual NIC. Think back to chapter 5, and this separation of VMs and virtual NIC makes sense in terms of how resources are managed. NSG rules control what traffic is permitted to flow to the VM, but they're applied to a virtual network subnet or virtual NIC, not the VM.

What does this mean for how you configure backend IP pools? You must create the rest of your virtual network resources before you can connect a VM to the load balancer. The steps to create the network resources should be a recap of what you learned a few chapters ago, so let's see how much you remember!

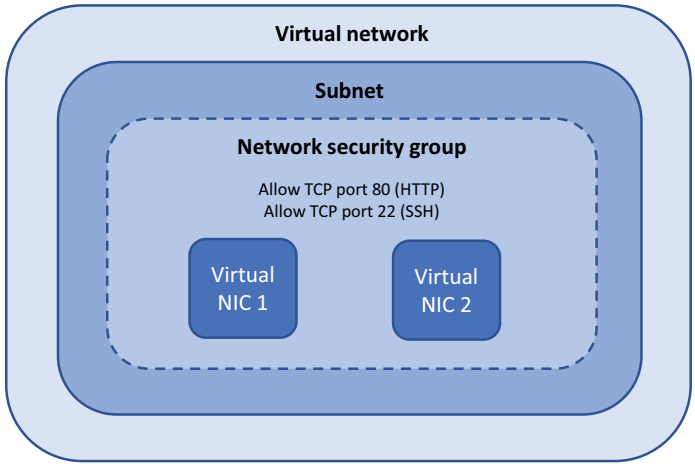


Figure 8.9 To prepare the virtual network, in this exercise you'll create a network, a subnet, and virtual NICs that are protected by an NSG. Rules attached to the NSG allow HTTP and SSH traffic.

Try it now

To create the additional network resources, as shown in figure 8.9, complete the following steps:

1. Open the Azure portal, and select the Cloud Shell icon at the top of the dashboard.
2. Create a virtual network and subnet:

```
az network vnet create \
  --resource-group azuremolchapter8 \
```

```
  --name vnetmol \
  --address-prefixes 10.0.0.0/16 \
  --subnet-name subnetmol \
  --subnet-prefix 10.0.1.0/24
```

In practice, there's a good chance that these network resources already exist. Also, these names and IP address ranges are the same ones you used in chapter 5. You should clean up Azure resources at the end of each chapter, so such reuse of IP ranges shouldn't be a problem. Just be aware that you typically won't create a virtual network and subnet every time you create a load balancer. Rather, you can use the existing virtual network resources that are already in place.

3 Create an NSG:

```
az network nsg create \
  --resource-group azuremolchapter8 \
  --name webnsg
```

4 Create an NSG rule that allows traffic from TCP port 80 to reach your VMs. This rule is necessary for the web server VMs to receive and respond to customer traffic:

```
az network nsg rule create \
  --resource-group azuremolchapter8 \
  --nsg-name webnsg \
  --name allowhttp \
  --priority 100 \
  --protocol tcp \
  --destination-port-range 80 \
  --access allow
```

5 Add another rule to allow SSH traffic for remote management. This NSG rule works with the load-balancer NAT rule created in section 8.1.4 for one of your VMs:

```
az network nsg rule create \
  --resource-group azuremolchapter8 \
  --nsg-name webnsg \
  --name allowssh \
  --priority 101 \
  --protocol tcp \
  --destination-port-range 22 \
  --access allow
```

6 Associate the NSG with the subnet created in step 2. The NSG rules are applied to all VMs that connect to this subnet:

```
az network vnet subnet update \
  --resource-group azuremolchapter8 \
  --vnet-name vnetmol \
  --name subnetmol \
  --network-security-group webnsg
```

7 The load balancer works with virtual NICs, so create two virtual NICs, and connect them to the virtual network subnet. Also specify the load-balancer name

www.dbooks.org

and backend address pool that the virtual NICs connect to. The load-balancer NAT rule is attached only to the first virtual NIC that's created:

```
az network nic create \
  --resource-group azuremolchapter8 \
  --name webnic1 \
  --vnet-name vnetmol \
  --subnet subnetmol \
  --lb-name loadbalancer \
  --lb-address-pools backendpool \
  --lb-inbound-nat-rules natrulessh
```

8 Create the second NIC in the same way, minus the load-balancer NAT rule:

```
az network nic create \
  --resource-group azuremolchapter8 \
  --name webnic2 \
  --vnet-name vnetmol \
  --subnet subnetmol \
  --lb-name loadbalancer \
  --lb-address-pools backendpool
```

8.2 Creating and configuring VMs with the load balancer

Let's pause to explore what you've created. Figure 8.10 shows the big picture of what your network resources and load balancer look like. Notice how integrated these resources are. The load balancer can't exist by itself. Virtual NICs must be connected

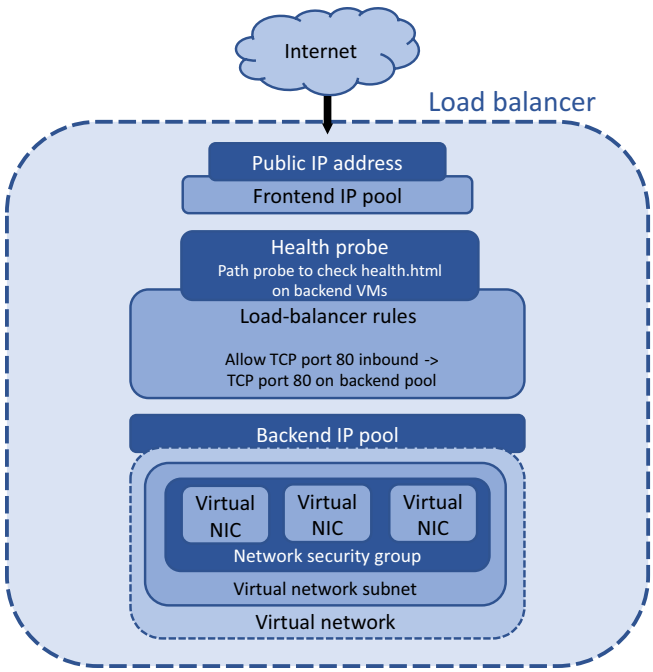


Figure 8.10 No VMs have been created here; the load-balancer configuration deals with virtual network resources. There's a tight relationship between the load balancer and virtual network resources.

to the load balancer for any traffic to be distributed. Those virtual NICs require a virtual network and subnet, and ideally are protected by an NSG. The VMs that run your application have almost nothing to do with the steps to create and configure the load balancer!

You've created a lot of network resources and configured multiple parts of the load balancer. The public IP address and load balancer were created in an availability zone as zone-redundant resources, so let's create two VMs across different zones to reinforce how availability zones enhance the high availability of your applications.

If you use Availability Sets rather than Availability Zones, this is where you create an Availability Set and then add your VMs to it; then the Azure platform distributes the VMs across the fault and update domains. You want to maximize the use of Azure high availability for your pizza store, so use Availability Zones.

Try it now

To create the VMs and attach them to the load balancer, complete the following steps:

1 Create the first VM, and assign it to an availability zone with `--zone 1`:

```
az vm create \
  --resource-group azuremolchapter8 \
  --name webvm1 \
  --image ubuntults \
  --size Standard_B1ms \
  --admin-username azuremol \
  --generate-ssh-keys \
  --zone 1 \
  --nics webnic1
```

2 Create the second VM; assign it to availability zone 2; and attach the second virtual NIC you created earlier, using `--nics webnic2`:

```
az vm create \
  --resource-group azuremolchapter8 \
  --name webvm2 \
  --image ubuntults \
  --size Standard_B1ms \
  --admin-username azuremol \
  --generate-ssh-keys \
  --zone 2 \
  --nics webnic2
```

To see the load balancer in action, you need to install a basic web server, as you did in chapter 2. You can also try out the load-balancer NAT rule. Can you start to see how all these components in Azure are related and build on one another?

www.dbooks.org

Try it now

In chapter 5, we discussed the SSH agent. The SSH agent allows you to pass an SSH key from one VM to the next. Only VM1 has a load-balancer NAT rule, so you need to use the agent to connect to VM2. To install a web server on your VMs, complete the following steps:

1. Start the SSH agent, and add your SSH key so that you can connect to both VMs:

```
eval $(ssh-agent) && ssh-add
```

2. Obtain the public IP address attached to the load-balancer frontend IP pool. This is the only way for traffic to route through the VMs:

```
az network public-ip show \
  --resource-group azuremolchapter8 \
  --name publicip \
  --query ipAddress \
  --output tsv
```

3. Now you're ready to SSH to VM 1. Specify the public IP address of the load balancer (replace `<your-ip-address>` in the following command) and the port that was used with the load-balancer NAT rule, such as 50001. The `-A` parameter uses the SSH agent to pass through your SSH keys:

```
ssh -A azuremol@<your-ip-address> -p 50001
```

   In chapter 2, you used `apt-get` to install the entire LAMP stack, including the Apache web server. Let's see something a little different from the Apache web server with the standalone but powerful NGINX web server. On a Windows VM, this is typically where you'd install IIS. Run the following command to install the NGINX web server:

```
sudo apt update && sudo apt install -y nginx
```

4. In the GitHub samples repo that you've used in previous chapters, there's a basic HTML web page and a health-check page for the load-balancer health probe. Clone these samples to the VM:

```
git clone https://github.com/foulds/azure-mol-samples-2nd-ed.git
```

5. Copy the sample HTML page and health check to the web server directory:

```
sudo cp azure-mol-samples-2nd-ed/08/webvm1/* /var/www/html/
```

6. Now you need to connect to the second VM and install the NGINX web server and sample code. Remember the SSH agent? You should be able to SSH from VM 1 to VM 2 on the internal, private IP address:

```
ssh 10.0.1.5
```

7 Install the NGINX web server:

```
sudo apt update && sudo apt install -y nginx
```

8 Clone the GitHub samples to the VM:

```
git clone https://github.com/foouldsy/azure-mol-samples-2nd-ed.git
```

9 Copy the sample HTML page and health check to the web server directory:

```
sudo cp azure-mol-samples-2nd-ed/08/webvm2/* /var/www/html/
```

Open a web browser, and connect to the public IP address of your load balancer. The basic web page loads and shows that your pizza store now has redundant VMs in Availability Zones that run behind a load balancer, as shown in figure 8.11! You may need to force-refresh your web browser to see that both VM 1 and VM2 respond as the load balancer distributes traffic between them.

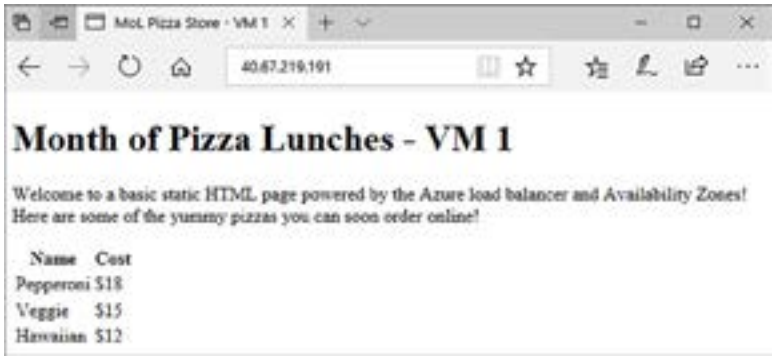


Figure 8.11 When you open the public IP address of the load balancer in a web browser, traffic is distributed to one of the VMs that run your basic website. The load-balancer health probe uses the health.html page to confirm that the web server responds with an HTTP code 200 (OK). If so, the VM is available as part of the load-balancer traffic distribution.

8.3 Lab: Viewing templates of existing deployments

This chapter ties together what you learned in multiple previous chapters. You created network resources, as in chapter 5. You made the load balancer and VMs highly available with Availability Zones, as in chapter 7. And you installed a web server and deployed sample files, as in chapter 2. Your pizza store has come a long way from the basic web page on a single VM at the start of the book!

To tie in one more theme from a previous chapter, in this lab I want you to explore all the resources that make up the load balancer. To do this, you look at the Resource Manager template, as you learned about in chapter 6. The goal of this lab is to see how a single template can create and configure what's taken many pages and multiple CLI

www.dbooks.org

commands. And trust me, it would take even more PowerShell commands! Follow these steps:

1. Open the Azure portal.
2. Browse to and select Resource Group from the navigation bar at left in the portal.
3. Choose your resource group, such as azuremolchapter8.
4. Choose Export Template from the bar on the left side, as shown in figure 8.12.
5. To see the relevant part of the template, select each of the resources shown in the list. Take a few minutes to explore this template and see how all the resources and components that you configured in the Azure CLI are present.

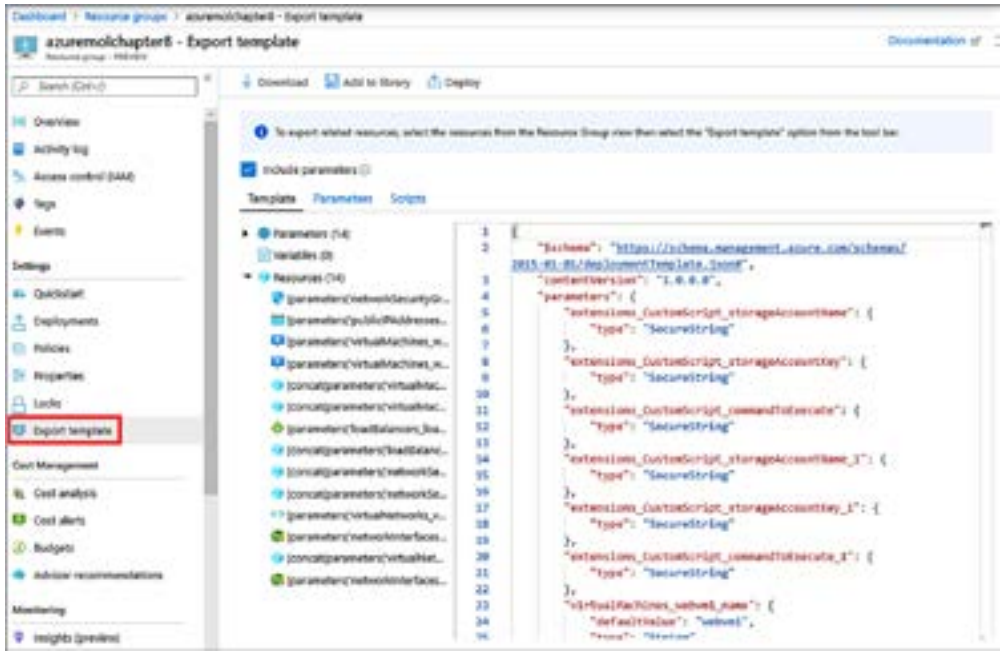


Figure 8.12 In the Azure portal, select your load-balancer resource group, and view the Resource Manager template.

A template makes it a lot easier to deploy a highly available, redundant, load-balanced application environment. You can change the load balancer's name, rules, and distribution mode, and let the template deploy and configure the entire application environment for you.

Don't forget to delete this resource group to make the most of your free Azure credits!

9 Applications that scale

In the previous two chapters, we examined how to build highly available applications and use load balancers to distribute traffic to multiple VMs that run your app. But how do you efficiently run and manage multiple VMs, and run the right number of VM instances when your customers need them most? When customer demand increases, you want to automatically increase the scale of your application to cope with that demand. And when demand decreases, such as in the middle of the night, when most people without young children are asleep, you want the application to decrease in scale and save you some money.

In Azure, you can automatically scale IaaS resources in and out with virtual machine scale sets. These scale sets run identical VMs, typically distributed behind a load balancer or application gateway. You define autoscale rules that increase or decrease the number of VM instances as customer demand changes. The load balancer or app gateway automatically distributes traffic to the new VM instances, which lets you focus on how to build and run your apps better. Scale sets give you control of IaaS resources with some of the elastic benefits of PaaS. Web apps, which we didn't cover a lot in the past couple of chapters, make a solid reappearance in this chapter, providing their own ability to scale with application demand.

In this chapter, we'll examine how to design and create applications that can scale automatically. We'll look at why this ability to scale with demand helps you run efficient applications, and we'll explore different ways to scale based on different metrics.

9.1 Why build scalable, reliable applications?

What does it mean to build applications that scale? It lets you grow and keep up with customer demand as the workload increases, even when you're at the movies on a weekend. It means you don't get stuck with a bill for a bunch of extra resources

www.dbooks.org

you don't use or, maybe worse, have your application go down due to a lack of available resources. The sweet spot for applications and the resources they need is rarely static. Usually, application demands ebb and flow throughout the day and night, or between weekdays and weekends.

There are two main ways you can scale resources, as shown in figure 9.1: vertically and horizontally. Both virtual machine scale sets and web apps can scale vertically or horizontally.

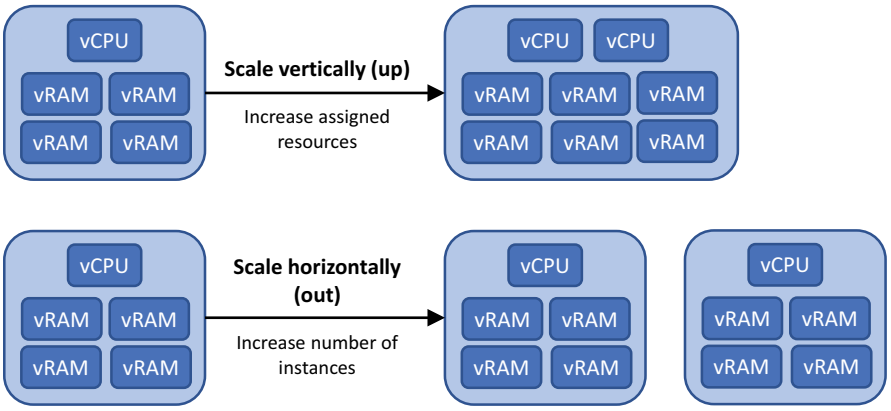


Figure 9.1 You can scale your applications up and down, or in and out. The method you use depends on how your application is built to handle scale. Vertical scale adjusts the resources assigned to a VM or web app, such as the number of CPU cores or amount of memory. This method to scale an application works well if the application runs only one instance. Horizontal scale changes the number of instances that run your application and helps increase availability and resiliency.

Scalable applications have a strong relationship with highly available applications. In chapters 7 and 8, we spent a lot of time with Availability Zones and Availability Sets, and how to configure load balancers. Both chapters centered around the need to run multiple VMs. When your application can scale automatically, the availability of that application is also increased as those VMs are distributed across Availability Sets or Availability Zones. All this is a Good Thing. The power of Azure is that you don't need to worry about how to add more application instances, spread those across data center hardware or even data centers, and then update network resources to distribute traffic to the new application instances.

9.1.1 Scaling VMs vertically

The first way to scale resources is often the time-honored way that you may have used in the past. If your application starts to perform slowly as more customers use it, what would you normally do? Increase the amount of CvPU or memory, right? You scale *up* the resource in response to demand.

One of the most common uses of vertical scale is for database servers. Databases are notoriously hungry when it comes to compute resources—even hungrier than your pizza-store customers! Database servers often consume all the resources provided to a VM, even if they don't use them immediately. This can make it hard to monitor the actual demands on the system and know when you need to scale vertically and provide more resources. Figure 9.2 shows the typical vertical scale response to a database server that needs more resources.

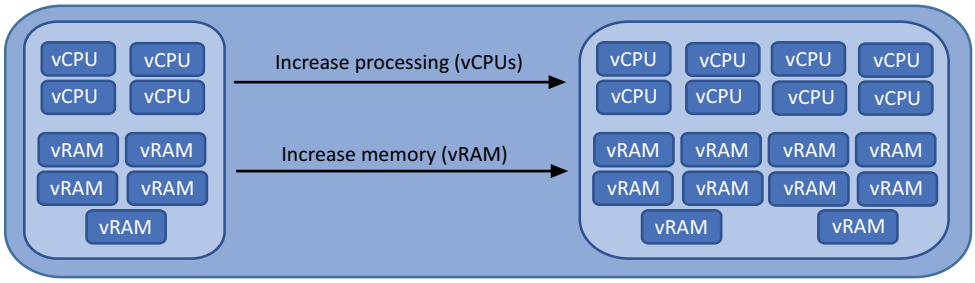


Figure 9.2 As a database grows, it needs more resources to store and process the data in memory. To scale vertically in this scenario, you add more CPU and memory.

You may need to scale beyond the demand for CPU or memory. What if you run a website that serves a lot of images or video? There may not be a lot of processing requirements, but the bandwidth demands may be high. To increase the available bandwidth, you can increase the number of NICs on your VM. And if you need to store more images and video, you add more storage. You can add or remove resources such as virtual NICs and storage as the VM continues to run.

RESIZING VIRTUAL MACHINES

In Azure, you can increase the VM size (scale up) if you need more compute resources for your application. Back in chapter 2, you created a basic VM. Its size was probably something like Standard_D2s_v3. That name doesn't tell you a lot about the compute resources assigned to a VM to determine whether you may need to increase the CPU or memory. If you want to scale vertically, you need to know what your options are.

Try it now

Follow along to see the available VM sizes and compute resources:

1. Open the Azure portal in a web browser, and then open Cloud Shell.
2. Enter the following Azure CLI command to list available VM sizes and the compute resources they provide:

```
az vm list-sizes --location eastus --output table
```

www.dbooks.org

The output from `az vm list-sizes` varies from region to region and changes over time as Azure adjusts its VM families. Here's a condensed example of the output, showing the `MemoryInMb` and `NumberOfCores` each VM size provides:

```
MaxDataDiskCount    MemoryInMb    Name                         NumberOfCores
------------------  ------------  ---------------------------  --------------------
                4          8192  Standard_D2s_v3                            2
                8         16384  Standard_D4s_v3                            4
               16         32768  Standard_D8s_v3                            8
               32         65536  Standard_D16s_v3                          16
                8          4096  Standard_F2s_v2                            2
               16          8192  Standard_F4s_v2                            4
               32         16384  Standard_F8s_v2                            8
                2          2048  Standard_B1ms                              1
                2          1024  Standard_B1s                               1
                4          8192  Standard_B2ms                              2
                4          4096  Standard_B2s                               2
```

So your Standard_D2s_v3 VM provides two CPU cores and 8 GB of memory—more than enough for a basic VM that runs a web server. Let's assume that your online pizza store starts to get some orders, and you want to scale vertically. You can use `az vm resize` to pick another size. You specify the VM size that has the number of CPU cores and memory your application needs.

The additional CPU and memory don't magically appear on your VM. This behavior may be a little different from what you experience with Hyper-V or VMware in an on-premises world. Within reason, you can add or remove core compute resources in an on-premises environment as the VM continues to run. In Azure, a reboot is usually required when you resize a VM to register the new compute resources and trigger the appropriate billing rules. When you want to scale vertically, plan for some downtime as the VM reboots.

Scaling down

What if you have a VM with more resources than it needs? This scenario is often more common than a VM that has fewer resources than needed. Application owners may choose a larger VM size than is required to make sure that their application runs smoothly. All those wasted resources cost money, and it's easy for the costs to go unnoticed until the bill arrives at the end of the month.

The ability to scale resources works in both directions. We've focused on how to scale *up* resources, but all the same concepts work to scale *down* resources. It's important to identify the VM sizes in use and how much demand the applications make on those resources. Then you can use `az vm resize` to pick a VM size with fewer CPU cores and memory. Again, a VM restart is currently needed for any resize operation.

9.1.2 Scaling web apps vertically

Web apps can scale up or down based on resource needs in the same way that VMs do. When you created a web app in chapter 3, the default S1 Standard size provided one CPU core and 1.75 GB RAM. Each web app tier and size provides a set amount of

resources, such as CPU cores, memory, and staging slots. Even if the default size or resource allocation changes, or if you choose a different web app size, the concept remains the same.

If you create your web app and find that the application requires more resources than the service plan provides, you can change to a different tier, as shown in figure 9.3. The same process works if you have more resources than you need. Your web app can scale up or down manually in this way as needed.

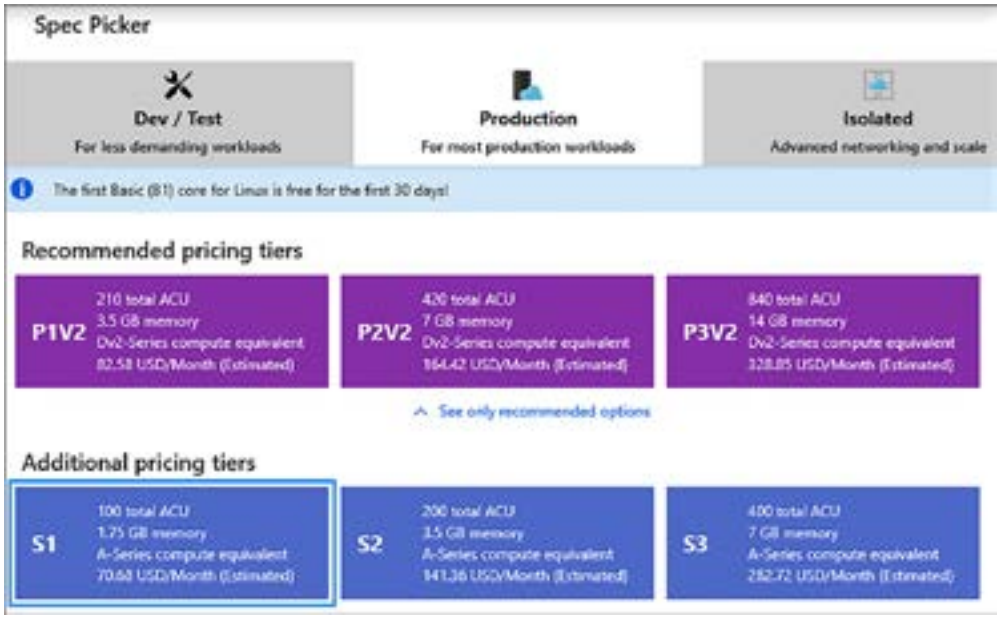


Figure 9.3 To manually scale a web app vertically, you change the pricing tier (size) of the underlying app service plan. The app service plan defines the amount of resources assigned to your web app. If your application requires a different amount of storage, number of CPUs, or deployment slots, you can change to a different tier to right-size the assigned resources to the application demand.

9.1.3 Scaling resources horizontally

A different approach to keep up with demand is to scale out horizontally. To scale vertically, you increase the amount of CPU and memory assigned to a single resource, such as a VM. To scale horizontally, you increase the number of VMs instead, as shown in figure 9.4.

To scale horizontally, your application does need to be aware of this ability and be able to process data without conflicts. A web application is a great candidate for scaling horizontally because the application can typically process data by itself.

As you build more complex applications, you may break an application into smaller individual components. If you think back to Azure storage queues from chapter 4, you

www.dbooks.org

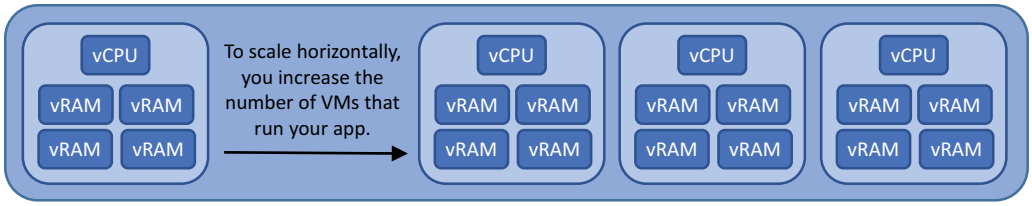


Figure 9.4 To deal with an increase in demand on your application, you can increase the number of VMs that run the application, distributing the load across multiple VMs rather than ever-larger single-instance VMs.

may have one application component that receives the frontend web orders and another application component that processes those orders and transmits them to the pizza store. The use of message queues is one approach to designing and writing applications that can operate in an environment that scales horizontally. This approach also lets you scale each application component separately and use different VM sizes or web app plans to maximize efficiency and reduce your monthly bill.

Historically, you'd scale vertically because it was easier to throw more compute resources at an application and hope it was happy. Setting up a cluster of resources and scaling an application horizontally were often complex in the physical world. With cloud computing and virtualization, the challenges of scaling horizontally are minimized to the point that you can often scale horizontally more quickly than vertically and without downtime.

Remember the `az vm resize` command earlier in this chapter? What happens as the VM resize operation completes? The VM is restarted. If that's the only instance of your application, no one can access it until it comes back online. When you scale horizontally, there's no downtime when you add VM instances; when the new VMs are ready, they start to process some of the application requests. The load-balancer health probes (chapter 8) automatically detect when a new VM in the backend pool is ready to process customer requests and traffic starts to be distributed to it.

Azure is designed to give you flexibility and choice when it comes to how you scale. If you're designing a new application environment, I suggest that you implement a horizontal scale approach. VMs have a cool cousin in Azure that can help you out here: virtual machine scale sets.

9.2 Virtual machine scale sets

VMs are some of the most common workloads in Azure, for good reason. The learning curve required to build and run a VM is shallow, because most of what you already know transfers straight to Azure. Web servers are among of the most common workloads for a VM, which again is convenient in that you don't have to learn new skills to transfer your knowledge to run Apache, IIS, or NGINX on an Azure VM.

What about a cluster of VMs that runs a web server? How would you handle that in your regular on-premises environment? There are many possible cluster solutions, to

start with. What about updates to your physical servers or VMs? How would you handle those? What if you wanted to automatically increase or decrease the number of instances in the cluster? Do you need to use another tool? Figure 9.5 shows an outline of a virtual machine scale set.

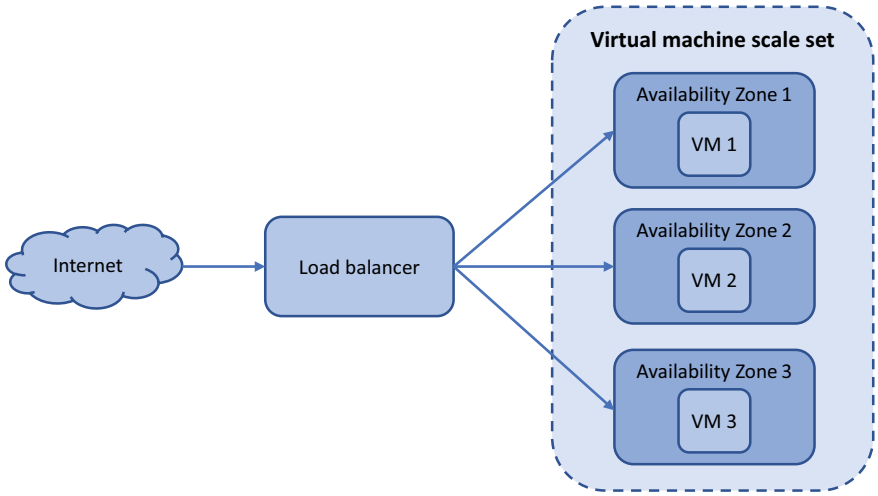


Figure 9.5 A virtual machine scale set logically groups together a set of VMs. The VMs are identical and can be centrally managed, updated, and scaled. You can define metrics that automatically increase or decrease the number of VMs in the scale set based on your application load.

A scale set simplifies how you run and manage multiple VMs to provide a highly available, load-balanced application. You tell Azure what size VM to use, a base image for the VM, and how many instances you want. Then you can define CPU or memory metrics to automatically increase or decrease the number of instances in response to the application load or on a schedule at peak customer hours. Scale sets combine the IaaS model of VMs with the power of PaaS features such as scale, redundancy, automation, and centralized management of resources.

A single-VM scale set?

If you build applications on VMs, plan to start out with a scale set, even if you only need one VM. Why? A scale set can expand at any time, and it automatically creates the connections to a load balancer or application gateway. If demand for the application suddenly increases in two months, you can tell the scale set to create an additional VM instance or two.

To expand a regular, standalone VM, you need to add that VM to a load balancer, and if you didn't begin with the VM in an Availability Set or Availability Zone, you have to plan how to make those VMs highly available. By creating a scale set to begin with, even for one VM, you futureproof your application with minimal additional work required.

www.dbooks.org

9.2.1 Creating a virtual machine scale set

Although a scale set makes it simpler to build and run highly available applications, you need to create and configure a few new components. That said, you can reduce the process to two commands to deploy a scale set with the Azure CLI.

Try it now

To create a scale set with the Azure CLI, complete the following steps:

1. Open the Azure portal, and select the Cloud Shell icon at the top of the dashboard.
2. Create a resource group with `az group create`; specify a resource group name, such as `azuremolchapter9`, and a location:

   ```
   az group create --name azuremolchapter9 --location westeurope
   ```

   Scale sets can use Availability Zones, so make sure to select a region where support is available.
3. To create a scale set, specify the number of VM instances you want and how the VM instances should handle updates to their configuration. When you make a change to the VMs, such as to install an application or apply guest OS updates, the VMs can update automatically as soon as they detect the change. Or you can set the upgrade policy to manual and apply the updates at a suitable time of your choice. The rest of the parameters should be familiar from when you created a single VM:

   ```
   az vmss create \
     --resource-group azuremolchapter9 \
     --name scalesetmol \
     --image UbuntuLTS \
     --admin-username azuremol \
     --generate-ssh-keys \
     --instance-count 2 \
     --vm-sku Standard_B1ms \
     --upgrade-policy-mode automatic \
     --lb-sku standard \
     --zones 1 2 3
   ```

That's it! You created multiple VMs across an Availability Zone that can scale. Get ready for what's really cool about the scale set you just created with the Azure CLI. Remember that entire chapter about load balancers (chapter 8), all those CLI commands you had to use, and how templates can simplify how you create a load balancer? That `az vmss create` command created and configured a load balancer for you!

Remember your quota limits

I mentioned this quota issue in chapter 7, but it's worth repeating in case you run into problems. In Azure, default quotas on your subscription prevent you from accidentally deploying resources and forgetting about them, which will cost you money! You can see the list of quotas at http://mng.bz/ddcx.

When you create multiple VMs, you may run into quota issues. You may also have issues if you don't delete resources from previous chapters and exercises. If you see error text along the lines of

```
Operation results in exceeding quota limits of Core.
Maximum allowed: 4, Current in use: 4, Additional requested: 2.
```

it's a good indication that you need to request an increase in your quotas. You can view your current quota for a given region as follows:

```
az vm list-usage --location westeurope
```

To request an increase in your quotas for a region, follow the steps outlined at http://mng.bz/Xq2f.

The Azure CLI helps you create a scale set with minimal prompts. A load balancer has been created and configured, a public IP address assigned, and the scale set VM instances added to the backend IP pool.

Try it now

Check out the resources created with your scale set, as described in the next commands.

To see what resources were created with your scale set, run the following command:

```
az resource list \
  --resource-group azuremolchapter9 \
  --output table
```

The output is similar to the following example. Look at the Type column for proof that a virtual network, public IP address, and load balancer were created:

```
Name      ResourceGroup      Type
--------  -----------------  ----------------------------------------------
mol       azuremolchapter9   Microsoft.Compute/virtualMachineScaleSets
molLB     azuremolchapter9   Microsoft.Network/loadBalancers
molLBIP   azuremolchapter9   Microsoft.Network/publicIPAddresses
molVNET   azuremolchapter9   Microsoft.Network/virtualNetworks
```

What does all this magic mean? When you create a scale set with the Azure CLI, a zone-redundant load balancer and public IP address are created for you. The VMs are created and added to a backend IP pool on the load balancer. NAT rules are created that allow you to connect to the VM instances. The only thing missing are load-balancer

www.dbooks.org

rules because they vary based on the applications you want to run. As you add VMs to or remove VMs from the scale set, the load-balancer configuration automatically updates to allow traffic to be distributed to the new instances. This magic isn't limited to the Azure CLI: if you use Azure PowerShell or the Azure portal, these supporting network resources are created and wired up to work together.

Try it now

Your scale was created with two instances. You can manually scale the number of VM instances in your scale set. When you do, the load balancer automatically updates the backend IP pool configuration. Set the `--new-capacity` of the scale set to four instances as follows:

```
az vmss scale \
    --resource-group azuremolchapter9 \
    --name scalesetmol \
    --new-capacity 4
```

9.2.2 Creating autoscale rules

When you created your scale set, you deployed a fixed number of instances. One of the most important features of scale sets is the ability to automatically scale in or out the number of VM instances that the scale set runs.

As shown in figure 9.6, the number of instances in a scale set can increase automatically as the application load increases. Think about a typical business application in your environment. Early in the workday, users start to access the application, which causes the resource load on those VM instances to increase. To ensure optimum application performance, the scale set automatically adds more VM instances. The load balancer starts to distribute traffic to the new instances automatically. Later in the workday, as users go home, application demand goes down. The VM instances use less resources, so the scale set automatically removes some VM instances to reduce unnecessary resources and lower cost.

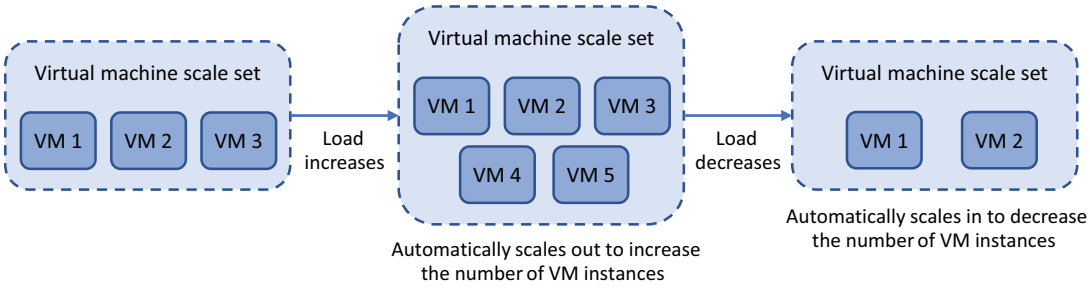


Figure 9.6 Scale sets can automatically scale in and out. You define rules to monitor certain metrics that trigger the rules to increase or decrease the number of VM instances that run. As your application demand changes, so does the number of VM instances. This approach maximizes the performance and availability of your application while minimizing unnecessary cost when the application load decreases.

You can base your scale-set rules on various metrics. You can look at host metrics for basic resource consumption, configure in-guest VM metrics collection for analysis of specific application performance counters, or use Azure Application Insights to monitor deep within the application code.

You can also use schedules to define a certain number of VM instances in a scale set for a time window. In the example of a common business application for which demand is higher during work hours than in the evening, you may want to define a higher fixed number of instances to run during business hours and define a lower fixed number of instances to run in the evening.

Autoscale rules based on metrics monitor performance over a defined time interval, such as five minutes, and may take another few minutes to spin up the new VM instances and configure them for application use. If you use fixed schedules to autoscale the number of VM instances in your scale set, those additional resources are already in use, and the load balancer distributes traffic to them throughout the day.

The use of schedules requires a baseline for the typical application demand and doesn't account for higher or lower demand at certain parts of the business account or sales cycle. You may end up with more resources than you need at times, so you pay more than necessary. And you may have situations in which the application load is higher than the number of VM instances the scale set can provide.

Try it now

To create autoscale rules for a scale set, complete the following steps:

1. Browse to and select Resource Group from the navigation bar at left in the Azure portal.
2. Choose the resource group you created for your template deployment, such as `azuremolchapter9`.
3. Select your scale set from the list of resources, such as scalesetmol.
4. Below Settings on the left side of the Scale Set window, choose Scaling. You can manually scale or create your own custom autoscale rules.
5. Choose to create custom autoscale rules.
6. Enter a name, such as `autoscale`, and then define a minimum, maximum, and default instance count. For this exercise, set the minimum to 2, maximum to 10, and default to 2.
7. Choose to add a rule, and then review the available rule settings, like those shown in figure 9.7.

   The default parameters look at the average CPU consumption. The rule triggers when the load is greater than 70% over a 10-minute interval. The scale set is increased by one VM instance, and the rules wait for 5 minutes before they begin to monitor and can trigger the next rule.

www.dbooks.org



Figure 9.7 When you add an autoscale rule, you define the exact behavior required for the rule to trigger.

This cool-down period gives the new VM instances time to deploy and begin to receive traffic from the load balancer, which should decrease the overall application load in the scale set. Without this cool-down period, the rules may trigger another VM instance to be added before the load has started to be distributed across the previously created VM instance.

8 To create the rule, select Add.

9 Choose to add another rule. This time, configure the rule to decrease the count by one when the average CPU load is less than 30% over a 5-minute duration.

10 Review your rules, like those shown in figure 9.8, and then select Save.

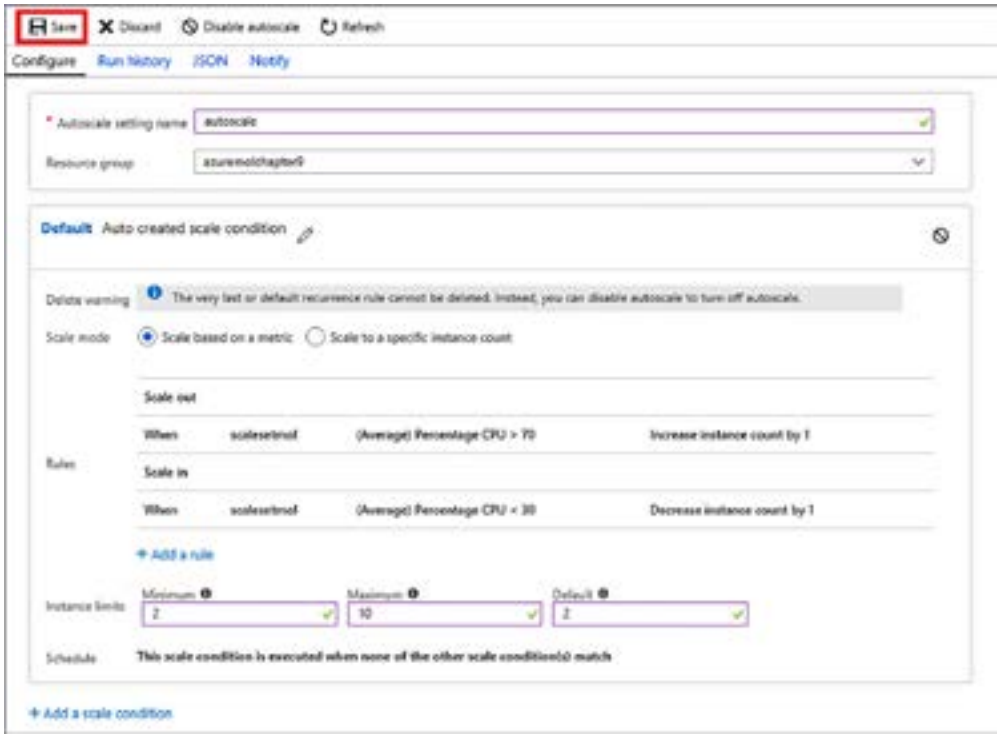


Figure 9.8 You should have one rule that increases the instance count by 1 when the average CPU load is greater than 70% and another rule that decreases the instance count by one when the average CPU load is less than 30%.

You can also configure autoscale rules with the Azure CLI, Azure PowerShell, or templates. The portal provides a nice visual way to review the rules and see available options for each parameter. As you build more complex rules, templates provide a way to create scale sets with the same set of rules in a reproducible fashion.

9.3 Scaling a web app

If you were super-interested in web apps in chapter 3, or in Azure tables and queues in chapter 4, these last three chapters, which have been heavy on IaaS VMs, may have left you scratching your head. Wasn't the cloud meant to be easier than this? For PaaS components such as web apps, absolutely!

www.dbooks.org

I don't want you to think the next few pages rush through how to provide the same high availability and autoscale capabilities to web apps. The truth is, it's a lot easier to do! As with most things, the choice between IaaS and PaaS is a balance of flexibility and ease of management. Much of the underlying redundancy is abstracted in PaaS services such as web apps, so you don't need a whole chapter on high availability and another chapter on load balancers.

The IaaS path to build and run your own VMs or scale sets with load balancers and availability zones may come from a business need or restriction. Developers, operations engineers, or tools and workflows may not be ready to go all-in to web apps. That said, I strongly urge you to look at web apps for new application deployments. The use of PaaS components such as web apps gives you more time to focus on the apps and your customers rather than infrastructure and administration.

Try it now

To create a web app with the Azure CLI, complete the following steps:

1. In chapter 3, you created a web app in the Azure portal. As with most resources, it's often quicker and easier to use the Azure CLI. Open Cloud Shell in the Azure portal.
2. Create an app service plan that's S1 Standard size. This size allows you to autoscale up to 10 instances of your web app:

```
az appservice plan create \
  --name appservicemol \
  --resource-group azuremolchapter9 \
  --sku s1
```

3. Create a web app that uses a local Git repo for deployment, as you did in chapter 3:

```
az webapp create \
  --name webappmol \
  --resource-group azuremolchapter9 \
  --plan appservicemol \
  --deployment-local-git
```

All the concepts and scenarios for autoscale rules and scale-set schedules discussed in section 9.2.2 also apply to web apps. As a quick recap, here are a couple of common scenarios for autoscaling web apps:

- Automatically increase or decrease the number of web app instances based on performance metrics to support application demand throughout the workday.
- Schedule a web app to automatically increase the number of instances at the start of the workday and then decrease the number of instances at the end of the workday.

In the case of the pizza store, the web app may receive more traffic later in the day and throughout the evening, so there isn't one set of autoscale rules that applies to every situation. Again, you need to baseline your application performance to understand how it runs under normal use and the performance metric at which the app needs to scale out or in. Even when you use autoscale schedules, you should continue to monitor and track when your peak application demands are to create rules that support that use pattern.

Try it now

To create autoscale rules for a web app, complete the following steps:

1. Browse to and select Resource Group from the navigation bar at left in the Azure portal.
2. Choose the resource group you created for your web app, such as azuremolchapter9.
3. Select your web app from the list of resources, such as webappmol.
4. Below Settings on the left side of the web app window, choose Scale Out (App Service Plan).
5. Again, choose to configure custom autoscale rules, not just manually scale the web app.
6. Enter a name, such as `autoscalewebapp`, and then define a minimum, maximum, and default instance count. For this exercise, set the minimum to 2, maximum to 5, and default to 2.
7. Choose to add a rule, and then review the available rule settings. This window looks the same as the autoscale rules for scale sets. The default parameters look at the average CPU consumption and trigger when the load is greater than 70% over a 10-minute interval. The web app is increased by one instance, and the rules wait for 5 minutes before they begin to monitor and can trigger the next rule.
8. Choose to add another rule. This time, configure the rule to decrease count by one when the average CPU load is less than 30% over a 5-minute duration.
9. Review and then save your rules.

When your autoscale rules trigger the web app to scale out or scale in, the Azure platform updates the traffic distribution to the available web app instances. There's no load balancer exposed to you, as you have with scale sets, but the traffic is still automatically distributed across the web app instances as your environment scales out or in. The concept is similar, just abstracted away from you, because you're meant to enjoy the PaaS approach and not worry so much!

www.dbooks.org

Both scale sets and web apps provide a way to build rules that automatically scale the number of instances that run your applications. With multiple instances to run your application, you also increase the availability of your app. Scale sets are a good middle ground between developers and business decision makers who want or need to build applications on VMs while using PaaS-like features to autoscale and reconfigure the flow of customer traffic.

In chapter 11, we'll look at Azure Traffic Manager, which really completes these high-availability deployments. Right now, you're still not quite production ready in terms of being able to offer multiple redundant scale sets or web app instances with traffic automatically distributed across them. We'll get to that soon, though!

9.4 Lab: Installing applications on your scale set or web app

We've covered a lot in this chapter, so now you can choose a quick final lab for either scale sets or web apps. Or if you want to extend your lunch break, do both!

9.4.1 Virtual machine scale sets

You have multiple VM instances in your scale sets, but they don't do a lot right now. For an overview of the different ways to install applications to VM instances in a scale set, see http://mng.bz/9Ocx. In practice, you'd use one of those automated deployment methods, but for now, manually install a web server on the VM instances as you did in chapter 8:

1. Remember load-balancer NAT rules? By default, each VM instance in a scale set has a NAT rule that allows you to SSH directly to it. The ports aren't on the standard TCP port 22. View the list of VM instances in a scale set and their port numbers as follows:

```
az vmss list-instance-connection-info \
  --resource-group azuremolchapter9 \
  --name scalesetmol
```

2. To connect to a specific port via SSH, use the `-p` parameter as follows (provide your own public IP address and port numbers):

```
ssh azuremol@40.114.3.147 -p 50003
```

3. Install a basic NGINX web server on each VM instance with `apt install`. Think back to how you did that in chapter 8.
4. To see the scale set in action, open the public IP address of the scale set load balancer in a web browser.
5. If you run into problems, make sure that the load balancer correctly created a load-balancer rule for TCP port 80 and has an associated health probe for either TCP port 80 or your own custom HTTP health probe that looks for /health.html on the VM.

9.4.2 Web apps

To deploy your application to a web app that runs multiple instances, the process is the same as the single web app from chapter 3. You push the application to the local Git repository for the web app, and thanks to the power of PaaS, the Azure platform deploys that single code base to multiple web app instances:

1. Initialize a Git repo in azure-mol-samples-2nd-ed/09, and then add and commit the sample files as you did in chapter 3:

```
cd azure-mol-samples-2nd-ed/09
git init && git add . && git commit -m "Pizza"
```

2. Your web app has a local Git repository. Add a remote for your web app the same way you did in chapter 3:

```
git remote add webappmolscale <your-git-clone-url>
```

3. Push this sample to your web app. This makes a single code commit, but then your app is distributed across the multiple web app instances:

```
git push webappmolscale master
```

www.dbooks.org

10 Global databases with Cosmos DB

Data. You can't get by without it. Almost every application that you build and run creates, processes, or retrieves data. Traditionally, this data has been stored in a structured database such as MySQL, Microsoft SQL, or PostgreSQL. These large, structured databases are established and well known, have ample documentation and tutorials, and can be accessed from most major programming languages.

With great power comes great responsibility, and a lot of infrastructure overhead and management typically go with these traditional structured databases. That's not to say you shouldn't use them—far from it—but when it comes to applications that run on a global scale, it's no easy feat to also build clusters of database servers that replicate your data and intelligently route customers to your closest instance.

That's where Azure Cosmos DB becomes your best friend. You don't need to worry about how to replicate your data, ensure consistency, and distribute customer requests. Instead, you add data in one of the many models available, and then choose where you want your data to be available. In this chapter, you'll learn about unstructured database models in Cosmos DB, how to create and configure your database for global distribution, and how to build web applications that use your highly redundant and scalable Cosmos DB instance.

10.1 What is Cosmos DB?

Chapter 4 started to explore unstructured databases with Azure storage tables. The example was basic, but the concepts are the foundations of Cosmos DB. First, let's take a step back and examine what we mean by *structured* and *unstructured* databases.

10.1.1 Structured (SQL) databases

Structured databases are the more traditional approach to storing data. A *structure*, or *schema*, to the database defines how the data is represented. Data is stored in tables, with each row representing one item and a fixed set of values assigned to it. If we take the pizza-store model, each row in a table that stores the types of pizza may indicate the name of the pizza, its size, and the cost. A basic SQL database is shown in figure 10.1.

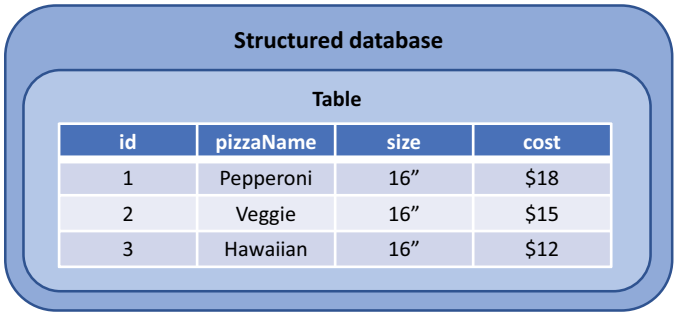


Figure 10.1 In a structured database, data is stored in rows and columns within a table. Each row contains a fixed set of columns that represent the schema for the database.

In structured databases, each server typically must contain the entire database for queries and data retrieval to succeed. The data is joined in queries to pull from different tables based on criteria the developer builds as part of the structured query. This is where the term *Structured Query Language* (SQL) comes from. As databases grow in both size and complexity, the servers that run the database must be sufficiently sized to handle that data in memory. That becomes difficult, and costly, with very large datasets. Given that they need a structure, it also makes it difficult to add properties and change the structure later.

10.1.2 Unstructured (NoSQL) databases



The unstructured data in NoSQL databases isn't stored in tables of rows and columns; rather, it's stored in dynamic arrays that allow you to add new properties for an item as needed. One big advantage of this approach is that you can quickly add a new pizza type or specialty topping without changing the underlying database structure. In a structured database, you'd need to add a new column to a table and then update the application to handle the additional column. In NoSQL databases, you add another property to a given entry from your code; see figure 10.2.

Figure 10.2 In an unstructured database, data is stored without fixed mappings of columns to a row in a table. You can add toppings to a single pizza, for example, without updating the entire schema and other records.

www.dbooks.org

NoSQL databases also offer different database models. These models give an indication of how the data is stored and retrieved in the database. Which model you use varies based on the size and format of the data you work with, and how you need to represent the data in your application. These models include document, graph, and table. Don't caught get too caught up in the models for now; different models work better for different sets of unstructured data, depending on how you need to relate and query the data. The key takeaway is that unstructured NoSQL databases have a different underlying concept in how they store and retrieve data, which you can use to your advantage as you build and run cloud applications in Azure.

10.1.3 Scaling databases

Remember that I said that for a structured database, the entire database typically needs to exist on each server? As you get into very large databases, you need ever-larger servers to run them. You may never work with databases that grow to hundreds of gigabytes or even terabytes, but NoSQL databases approach how databases grow and scale differently from SQL databases. The difference is that NoSQL databases typically scale horizontally rather than vertically.

There's a limit to how much you can vertically scale a VM—that is, give it more memory and CPU. You start to encounter performance issues in other parts of the compute stack as you squeeze out the maximum storage throughput and network bandwidth. And that's without the hit to your wallet (or your boss's wallet) when you see the bill for such large VMs. As a recap from chapter 9, vertical scaling is illustrated in figure 10.3. Now imagine a cluster of such large database VMs, because you want redundancy and resiliency for your application, right?

By contrast, scaling horizontally allows you to run database VMs with less resources and a lower price to go along with them. To do this, NoSQL databases split data across database nodes and route requests from your application to the appropriate node. The other nodes in the cluster don't need to be aware of where all the data is stored;

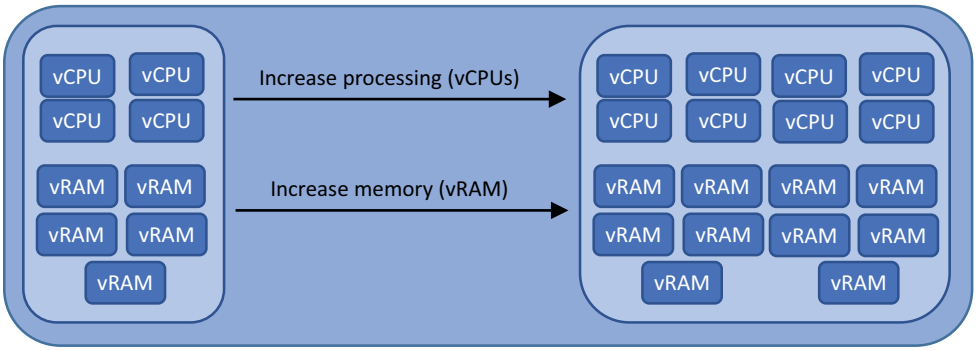


Figure 10.3 Traditional structured databases scale vertically. As the database grows, you increase the amount of storage, memory, and CPU power on the server.

they just need to respond to their own requests. You can quickly add nodes to a cluster in response to customer demand as needed.

As a result, in a NoSQL database, the entire database doesn't need to fit in the memory of a host. Only part of the database—a *shard*—needs to be stored and processed. If your application works with large amounts of *structured* data, a NoSQL database may hurt performance because the different hosts are queried for their pieces of information to return to the customer. If you have a large amount of *unstructured* data to process, NoSQL databases may offer a performance improvement, if not a management and efficiency benefit. An example of how unstructured databases scale horizontally across hosts is shown in figure 10.4.

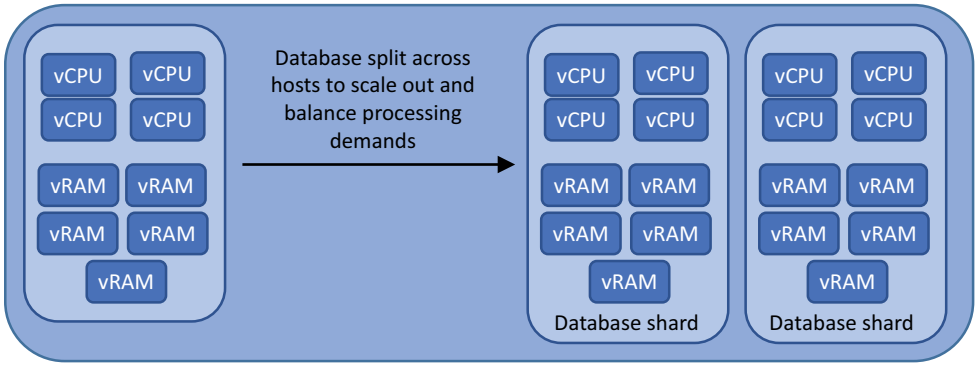


Figure 10.4 Unstructured NoSQL databases scale horizontally. As a database grows, it's sharded into segments of data that are distributed across the database servers.

10.1.4 Bringing it all together with Cosmos DB

So, what is Cosmos DB? It's an autoscaling, globally distributed database platform that allows you to use various forms of NoSQL databases. As with services like Web Apps, Cosmos DB abstracts a lot of the management layer from you. When you create a web app, you don't need to configure load balancing or clustering; you choose your regions and can configure autoscaling and then upload your application code. The Azure platform handles how to replicate and distribute the web app traffic in a highly available way. With Cosmos DB, you don't worry about how large a database you need, how much memory to assign, or how to replicate data for redundancy. You choose how much throughput you may need and what regions to store your data in; and then you start adding data.

This chapter uses an SQL model for Cosmos DB, but the data is stored in a NoSQL JSON format. These may be new concepts, but stick with me. You can use other models, including Mongo, Cassandra, Gremlin, and Table. The functionality is the same for all of them: pick your model, choose your regions, and add your data. That's the power of Cosmos DB.

www.dbooks.org

10.2 Creating a Cosmos DB account and database

Let's see Cosmos DB and unstructured databases in action, which we can do in a couple of ways. The first is to use the Azure portal to create an account, select and create a database model, and enter data into the database so that your app can query it. Or you can use the Azure CLI, Azure PowerShell, or language-specific software development kits (SDKs) to create it all in code. Let's use the Azure portal so that we can also visually create and query the data.

10.2.1 Creating and populating a Cosmos DB database

In chapter 4, you created your first NoSQL database with an Azure storage table. Let's use Cosmos DB to create a similar database, this time one that offers all the geo-redundancy and replication options to make sure that the online store allows customers to order pizza without any downtime. Let's create a Cosmos DB account and a document database, and then add some data entries for three types of pizza, as shown in figure 10.5.

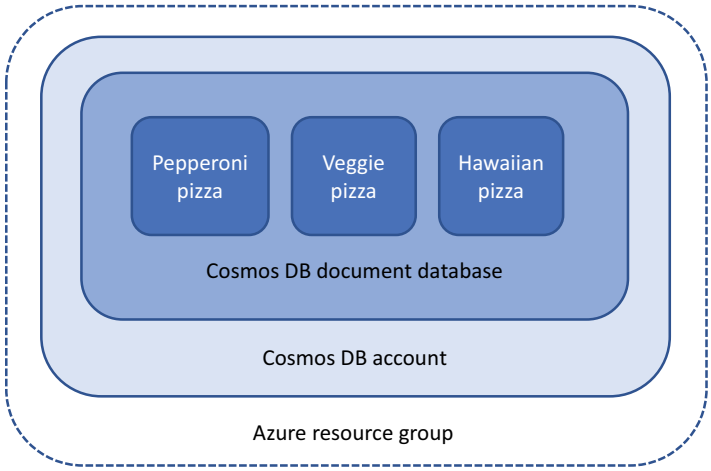


Figure 10.5 In this section, you'll create a resource group and a Cosmos DB account. A document database is created in this account, and you'll add three entries to represent a basic menu for your pizza store.

Try it now

To see Cosmos DB in action, create an account using the Azure portal:

1. Open the Azure portal, and select Create a Resource in the top-left corner of the dashboard.
2. Search for and select Azure Cosmos DB, and then choose Create.
3. Choose to create a resource group, such as `azuremolchapter10`, and enter a unique name for your Cosmos DB account, such as `azuremol`.

4. The type of model you can use for your database is referred to as the API. For this example, choose Core (SQL) from the drop-down menu.
5. For Location, select East US. Cosmos DB is available in all Azure regions, but for this example, the web application you deploy in the end-of-chapter lab expects you to use East US.
6. Leave the option for georedundancy as disabled, along with any other additional features, such as multiwrite regions. Section 10.2.2 dives into how to replicate your database globally.

Secure traffic with service endpoints

You have an option to connect your Cosmos DB to an Azure virtual network with something called a *service endpoint*. We won't discuss this option now, but it's a cool feature that helps secure your instance by allowing access to the database only from a defined virtual network.

If you build middleware applications that use Cosmos DB, or internal-only applications, you can use a virtual network service endpoint to scope down access from a specific virtual network, not over the internet and with a public endpoint. A growing number of Azure services support these kinds of endpoints, and it's another example of giving you options to secure your environment to fit your business requirements.

7. When you're ready, review and create your Cosmos DB account. It takes a few minutes to create the account.

Your database is empty right now, so let's explore how you can store some basic data for your pizza-store menu. Cosmos DB groups the data in a database into something called a *container*. No, that's not the same kind of container that's the driving force behind Docker, Kubernetes, and cloud-native applications that you may have heard of. This naming confusion isn't a plus, but stick with me for now.

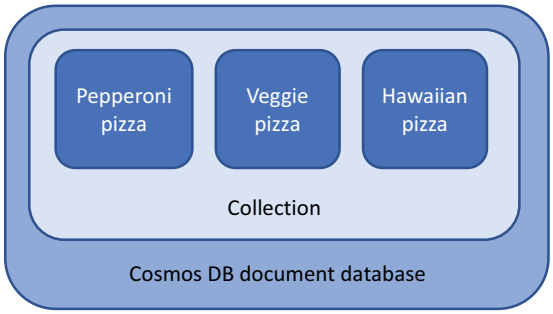


Figure 10.6 A Cosmos DB database that uses the document model stores data in collections. These collections let you group data for quicker indexing and querying.

In Cosmos DB databases that use the document model, data is logically grouped into containers called *collections*. Other API models have a slightly different name for the container entity, such as *graph* for the Gremlin API. For our SQL API, collections store related pieces of data that can be quickly indexed and queried, as shown in figure 10.6. Collections aren't totally dissimilar to how you organize a traditional SQL database into tables, but collections offer a lot more flexibility when it comes to distributing the data for performance or redundancy.

www.dbooks.org

Because Cosmos DB is designed to handle very large amounts of data and throughput, you can choose how to size and control the flow, and cost, of that data. Throughput is calculated in request units per second (RU/s), and one request unit is the equivalent of 1 KB of document data. Essentially, you define how much bandwidth you want your database to have. In case you haven't guessed, the more bandwidth (RU/s) you want, the more you pay. Cosmos DB shows you how much data you're using and how much throughput your application uses, and you typically don't need to worry too much about right-sizing things. For your pizza store, let's not start out too crazy, though!

Try it now

To create a collection and populate some entries in the database, complete the following steps:

1. Browse to and select Resource Group from the navigation bar at left in the Azure portal.
2. Choose the resource group in which you created your Cosmos DB database, such as azuremolchapter10.
3. Select your Cosmos DB account from the list of resources, and then select the Overview page.
4. Choose to add a container.
5. This is your first database, so enter a name, such as `pizzadb`.
6. Leave throughput set to the default value.
7. For the container ID, enter `pizzas`. This step creates a logical container that you can use to store the items on your pizza-store menu.
8. Enter a partition key of `/description` to make sure that pizza types are evenly distributed.

   The partition key identifies how the data could be split apart in the database. It's not really needed in a small sample database like this one, but using it is good practice as your app scales.
9. Don't choose to add a unique key. Keys further logically define the container, such as for subsections of food customers can order. The wider collection is for your menu, but in much larger databases, you may want partition keys for things like pizzas, drinks, and desserts.
10. To create the database and collection, select OK.

Now you have a Cosmos DB account, a database, and a collection, but Cosmos DB still doesn't contain your pizzas. You could import some data or write code that enters a bunch of data. Let's create three pizzas manually to explore some of the graphical tools built in to the Azure portal for browsing, querying, and manipulating the data in your Cosmos DB database.

Try it now

To create add some entries to the database, complete the following steps, as shown in figure 10.7:

1 In your Cosmos DB account, chose Data Explorer from the menu at left in the Overview window.

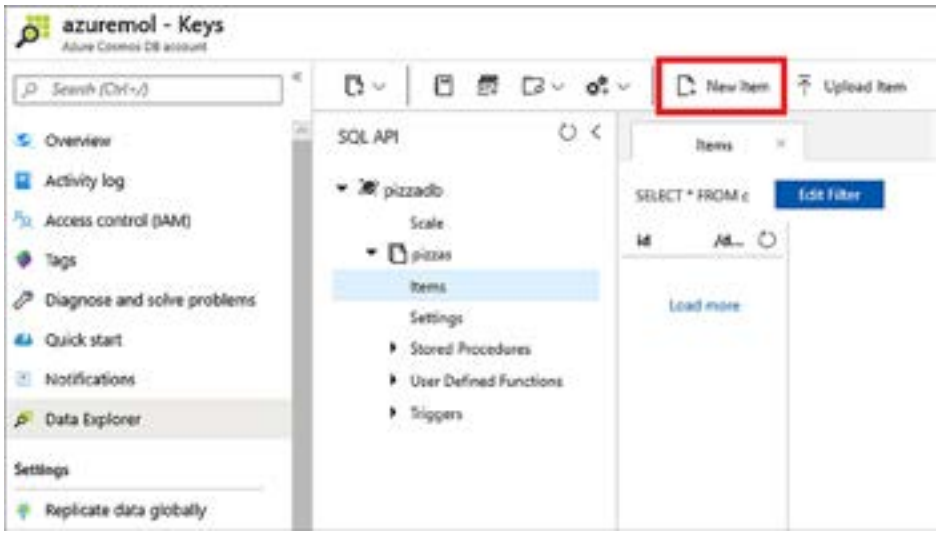


Figure 10.7 With the Data Explorer in the Azure portal, you can browse your collections to query or create new documents. This graphical tool lets you manage your database quickly from a web browser.

2 Expand first the `pizzadb` database, and then the `pizzas` collection.

3 Add a new item to put some pizzas in the database. Data is added in JSON format.

4 In the text box, replace any existing text with the following data to create a new menu item for a basic pepperoni pizza:

```
{
    "description": "Pepperoni",
    "cost": "18"
}
```

5 To add the data to the database, select Save.

www.dbooks.org

6 Add another pizza to your menu. This time, add a property to indicate that this pizza has a gluten-free crust. You don't need to do anything special to the underlying database; just add another property to your data. To add another new item, enter the following data, and select Save:

```
{
    "description": "Veggie",
    "cost": "15",
    "gluten": "free"
}
```

7 Add one final type of pizza. This time, add a property that includes what toppings are on the pizza. To add one more new item, enter the following data, and select Save:

```
{
    "description": "Hawaiian",
    "cost": "12",
    "toppings": "ham, pineapple"
}
```

These three entries show the power of a NoSQL database. You added properties to the entries without needing to update the database schema. Two different properties showed that the veggie pizza has a gluten-free crust and what toppings are on the Hawaiian pizza. Cosmos DB accepts those additional properties, and now that data is available to your applications.

Some extra JSON properties get added for things like `id`, `_rid`, and `_self`. These aren't properties you need to worry too much about for now. Cosmos DB uses these properties to track and identify the data; you shouldn't manually edit or delete them.

10.2.2 Adding global redundancy to a Cosmos DB database

You have a Cosmos DB database that stores a basic pizza menu in the East US region. But your pizza store is ready to open franchises all around the world! You want to replicate the data about your pizzas to Azure regions in different locations, close to your new customers.

Why would you want to do this? If all your customers read and write data from the database in one region, that's potentially a lot of traffic crossing under-ocean cables and routing around the world. To provide the best low-latency experience to customers, you can replicate your data to Azure regions around the world, and customers can connect to the closest replica to them, as shown in figure 10.8.

Consistency models and guarantees are built into the Cosmos DB platform to handle data consistency and replication for you. You designate one or more regions as the primary write location. This book's examples use a single write point, but you can use multimaster support to write data to the closest endpoint, which is then propagated

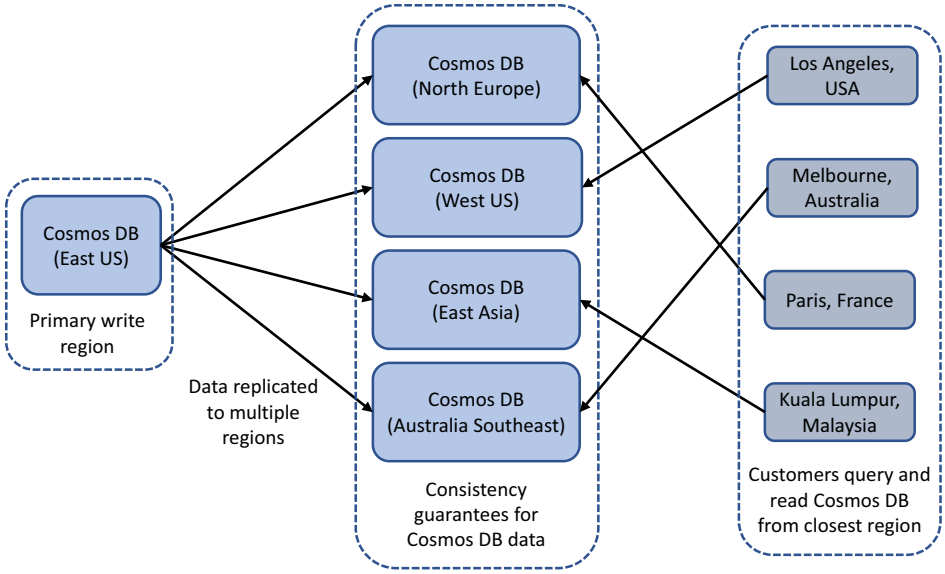


Figure 10.8 Data is replicated from one primary Cosmos DB instance to multiple Azure regions around the world. Then web applications can be directed to read from their closet region, and customers can be dynamically routed to the closest location to minimize latency and improve response times.

asynchronously to other regions. The data is also rapidly replicated to the read regions that you designate. You can control the order of failover to designate read regions and, with your application, automatically or manually specify regions to read from.

You can define a consistency model—which is more of a design consideration than an operational one—to define how quickly writes across multiple regions are replicated. The consistency models range from *strong*, which waits on replicated writes to be confirmed by replicas and so guarantees reads are consistent, to *eventual*, which is more relaxed. The eventual model guarantees that all the data is replicated, but there may be a slight delay when reads from replicas return different values until they're all in sync.

There's a balance between a more limited geographic distribution, such as with the strong consistency model, and a wider geographic replication, such as with the eventual consistency model, but with the understanding that there's a slight delay as the data is replicated. There are also bandwidth and processing costs, depending on how consistently and timely you wish the data to be replicated. The Azure platform handles the underlying replication of data from your write point; you don't need to build your applications to replicate the data or determine how best to read data from replicated endpoints.

On a global scale, you could have multiple VMs or web apps like the ones you created in previous chapters, but in different regions around the world. Those apps

www.dbooks.org

connect to a local Cosmos DB instance to query and read all their data. Through some cool Azure network traffic features we'll discuss in chapter 11, users can be routed automatically to one of these local web application instances, which also use a local Cosmos DB instance. In the event of regional outages or maintenance, the entire platform routes the customer to the next-closest instance.

In the traditional structured database world, in which you manage the VMs, database install, and cluster configuration, such a setup takes serious design planning and is complicated to implement. With Cosmos DB, the process takes three mouse clicks. Honestly!

Try it now

To replicate your Cosmos DB data globally, complete the following steps:

1. Browse to and select Resource Group from the navigation bar at left in the Azure portal.
2. Choose the resource group in which you created your Cosmos DB database, such as `azuremolchapter10`.
3. Select your Cosmos DB account from the list of resources. Those two mouse clicks were free, but start counting from here!
4. Select the menu option at left to replicate data globally. The map, which shows all the available Azure regions, shows that your database is currently available in the East US region (figure 10.9).



Figure 10.9 Select an Azure region to replicate your Cosmos DB database to, and then choose Save. Those are all the steps required to globally distribute your data.

5 Choose West Europe, and then select Save. You can choose any Azure region you wish, but the end-of-chapter lab expects your data to be replicated to West Europe. It takes a few moments to replicate the data to the region you selected and bring the data online for your applications to use.

Okay, count those mouse clicks! Three clicks, right? Let's be generous and consider the first two mouse clicks to select the resource group and Cosmos DB account. So in no more than five mouse clicks and a matter of seconds, you created a replica instance of your database that allows your applications to access data from the closest region to them. Can you do that with a traditional MySQL cluster? Please tweet me @fouldsy if you can do that this quickly outside of Cosmos DB!

With your database now distributed globally, does it take a lot of changes to your code to determine which Cosmos DB region to connect to? How can you maintain all these different versions of your applications based on what Azure region they run? Easy—let the Azure platform determine it all for you!

10.3 Accessing globally distributed data

For the most part, the Azure platform determines the best location for your application to talk to. An application typically needs to read and write data. You can define the failover policies for your Cosmos DB database, which controls the primary write location. This write location acts as the central hub to ensure that data is consistently replicated across regions. But your web app can typically read from multiple available regions to speed the queries and return data to the customer. All this is handled by REST calls.

Let's see what happens from the Azure CLI when you ask for information about a Cosmos DB database. This process is like an application making a connection to a database, but it stops you from getting too deep into the code.

Try it now

Use `az cosmosdb show` to find information about your read and write location:

1 Open the Azure portal in a web browser, and then open Cloud Shell.

2 Use `az cosmosdb show` to view the read and write locations for your Cosmos DB database.

Enter the resource group name and database name you created in the previous "Try it now" exercises. In the following example, the resource group is `azuremolchapter10`, and the Cosmos DB database name is `azuremol`:

```
az cosmosdb show \
  --resource-group azuremolchapter10 \
  --name azuremol
```

www.dbooks.org

A lot of output is returned from this command, so let's examine the two key parts: read locations and write locations. Here's some example output for the `readLocations` section:

```
"readLocations": [
    {
      "documentEndpoint":"https://azuremol-eastus.documents.azure.com:443/",
      "failoverPriority": 0,
      "id": "azuremol-eastus",
      "isZoneRedundant": "false",
      "locationName": "East US",
      "provisioningState": "Succeeded"
    },
    {
      "documentEndpoint":
      ➥"https://azuremol-westeurope.documents.azure.com:443/",
      "failoverPriority": 1,
      "id": "azuremol-westeurope",
      "isZoneRedundant": "false",
      "locationName": "West Europe",
      "provisioningState": "Succeeded"
      }
],
```

When your application makes a connection to a Cosmos DB database, you can specify a connection policy. If databases aren't normally your thing, think of a basic Open Database Connectivity (ODBC) connection you may create on a Windows machine. The connection string typically defines a hostname, a database name, a port, and credentials. Cosmos DB is no different. You can connect to Cosmos DB from multiple languages, including .NET, Python, Node.js, and Java. The languages may differ, but all the SDKs have a similar setting: endpoint discovery. Two main properties of the connection policy are important:

- *Automatic endpoint discovery*—The SDK reads all the available endpoints from Cosmos DB and uses the failover order specified. This approach ensures that your application always follows the order you specify at the database level. You may want all reads to go through East US, for example, and use West Europe only when there's maintenance in the primary location.
- *Preferred endpoint locations*—You specify the locations you wish to use. An example is if you deploy your app to West Europe and want to ensure that you use the West Europe endpoint. You lose a little flexibility as endpoints are added or removed, but you make sure that your default endpoint is close to your app without needing more advanced network routing to help determine this for you.

Typically, your application lets the Cosmos DB SDK handle this task. Your application doesn't change how it handles the connection to the database: it just knows that it *can* connect to different locations. But the SDK is what *makes* the connection and uses this location awareness.

Figure 10.10 shows a simplified approach to how this location awareness is used between your application and the SDK. Again, the language doesn't matter, and the approach is the same; the figure uses the Python SDK because that's what a couple of the examples have been written in. This example also assumes that you're using automatic endpoint locations.

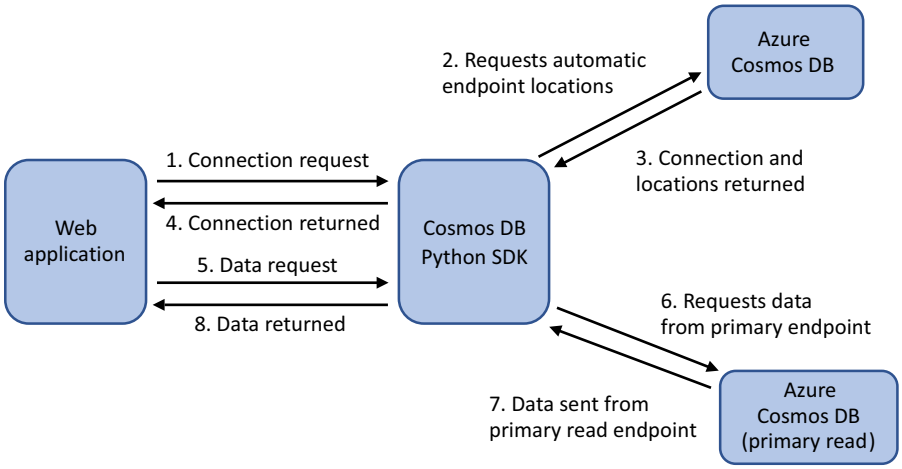


Figure 10.10 The flow of requests through a Cosmos DB SDK when an application uses location awareness to query Cosmos DB

The steps illustrated in figure 10.10 are as follows:

1. Your application needs to make a connection to a Cosmos DB database. In the connection policy, you enable automatic endpoint discovery. The application uses the Cosmos DB SDK to make a database connection.
2. The Cosmos DB SDK makes a connection request and indicates that it wishes to use automatic endpoint locations.
3. A connection is returned based on the credentials and database requested.
4. The SDK returns a connection object for the application to use. The location information is abstracted from the application.
5. The application requests some data from the Cosmos DB database. The SDK is again used to query and obtain the data.
6. The SDK uses the list of available endpoints and makes the request to the first available endpoint. Then the SDK uses the connection endpoint to query the data. If the primary endpoint is unavailable, such as during a maintenance event, the next endpoint location is used automatically.
7. Cosmos DB returns the data from the endpoint location.

www.dbooks.org

8 The SDK passes the data from Cosmos DB back to the application to parse and display as needed.

The last things to look at in Cosmos DB are access keys, which allow you to control who can access the data and what permissions they have. Keys can be regenerated, and as you do with passwords, you may want to implement a policy to regularly perform this key-regeneration process. To access the distributed data in Cosmos DB, you need to get your keys. The Azure portal provides a way to view all the keys and connection strings for your database.

Try it now

To view the keys for your Cosmos DB account, complete the following steps:

1 Browse to and select Resource Group from the navigation bar at left in the Azure portal.
2 Choose the resource group in which you created your Cosmos DB database, such as `azuremolchapter10`.
3 Select your Cosmos DB account from the list of resources.
4 On the left side, choose Keys.
5 Make a note of the URI and primary key (figure 10.11). You'll use these values in the end-of-chapter lab.



Figure 10.11 The Keys section of your Cosmos DB account lists the connection information and access keys. You need this information when you build and run applications, such as in the end-of-chapter lab.

A lot in Cosmos DB happens under the hood to distribute your data and allow your applications to read and write from the most appropriate locations. But that's the whole point. An awareness of what the Cosmos DB service does helps you design and plan your application, or troubleshoot if applications don't let the SDK perform read and write operations as needed. But you don't need to worry about the how and when; focus on your applications, and use Azure services like Cosmos DB to provide the cloud functionality and benefits that allow you to operate on a global scale.

10.4 Lab: Deploying a web app that uses Cosmos DB

In section 10.2.2, you distributed your Cosmos DB database globally. Then we went over a bunch of theory as to how web applications can read from locations around the world. Now you probably just want to see Cosmos DB in action, so here's your chance! In this lab, the basic web app from previous chapters is used, but this time, the pizza menu comes from the items you added to the Cosmos DB database in an earlier "Try it now" exercise:

1. In the Azure portal, create a web app.
2. As the pizza store isn't a basic HTML page anymore, pick `Node LTS` for the run time that runs on Linux.
3. When the web app is ready, create a deployment source (local Git repository). The steps are the same as when you created one in previous chapters, such as chapter 3, so check those exercises if you need a refresher.
4. Open Cloud Shell. In earlier chapters, you obtained a copy of the Azure samples from GitHub. If you didn't, grab a copy as follows:

   ```
   git clone https://github.com/foouldsy/azure-mol-samples-2nd-ed.git
   ```

5. Change to the directory that contains the Cosmos DB web app sample:

   ```
   cd ~/azure-mol-samples-2nd-ed/10/cosmosdbwebapp
   ```

6. Edit the configuration file with the database URI and access key that you copied in the previous "Try it now" exercise to view your Cosmos DB keys:

   ```
   nano config.js
   ```

7. Write out the file by pressing Ctrl-O, and then exit by pressing Ctrl-X.
8. Add and commit your changes in Git with the following command:

   ```
   git init && git add . && git commit -m "Pizza"
   ```

9. Create a link to the new Git repository in your staging slot with `git remote add azure`, followed by your Git deployment URL.
10. Use `git push azure master` to push your changes to your web app.
11. Select the URL to your web app from the Azure portal Overview window.

www.dbooks.org

12 Open this URL in a web browser to see your pizza store, which is now powered by Cosmos DB, as shown in figure 10.12.

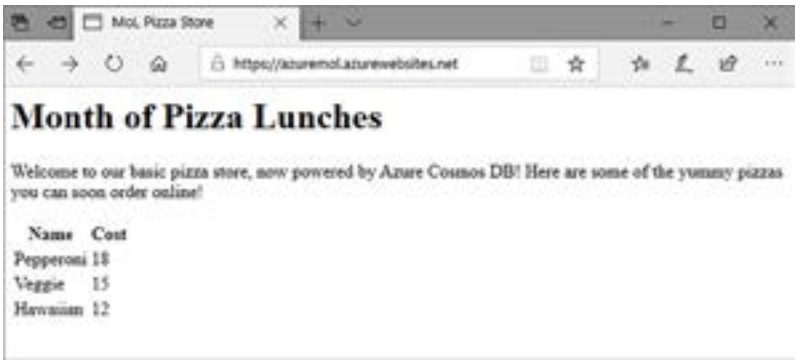


Figure 10.12 The basic Azure web app shows your short pizza menu based on data in the Cosmos DB database. The pizza store from previous chapters is shown, but now the list of pizzas and their prices is powered by Cosmos DB. The site is still basic, as the goal is for you to see the service in action and understand how you could start to build your own applications.

11 Managing network traffic and routing

Domain Name System (DNS) resolution is at the heart of almost every digital connection you make. It's how you browse the web, receive email, watch Netflix, and make Skype calls. DNS is the mechanism that translates a name, such as manning.com, into an IP address. When I want to learn a new topic, I don't need to remember 35.166.24.88; I just enter `manning.com` in a web browser and browse some books! Network devices route traffic based on IP addresses, so you need an approach that helps those of us with bad memories to do things like buy books or pizza online.

Over the past few chapters, you've spent a lot of time learning how to build applications that can scale, are highly available, and are globally distributed. One of the last missing pieces is how to direct customers from around the world to the most appropriate application instance—typically, the instance closest to them. Azure Traffic Manager makes it easy to automatically route customers to your application instances based on performance or geographic location. In this chapter, we'll discuss how you can create and manage DNS zones in Azure, and then how to use Traffic Manager to route customers with DNS queries, as shown in figure 11.1.

11.1 What is Azure DNS?

You don't need a deep understanding of how DNS works to complete this chapter and use Azure DNS. Figure 11.2 shows a high-level overview of how a user queries a DNS service to obtain the IP address for a web application. A lot of substeps could happen around steps 1 and 2, so if you have a little time left in your lunch break at the end of this chapter, feel free to read up on how DNS queries and recursion work.

www.dbooks.org

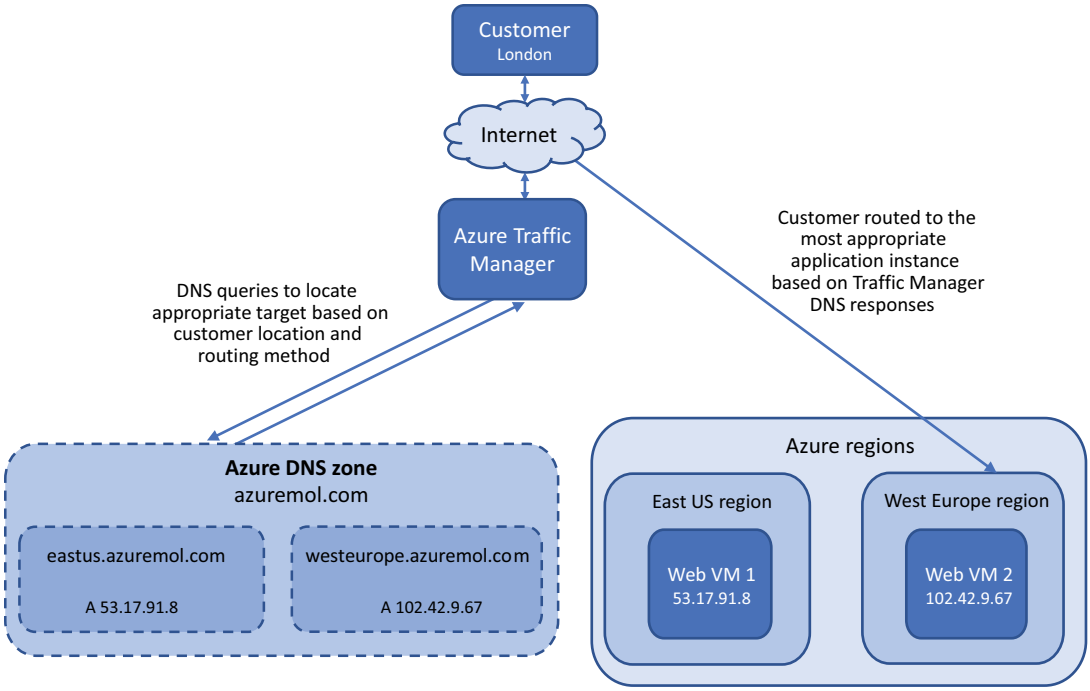


Figure 11.1 In this chapter, we'll examine how you can create DNS zones in Azure DNS. To minimize latency and improve response times, Traffic Manager can be used to query DNS and direct customers to their closet application instance.

Azure DNS functions the same as way any existing DNS solution you may use or be familiar with. Your zone and records are stored in Azure, and the name servers that respond to DNS queries are distributed globally across the Azure data centers.

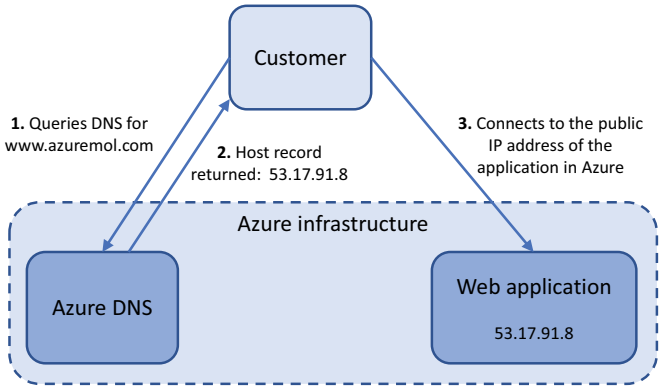


Figure 11.2 This simplified flow of DNS traffic shows how a user sends a DNS request for www.azuremol.com to a DNS server, receives a response that contains the associated IP address, and then connects to the web application.

Azure DNS supports all the record types you'd expect in a regular DNS service offering. Both IPv4 and IPv6 records can be created. The record types are as follows:

- *A*—IPv4 host records, to point customers to your applications and services
- *AAAA*—IPv6 host records, for you cool kids who use IPv6 to point customers to your applications and services
- *CNAME*—Canonical name, or alias, records, such as to provide a short name that's easier to use than the full hostname of a server
- *MX*—Mail exchange records to route email traffic to your mail servers or provider
- *NS*—Name server records, which include automatically generated records for the Azure name servers
- *PTR*—Pointer records, for reverse DNS queries to map IP addresses to hostnames
- *SOA*—Start-of-authority records, which include automatically generated records for the Azure name servers
- *SRV*—Service records, to provide network services discovery, such as for identity
- *TXT*—Text records, such as for Sender Protection Framework (SPF) or DomainKeys Identified Mail (DKIM)

In a typical DNS configuration, you configure multiple DNS servers. Even with geographic distribution of those servers for redundancy, customers may query a name server on the other side of the world. Those milliseconds required to query, resolve, and then request a response for the web application can add up when you have lots of customers wanting to order pizza.

An Azure DNS zone is replicated globally across the Azure data centers. *Anycast* networking ensures that when a customer makes a DNS query to your domain, the closest available name server responds to their request. How does anycast routing do this? Typically, a single IP address is advertised across multiple regions. Rather than using a simple DNS query that resolves back to a single IP address that only exists in one location, anycast routing allows the network infrastructure to intelligently determine where a request is coming from and route the customer to the closest advertised region. This routing allows your customers to connect to your web application more quickly and provides a better overall customer experience.

You don't need to be an expert at networking to fully understand how this works; Azure handles it for you! When you combine Azure DNS with Azure Traffic Manager (section 11.2), you not only return DNS queries from the closest name servers, but also connect customers to the closest application instance to them. Make those milliseconds count!

11.2 Delegating a real domain to Azure DNS

When you register a real domain, your provider gives you a management interface and tools to manage that domain. To allow customers to access your services and use the Azure DNS zone and records, you delegate authority of your domain to the Azure

www.dbooks.org

name servers. This delegation causes all DNS queries to be directed to those Azure name servers immediately, as shown in figure 11.3. Azure currently doesn't allow you to purchase and register domains within the platform, so you need to purchase the domain name through an external registrar and then point the NS records to the Azure name servers.

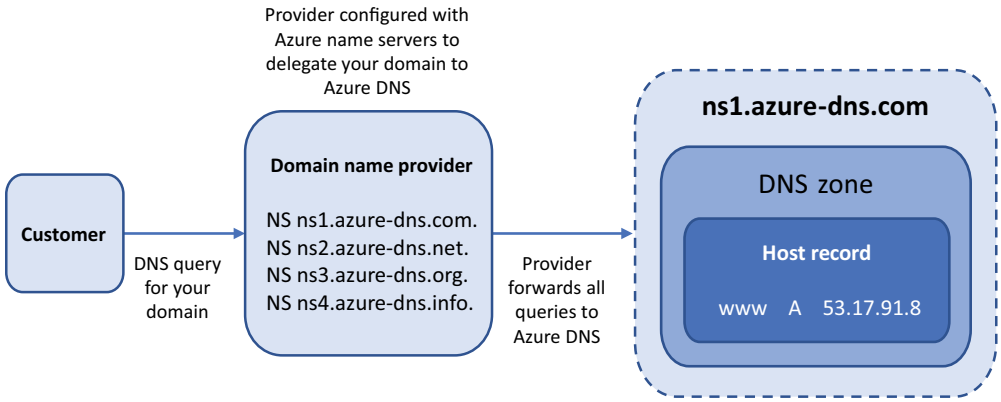


Figure 11.3 To delegate your domain to Azure, configure your current domain provider with the Azure name server addresses. When a customer makes a DNS query for your domain, the requests are sent directly to the Azure name servers for your zone.

Why delegate your DNS to Azure? To simplify management and operations. If you create additional services, adjust the load-balancer configuration, or want to improve response times with globally replicated DNS, Azure provides that single management interface to complete those tasks. When your DNS zones are hosted in Azure, you can also implement some of the Resource Manager security features discussed in chapter 6: features such as role-based access control (RBAC) to limit and audit access to the DNS zones, and resource locks to prevent accidental, or even malicious, zone deletion.

Most domain registrars provide rather basic interfaces and controls to manage DNS zones and records. To reduce management overhead and improve security, Azure DNS allows you to use the Azure CLI, Azure PowerShell, or the REST APIs to add or edit records. Operations teams can use the same tools and workflows to onboard new services, and if problems occur, it's often easier to troubleshoot when you can verify that DNS operates as you expect without introducing the variable of a third-party DNS provider.

So if you're convinced that there's logic to delegating your domain to Azure DNS, what Azure name servers do you point your domain to? If you create an Azure DNS zone, the name servers are listed in the portal, as shown in figure 11.4. You can also access these name server addresses with the Azure CLI or Azure PowerShell.

There have been no "Try it now" exercises in the previous few pages, because unless you purchase and configure a real domain, you can't test how to route real

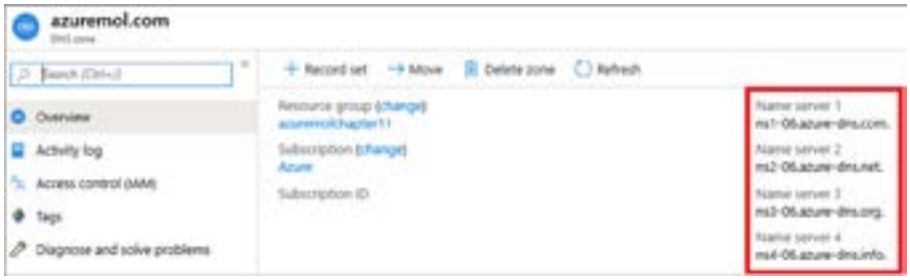


Figure 11.4 You can view the Azure name servers for your DNS zone in the Azure portal, Azure CLI, or Azure PowerShell.

traffic. You can create an Azure DNS zone without a real domain, but no traffic can route to it. In real life, you update the NS records with your current provider to point any queries for your domain to the Azure name servers. It can take 24 to 48 hours (although usually much less time) for the delegation of your domain to propagate throughout the global DNS hierarchy, so plan accordingly; this behavior may cause brief interruptions for customers who access your application.

11.3 Global routing and resolution with Traffic Manager

In previous chapters, you learned about highly available applications that are globally distributed. The goal is multiple web app or VM instances, in different regions or continents, that connect to a Cosmos DB instance close to them. But how do you get your customers to connect to the closest VM or web app that runs your application?

Azure Traffic Manager is a network service that acts as a central destination for your customers. Let's use the example of a web application at the address www.azuremol.com. Figure 11.5 provides an overview of how Traffic Manager routes users to the closest available application.

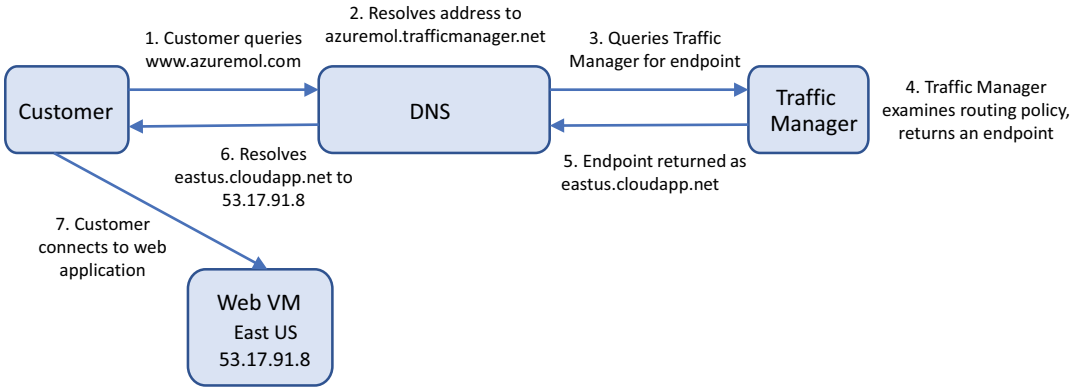


Figure 11.5 A customer sends a DNS query to a DNS service for www.azuremol.com. The DNS service forwards the query to Traffic Manager, which returns an endpoint based on the routing method in use. The endpoint is resolved to an IP address, which the customer uses to connect to the web application.

www.dbooks.org

Traffic Manager doesn't perform the role of a load balancer, which you learned about in chapter 8. As figure 11.5 shows, Traffic Manager routes traffic to a public IP. Let's examine the flow of traffic a little more closely:

1. The user makes a DNS query for www.azuremol.com. Their DNS server contacts the name servers for azuremol.com (which could be Azure name servers if you use Azure DNS!) and requests the record for www.
2. The www host resolves to a CNAME record that points to azuremol.trafficmanager.net.
3. The DNS service forwards the DNS request to the Azure name servers for trafficmanager.net.
4. Traffic Manager examines the request and determines an endpoint to direct the user toward. Endpoint health and status are examined, as with Azure load balancers. The Traffic Manager routing method is also reviewed. The routing methods that Traffic Manager can use are as follows:
   - *Priority*—Controls the order in which endpoints are accessed
   - *Weighted*—Distributes traffic across endpoints based on an assigned weight metric
   - *Performance*—Latency-based routing of users to an endpoint so that the user receives the quickest possible response time
   - *Geographic*—Associates endpoints with a geographic region, and directs users to them based on their location
5. The endpoint eastus.cloudapp.net is returned to the DNS service by Traffic Manager.
6. The DNS service looks up the DNS record for eastus.cloudapp.net and returns the result of the query to the customer.
7. With the IP address of their requested endpoint, the customer contacts the web application directly. At this point, the traffic could hit the public IP address of an Azure load balancer rather than a VM directly.

As you can see, the role of Traffic Manager is to determine a given application endpoint to direct customers to. Some health checks monitor the status of endpoints, similar to the load-balancer health probes you learned about in chapter 8. And you can define a priority or weighted traffic-routing mechanism to distribute users across a set of available endpoints—again, similar to a load balancer. Traffic Manager typically directs traffic to an Azure load balancer or application gateway, or to a web app deployment.

Azure Front Door

Traffic Manager, which we look at in this section, is great for globally distributing and routing traffic. It works with any type of internet endpoint, not just resources in Azure. The traffic routing is based on DNS and doesn't look at the actual application itself.

If you need application-level traffic distribution and the ability to do TLS/SSL offloading or per-HTTP/HTTPS request routing, Azure Front Door helps you out. Traffic Manager

(continued)

and Front Door offer the same type of service and configuration options, but Front Door is specifically designed to work at the application layer. Front Door also has some cool performance tricks, such as split TCP to break connections into smaller pieces and reduce latency.

Back in chapter 8, we looked at load balancers and mentioned Application Gateway, which works at the application layer and does things like TLS offloading. The focus in that chapter was on load balancers to help you learn the core concepts, which Application Gateway would build on. The same is true here. We focus on Traffic Manager in this chapter, though many of the same concepts and configuration options, such as routing options, are also available for Azure Front Door. As with most things in Azure, what to use in each service is driven by the applications you run and their needs.

11.3.1 Creating Traffic Manager profiles

Traffic Manager uses profiles to determine what routing method to use and what the associated endpoints are for a given request. To continue the theme of the previous chapters about a globally distributed application, you want your users to use the web application closest to them. If you look at the routing methods again, you have two ways to do this:

- *Performance routing*—The customer is routed to the endpoint with the lowest latency relative to the source of the request. This routing method provides some intelligence and always allows Traffic Manager to forward the customer to an available endpoint.
- *Geographic routing*—The customer is always routed to a given endpoint based on the source of their request. If the customer is in the United States, they're always directed to East US, for example. This routing method requires you to define geographic regions to be associated with each endpoint.

When you use geographic routing, you get a little more control of the endpoints that customers use. There may be regulatory reasons requiring customers in a given region to always use endpoints in the same region. The exercises use geographic endpoints to show a more real-world example because there's a trick to geographic routing: you should specify a *child profile*, not an endpoint directly.

The sky won't fall if you use the geographic routing method with endpoints, but the recommended practice is to use another Traffic Manager profile to pass traffic to the final endpoint. Why? Regions can be associated with only one Traffic Manager profile. In the previous chapters on high availability, you always wanted to make sure that you have redundancy. If you associate a region with a given endpoint and use geographic routing, you have no failover option should that endpoint encounter a problem or if you perform maintenance.

Instead, nested child profiles allow you to set a priority that always directs traffic to a healthy endpoint. If the endpoint is unhealthy, traffic goes to an alternative

www.dbooks.org

endpoint. Figure 11.6 shows traffic failing over to a different region, although you could also create multiple web app instances in West US and use a weighted routing method on the child profile. As you start to scale out your application environment, take time to think about how best to provide high availability to endpoints behind Traffic Manager. For these examples, you'll create failover between regions to clearly see the differences in behavior.

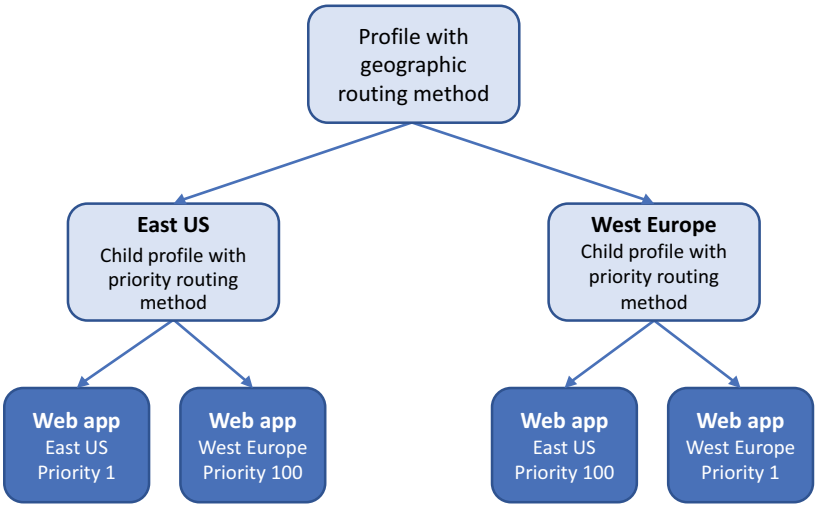


Figure 11.6 A parent Traffic Manager profile with the geographic routing method should use child profiles that contain multiple endpoints. Then those child endpoints can use priority routing to direct traffic to the preferred endpoint. The East US child profile, for example, always sends traffic to the endpoint in East US, provided that the endpoint is healthy. If the endpoint is unhealthy, traffic is directed to West Europe. Without this child profile, customers in East US couldn't fail over to an alternate endpoint and would be unable to access your web application.

Try it now

To create the Traffic Manager profiles for your distributed application, complete the following steps.

The rest of the exercises use East US and West Europe. If you don't live in one of those regions, pick a different region that's more appropriate. Just remember to be consistent throughout the exercises! The end-of-chapter lab shows how this all comes together and works, but you won't be correctly directed to your web apps if you live outside North America or Europe and don't change the regions accordingly.

1. Open the Azure portal, and select the Cloud Shell icon at the top of the dashboard.
2. Create a resource group, specifying a resource group name, such as `azuremolchapter11`, and a location, such as `eastus`:

```
az group create --name azuremolchapter11 --location eastus
```

3 Create the parent Traffic Manager profile. You want to use the geographic routing method and then specify a name, such as `azuremol`. The parameter for the DNS name tells you that it must be unique, so provide a unique name. The following domain creates the hostname azuremol.trafficmanager.net, which you use to configure the web apps in the lab at the end of the chapter:

```
az network traffic-manager profile create \
  --resource-group azuremolchapter11 \
  --name azuremol \
  --routing-method geographic \
  --unique-dns-name azuremol
```

4 Create one of the child Traffic Manager profiles. This time, use the priority routing method and the name eastus, and specify another unique DNS name, such as `azuremoleastus`:

```
az network traffic-manager profile create \
  --resource-group azuremolchapter11 \
  --name eastus \
  --routing-method priority \
  --unique-dns-name azuremoleastus
```

5 Create one more child Traffic Manager profile with the name westeurope and another unique DNS name, such as `azuremolwesteurope`:

```
az network traffic-manager profile create \
  --resource-group azuremolchapter11 \
  --name westeurope \
  --routing-method priority \
  --unique-dns-name azuremolwesteurope
```

6 You've created a web app a couple of times now, so use the CLI to quickly create two app service plans and a web app in each plan. One of these web apps is in East US; the other is in West Europe. In the end-of-chapter lab, you'll upload sample web pages to these web apps, so for now, create the empty website and get the apps ready to use a local Git repository.

Create the web app in East US as follows:

```
az appservice plan create \
  --resource-group azuremolchapter11 \
  --name appserviceeastus \
  --location eastus \
  --sku S1
az webapp create \
  --resource-group azuremolchapter11 \
  --name azuremoleastus \
  --plan appserviceeastus \
  --deployment-local-git
```

7 Create a second web app in West Europe:

```
az appservice plan create \
  --resource-group azuremolchapter11 \
  --name appservicewesteurope \
```

www.dbooks.org

```
  --location westeurope \
  --sku S1
az webapp create \
  --resource-group azuremolchapter11 \
  --name azuremolwesteurope \
  --plan appservicewesteurope \
  --deployment-local-git
```

11.3.2 Globally distributing traffic to the closest instance

You've created the Traffic Manager profiles and endpoints but no traffic that can flow. If customers were directed to the profiles, there would be no association with your endpoints. The diagram in figure 11.7 shows how you need to associate endpoint with profiles.

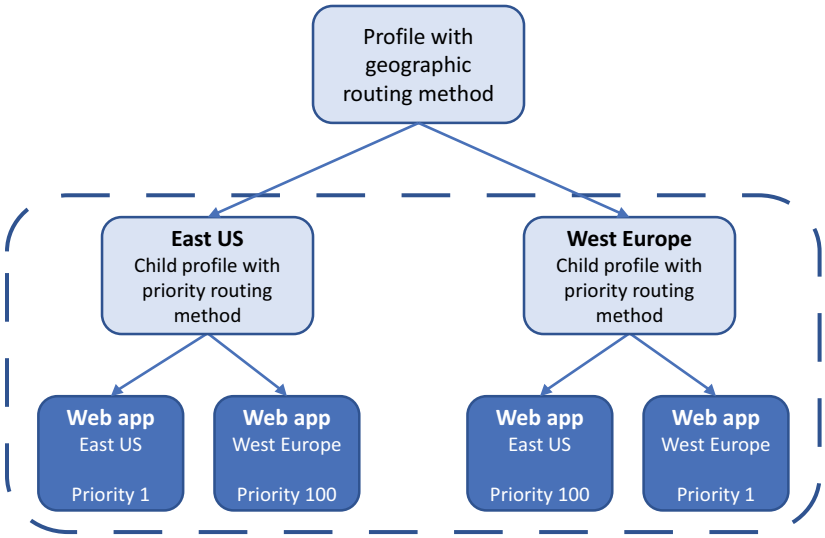


Figure 11.7 In this section, you'll associate your endpoints with the Traffic Manager profiles and define the priority for the traffic to be distributed.

The first associations you make are for your web app endpoints. Remember that for high availability, you want both web apps to be available to each Traffic Manager profile. You use a priority routing method to direct all traffic to the primary web app for each profile. If that web app is unavailable, the traffic can fail over to the secondary web app endpoint.

When you created the Traffic Manager profiles in section 11.3.1, a few defaults were used for the health-check options and endpoint monitoring. Let's explore what those options are:

- *DNS Time to Live (TTL): 30 seconds*—Defines how long the DNS responses from Traffic Manager can be cached. A short TTL ensures that customer traffic is routed appropriately when updates are made to the Traffic Manager configuration.

- *Endpoint Monitor Protocol: HTTP* —You can also choose HTTPS or a basic TCP check. As with load balancers, HTTP or HTTPS ensures that an HTTP 200 OK response is returned from each endpoint.
- *Port: 80*—The port to check on each endpoint.
- *Path: /*—By default, checks the root of the endpoint, although you could also configure a custom page, like the health-check page used by load balancers.
- *Endpoint Probing Interval: 30 seconds* —How frequently to check endpoint health. The value can be 10 seconds or 30 seconds. To perform fast probing every 10 seconds, there's an additional charge per endpoint.
- *Tolerate Number of Failures: 3*—How many times an endpoint can fail a health check before the endpoint is marked as unavailable.
- *Probe Timeout: 10 seconds* —The length of time before a probe is marked as failed and the endpoint is probed again.

You don't need to change any of these default options. For critical workloads when you build your own application environments in the real world, you could lower the number of failures to tolerate or the probing interval. These changes would ensure that any health issues were detected quickly, and traffic would be routed to a different endpoint sooner.

Try it now

To associate endpoints with profiles and finish the geographic routing, complete the following steps:

1. In the Azure portal, browse to and select your resource group. For this exercise, select the Traffic Manager profile you created for East US.
2. Choose Endpoints from the navigation bar at left in the profile, and then select Add.
3. Create an Azure endpoint, and enter a name, such as `eastus`.
4. There are different target resource types; you want to use App Service. For the target resource, select your web app in East US, such as azuremoleastus.
5. Leave Priority set to 1, accept any other defaults that may be set, and then select OK.
6. Repeat the process to add another endpoint. This time, name the endpoint `westeurope`, select your web app in West Europe as the target resource, and set a priority of 100.

   Now your Traffic Manager profile lists two endpoints: one for the web app in East US and one for the web app in West Europe, as shown in figure 11.8. This priority-based routing of the endpoints always directs traffic to the web app in East US when that resource is healthy. If that resource is unavailable, there's redundancy to fail over to the web app in West Europe.

www.dbooks.org



Figure 11.8 Two endpoints are listed for the Traffic Manager profile. The endpoint for East US has the lower priority, so it always receives traffic when the endpoint is healthy. Redundancy is provided with the West Europe endpoint, which is used only when the East US endpoint is unavailable.

7. Go back to your resource group, and select the Traffic Manager profile for West Europe.
8. Choose to add endpoints.
9. Repeat the steps to add two endpoints and configure them as follows:
   - Name: westeurope
     Target Resource: Web app in West Europe
     Priority: 1
   - Name: eastus
     Target Resource: Web app in East US
     Priority: 100

Now your Traffic Manager profile lists two endpoints: one for the web app in West Europe and one for the web app in East US, as shown in figure 11.9. You've provided the same redundancy as in the previous Traffic Manager profile, this time with all traffic going to West Europe when healthy and East US if not.



Figure 11.9 The same configuration of endpoints as the previous Traffic Manager profile, this time with the location of the web apps reversed. These child profiles can be used to route customers to the web app in either East US or West Europe, but now you have redundancy to fail over to another endpoint if the primary endpoint in the region is unavailable.

This process has just one more part, I promise! Remember, this is a best practice for high availability if you use Traffic Manager for global distribution of applications. In the real world, your environment may not be this complex. Look at the diagram again to see the child profiles and associations with the regional web apps you need to create, as shown in figure 11.10.

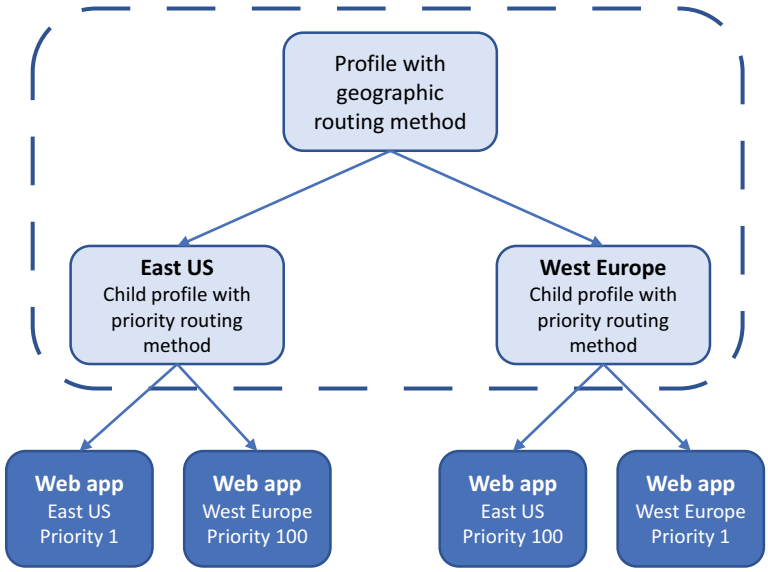


Figure 11.10 The child Traffic Manager profiles for East US and West Europe have been created, with the regional web apps and priorities configured as needed. Now you need to associate the child profiles with the parent profile.

To direct traffic based on geography, you define a region, such as North America, and a nested profile, such as eastus. All customers in the North America region are directed to this child profile. You configured the priorities on that child so that the web app in East US always serves the traffic. But you've provided a redundant option to fail over to the web app in West Europe as needed.

The inverse happens for customers in West Europe. Another endpoint for the parent Traffic Manager profile can be added, this time with Europe as the region to be associated with the endpoint, and then the `westeurope` nested profile. All European traffic is routed to this profile, and the web app in West Europe always serves the web application. In the event of a problem, the traffic can fail over to East US.

If you have policy or data sovereignty mandates such that traffic can't fail over to a different region like this, you may need to adjust how the Traffic Manager endpoints and profiles are set up. You could, for example, create multiple web apps in West

www.dbooks.org

Europe, as you saw in chapter 9. This way, you have multiple web app instances that can serve customers. Or if your application runs on VMs, use a scale set behind a load balancer to profile similar redundancy.

Try it now

This is where your own regional location matters! If you live outside one of the regional groupings shown in the Traffic Manager profiles, make sure that you select your own region; otherwise, you won't be able to access the web app in the end-of-chapter lab.

To associate the child profiles with the parent profile, complete the following steps:

1. In the Azure portal, browse to and select your resource group.
2. Select the parent Traffic Manager profile. In the earlier examples, that was called azuremol.
3. Choose Endpoints from the navigation bar at left in the profile, and then select Add.
4. Create an endpoint that uses the first child profile. Set the type as a nested endpoint, and provide a name, such as `eastus`. As the target resource, select the Traffic Manager profile you created for East US.
5. Under Regional Grouping, choose North America/Central America/Caribbean from the drop-down menu, and then select OK.
6. Repeat the steps to add another endpoint. This time, name the endpoint westeurope, set the target resource to the child Traffic Manager profile for West Europe, and choose Europe from the drop-down menu for regional grouping.

   Now your endpoints for the parent profile list the two child profiles, each having an endpoint associated with the appropriate geographic region, as shown in figure 11.11.



Figure 11.11 Nested child profiles with associated geographic regions. This parent Traffic Manager profile directs all traffic from Europe to the web app in West Europe, with redundancy to use East US if there's a problem. The opposite is true for customers in North America/Central America/Caribbean.

The web apps are currently set to accept traffic only on their default domain, which is in the form *webappname*.azurewebsites.net. When Traffic Manager directs customers to those web app instances, the traffic appears to come from the domain of the parent profile, such as azuremol.trafficmanager.net. The web apps don't recognize this domain, so the web application won't load.

7 Add the domain of the parent Traffic Manager profile to both web app instances you created in steps 4-6. If necessary, you can find the domain name on the Overview page of the parent Traffic Manager profile:

```
az webapp config hostname add \
  --resource-group azuremolchapter11 \
  --webapp-name azuremoleastus \
  --hostname azuremol.trafficmanager.net
az webapp config hostname add \
  --resource-group azuremolchapter11 \
  --webapp-name azuremolwesteurope \
  --hostname azuremol.trafficmanager.net
```

Now when you open the address of your parent Traffic Manager profile in a web browser, such as https://azuremol.trafficmanager.net, you can't tell which endpoint you access, as both web apps run the same default web page. In the end-of-chapter lab, you'll upload a basic web page to each web app to differentiate between them!

Let's stop to examine what you've created through these exercises. It's important, because now customers can use all the high-availability and redundancy features from previous chapters, with automatic traffic routing that directs them to the closest instance of your web application. In this chapter, you've created the following:

- A web app in East US and another in West Europe
- Traffic Manager profiles that use geographic routing to direct all customers in North and Central America to the East US web app, and all customers in Europe to the West Europe web app
- Child Traffic Manager policies with priority routing to provide failover use of the alternative region if the primary web app for the region is unavailable

In terms of high availability:

- If you combine this setup with web apps that autoscale, you have a ton of redundancy right now.
- If you combine these web apps with Cosmos DB, you have your entire application automatically scaling and globally distributed, with customers always accessing resources close to them for the lowest latency on response times and the best performance.
- Even if you stuck with VMs, you can use scale sets with load balancers to provide the same highly available, globally distributed environment.

And yes, you could replace Traffic Manager with Front Door if you need to use advanced application-level traffic management features.

www.dbooks.org

I know that the past few chapters contain a lot of new stuff, and each chapter has taken up pretty much all of your lunch break each day! But look at how far you've come in the past week. Now you can now create a web application with either IaaS VMs or PaaS web apps, make them highly available and load balanced, and let them scale automatically (figure 11.12). You can use a globally distributed Cosmos DB backend for your database needs, and you can automatically route customers to the closest regional instance of your application, all with DNS that's hosted in Azure.

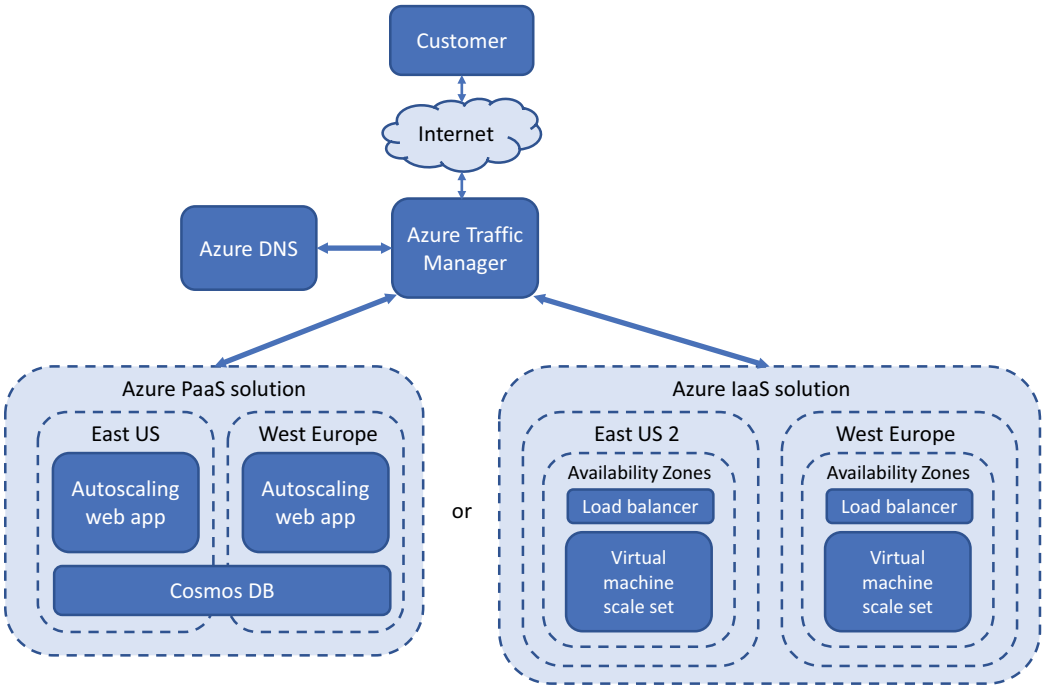


Figure 11.12 After the past few chapters, you should understand how to create highly available IaaS or PaaS applications in Azure. The IaaS solutions can use availability zones, load balancers, and scal.e sets. The PaaS solutions can use autoscaling web apps and Cosmos DB. Traffic Manager and Azure DNS can route customers to the most appropriate application instance automatically, based on their geographic location.

The end-of-chapter lab uploads a couple of basic websites to your web apps just to prove that Traffic Manager works and the appropriate endpoint serves your traffic. If you have time, feel free to complete the exercise; otherwise, pat yourself on the back, and go take a nap. I won't tell your boss!

We have one more chapter in this second section of the book, and it talks about how to make sure your applications remain healthy: how to monitor and troubleshoot your applications and infrastructure.

11.4 Lab: Deploying web apps to see Traffic Manager in action

This chapter covered a lot, so this exercise should be one that keeps building the mental muscle of your Azure skills with web apps. In the Azure samples GitHub repo are two basic web pages for the online pizza-store application. Each web page's title shows the location of the web app. Upload these web pages to the relevant web app instance to see your Traffic Manager flows in practice:

1. If necessary, clone the GitHub samples repo in your Cloud Shell as follows:

   ```
   git clone https://github.com/foouldsy/azure-mol-samples-2nd-ed.git
   ```

2. Start with the eastus web page, and then repeat the following steps in the westeurope directory:

   ```
   cd ~/azure-mol-samples-2nd-ed/11/eastus
   ```

3. Initialize the Git repo, and add the basic web page:

   ```
   git init && git add . && git commit -m "Pizza"
   ```

4. In the Azure portal, the Overview window for your web app lists the Git clone URL. Copy this URL, and then set it as a destination for your HTML sample site in Cloud Shell with the following command:

   ```
   git remote add eastus <your-git-clone-url>
   ```

5. Push the HTML sample site to your web app:

   ```
   git push eastus master
   ```

6. Repeat these steps for the azure-mol-samples-2nd-ed/11/westeurope directory.
7. When you're finished, open your web browser to the domain name of your parent Traffic Manager profile, such as https://azuremol.trafficmanager.net, to see the traffic flow.

www.dbooks.org

12 Monitoring and troubleshooting

In previous chapters, you learned how to make your applications highly available and route customers from around the world to globally distributed instances of your application. One goal was to minimize the amount of interaction with your application infrastructure and let the Azure platform manage health and performance for you. Sometimes, you still need to roll up your sleeves and review diagnostics or performance metrics. In this chapter, you'll learn how to review boot diagnostics for a VM, monitor performance metrics, and troubleshoot connectivity issues with Network Watcher.

12.1 VM boot diagnostics

With web apps, you deploy your code and let the Azure platform handle the rest. In chapter 3, we looked at the basics of troubleshooting and diagnosing problems with web app deployments. You learned how to see real-time application events to monitor performance. When you work with VMs in the cloud, it's often hard to troubleshoot a problem when you can't physically see the computer screen, the way you can get web app diagnostics.

One of the most common issues with VMs is lack of connectivity. If you can't SSH or RDP to a VM, how can you troubleshoot what's wrong? One of the first things you may want to check is whether the VM is running correctly. To help, Azure provides VM boot diagnostics that include boot logs and a screenshot of the console.

Interactive boot console access

For specific troubleshooting scenarios, you can access a live serial console for VMs in Azure. This serial console allows for interactive logons and troubleshooting in the event of boot problems. You can reconfigure your VM to correct for failed boot scenarios or misconfigurations of services and applications that prevent your VM from booting correctly.

This chapter doesn't go into specific scenarios for serial console use, but it's a great resource that lets you virtually sit in front of the screen of a VM as it starts up. You also need boot diagnostics enabled, so these exercises are prerequisites for the serial console.

Try it now

To create a VM and enable boot diagnostics, complete the following steps:

1. In the Azure portal, select Create a Resource in the top-left corner.
2. Search for and select a Windows Server 2019 Datacenter VM image.
3. Create a resource group, such as `azuremolchapter12`, and then select the most appropriate Azure region closest to you.
4. Select a VM size, such as DS1_v2.
5. Enter a username for the VM, such as `azuremol`, and a password. The password must be a minimum 12 characters long and contain 3 of the following: a lowercase character, an uppercase character, a number, and a special character.
6. Accept any options for redundancy or inbound port rules.
7. Accept the defaults for disks and networking; there's nothing you need to change. Those settings should be familiar to you by now.

   One thing you may have skipped previously was the Management section. As shown in figure 12.1, the boot diagnostics option is enabled by default, and a storage account is created.
8. For now, leave the OS guest diagnostics option disabled.
9. Review your VM configuration settings, and select Create.

It takes a few minutes to create and configure the VM, so let's continue to explore the boot diagnostics.

If you don't have boot diagnostics enabled but run into a problem, you likely can't boot the VM to enable diagnostics. It's a fun chicken-and-egg scenario, right? As a result, boot diagnostics are automatically enabled for VMs created in the Azure portal. For Azure PowerShell, Azure CLI, and the language-specific SDKs, you need to enable

www.dbooks.org

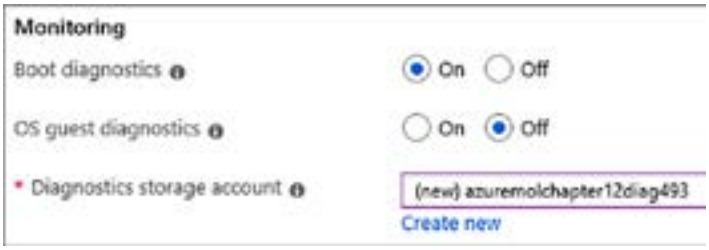


Figure 12.1 By default, boot diagnostics are enabled when you create a VM in the Azure portal. A storage account is created; the boot diagnostics are stored in this account. In a later exercise, you'll review and enable OS guest diagnostics, so don't enable them right now. For production use, I recommend that you enable both boot diagnostics and OS guest diagnostics for each VM you create.

boot diagnostics. I highly recommend that you enable boot diagnostics on your VMs when you create them. Make a habit of using Azure Resource Manager templates (chapter 6) or your own Azure CLI or PowerShell scripts that enable boot diagnostics during deployment.

You do need to create a storage account for the boot logs and console screenshots, but the cost to store this data is likely less than $0.01 per month unless you have a busy VM that generates a lot of data. The first time you run into a VM problem and need access to the boot diagnostics, that penny per month will be worth it! This storage account can also be used to hold additional VM-level performance metrics and logs, which we'll examine in section 12.2. Again, the storage costs should be minimal. Even as your VM environment grows, it's worth the additional minor cost to be able to quickly troubleshoot an issue when things go wrong.

Try it now

To view the boot diagnostics for your VM, complete the following steps:

1. In the Azure portal, select Virtual Machines from the menu on the left.
2. Choose the VM you created in the previous exercise.
3. In the Support + troubleshooting section of the VM menu, choose Boot diagnostics. The boot diagnostics and VM status are displayed, as shown in figure 12.2. The health report indicates any boot problems with the VM and allows you to diagnose the root cause of the issue.

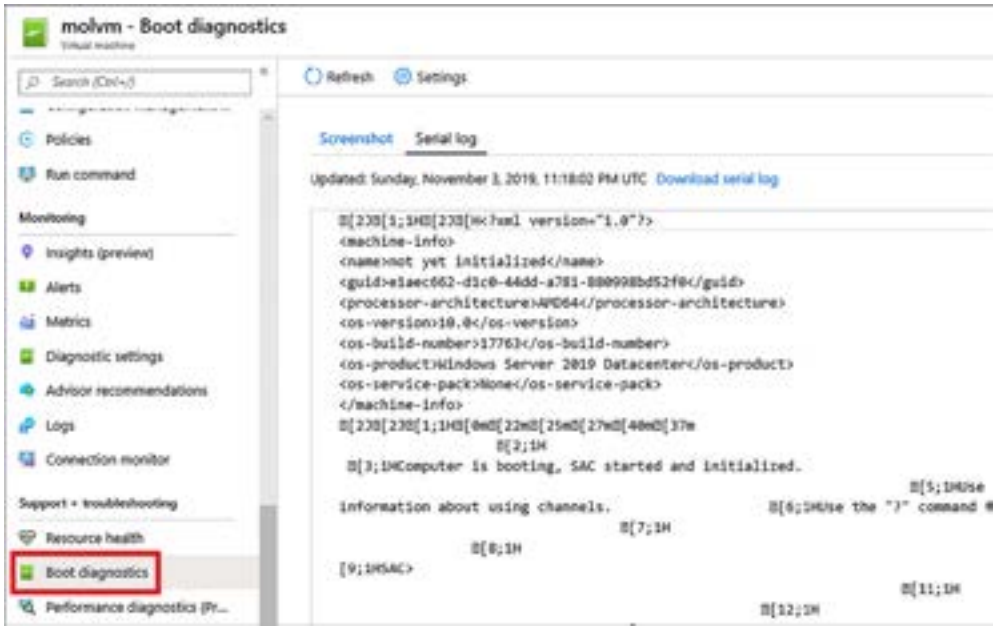


Figure 12.2 The boot diagnostics for a VM report on the health and boot status. If errors are displayed, you should be able to troubleshoot and diagnose the root cause. You can also download the logs from the portal for analysis on your local computer.

12.2 Performance metrics and alerts

One of the first steps in troubleshooting an issue is a performance review. How much memory is available, how much CPU is consumed, and how much disk activity is there?

As you build and test your applications in Azure, I recommend that you record performance baselines at various points. These baselines give you an idea of how your application should perform under different amounts of load. Why is this important? In three months, how can you determine whether you encounter performance problems without some data to compare the current performance with?

When you learned how to autoscale applications in chapter 9, you used basic performance metrics, such as CPU use, to tell the Azure platform when to increase or decrease the number of instances of your application. These basic metrics give you only small insights into how the VM performs, however. For more detailed metrics, you need to look at the performance of the VM, and to do this, you need to install the Azure diagnostics extension.

12.2.1 Viewing performance metrics with the VM diagnostics extension

To add functionality to your VMs, Azure has dozens of extensions that you can install seamlessly. These extensions install a small agent or application runtime into the VM, which often reports information back to the Azure platform or third-party solutions.

www.dbooks.org

VM extensions can automatically configure and install components or run scripts on your VMs.

The VM diagnostics extension is a common one that's used to stream performance metrics from inside the VM to a storage account. These performance metrics can be analyzed in the Azure portal or downloaded and used in an existing monitoring solution. You can use the diagnostics extension to gain deeper understanding of the performance of CPU and memory consumption from within the VM, which can typically provide a more detailed and accurate picture than the host.

Automation and VM extensions

In chapter 18, we'll discuss Azure Automation, which allows you to perform tasks on your VMs in an automated, scheduled manner. One powerful feature of Azure Automation is acting as a PowerShell Desired State Configuration (DSC) pull server. PowerShell DSC defines a given state of how a system should be configured, what packages should be installed, files and permissions, and so on. You create definitions for the desired configuration and apply them to VMs or physical servers. Then you can report on and enforce compliance with those policies. The Azure PowerShell DSC extension is used to apply DSC configurations, such as from an Azure Automation pull server.

Other extensions that can apply configurations and run scripts on VMs include the Azure Custom Script Extension. With the Custom Script Extension, you define a simple set of commands or point to one or more external scripts, such as those hosted in Azure Storage or GitHub. These scripts can run complex configuration and installation tasks, and ensure that all deployed VMs are configured consistently.

Both the Azure PowerShell DSC extension and Custom Script Extension are commonly used with virtual machine scale sets. You apply one of these extensions to the scale set, and as VM instances are created within the scale set, they're automatically configured to run your application. The goal of these extensions is to minimize the required manual configuration of VMs, which is an error-prone process that requires human interaction.

Other ways to automate VM configurations include Puppet and Chef, both of which have Azure VM extensions available. If you already have a configuration-management tool in use, check with the vendor for its supported approach for use in Azure. There's a good chance that a VM extension is available to make your life easier.

Try it now

To enable the VM diagnostics extension, complete the following steps:

1. In the Azure portal, choose Virtual Machines from the menu on the left.
2. Choose the VM you created in a previous exercise.
3. In the Monitoring section of the VM menu, choose Diagnostic settings.
4. Select the button to enable guest-level monitoring.

It takes a couple of minutes to enable guest-level monitoring. Behind the scenes, here's what Azure does:

- Installs the VM diagnostics extension
- Configures the extension to stream guest-level metrics for the following areas:
  - Logical disk
  - Memory
  - Network interface
  - Process
  - Processor
  - System
  - Enables application, security, and system logs to be streamed to Azure Storage

When the diagnostics extension is installed, you can limit what data is collected by selecting only certain performance counters to report. You may wish to collect only memory use, for example, or enable the collection of Microsoft SQL Server metrics. By default, metrics are collected every 60 seconds. You can adjust this sample rate as desired for your applications and infrastructure.

The VM diagnostics extension can also stream log files from your VM, allowing you to centralize the application, security, and system logs for analysis or alerts, as shown in figure 12.3. By default, application and system logs that generate Critical, Error, or Warning alerts are logged, along with security events for Audit Failure. You can change the log levels to enable log collection from IIS and application logs, and for Event Tracing for Windows (ETW) events. As part of your application planning and deployment, determine what logs you want to collect.

There's nothing unique to Windows VMs here. You can use the diagnostics extension on Linux VMs in the same way to obtain performance metrics and stream various logs.

If your VM encounters a problem, often the only way to analyze what happened is to review the *crash dumps*. Support channels often request these dumps if you want to get to the root cause of a problem. As with boot diagnostics, there's no way to retroactively enable crash dumps to see why something failed, so determine whether you need to monitor certain processes, and be proactive about configuring crash dumps. You could monitor the IIS process and record a full crash dump to Azure Storage if the process fails, for example.

Here are a couple of other areas that you can configure for guest metrics:

- *Sinks* allow you to configure the VM diagnostics extension to send certain events to Azure Application Insights. With Application Insights, you can gain visibility directly into how your code performs.
- *Agent* lets you specify a storage quota for all your metrics. (The default is 5 GB.) You can also enable the collection of logs for the agent itself or uninstall the agent.

www.dbooks.org

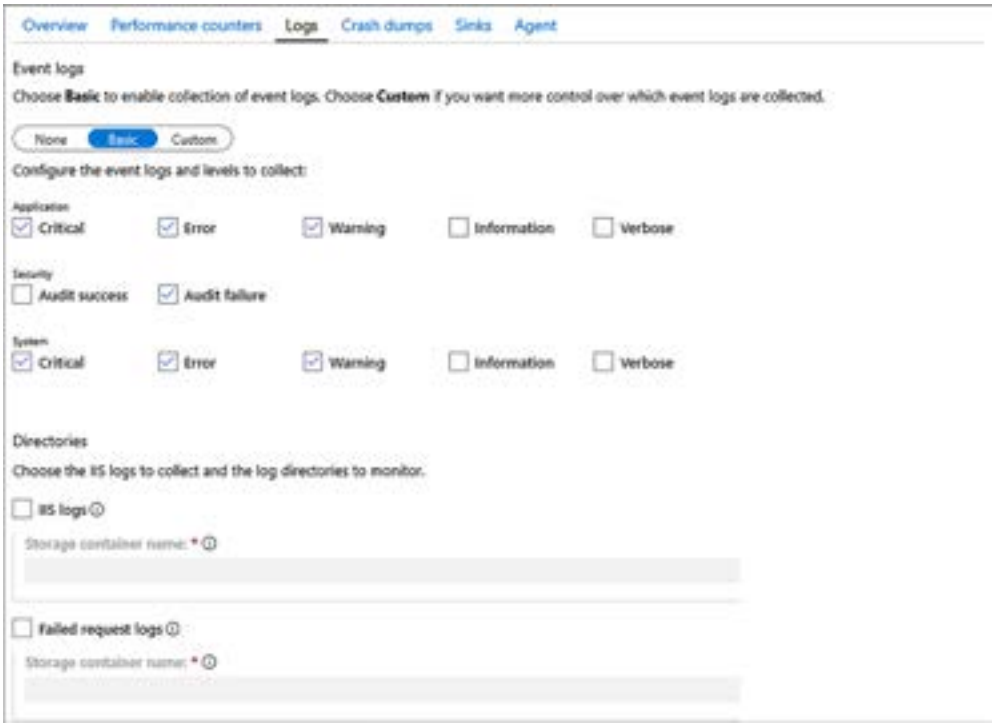


Figure 12.3 You can configure events and log levels for various components within the VM. This feature lets you centralize your VM logs for analysis and to generate alerts. Without the need to install complex, often costly monitoring systems, you can review and receive notifications when issues arise on your Azure VMs.

Try it now

To view guest-level metrics, complete the following steps:

1. In the Azure portal, choose Virtual Machines from the menu on the left.
2. Choose the VM you created in a previous exercise.
3. In the Monitoring section of the VM menu, choose Metrics.

Many more metrics are now available, compared with the basic host-based metrics from chapter 9. Explore some of the VM host and guest metrics available, and think about some applications you may want to monitor specific metrics for.

12.2.2 Creating alerts for performance conditions

With your VM configured to expose guest-level performance metrics, how do you know when there's a problem? You don't want to sit and watch the performance graphics in real time and wait until a problem occurs! I'm not your boss, if that's your thing. But there's a much better way: metric alerts.

Metric alerts let you select a resource, metric, and threshold, and then define who and how you want to notify when that threshold is met. Alerts work on more than just VMs. You can define alerts on public IP addresses that watch for inbound distributed denial of service (DDoS) packets, for example, and warn you when a certain threshold is met that could constitute an attack.

When alerts are generated, you can choose to send an email notification to owners, contributors, and readers. These users and email addresses are obtained based on the RBAC policies applied. In larger organizations, alerts could send email notifications to a large group of people, so use with care! Another option is to specify email addresses, which could be the application owners or specific infrastructure engineers, or a distribution list or group targeted to the directly involved parties.

A couple of other useful options exist for actions to take when an alert is triggered:

- *Execute a runbook*. In chapter 18, we'll examine Azure Automation. The Automation service allows you to create and use runbooks that execute scripts. These scripts could perform a basic remedial action on the VM, such as to restart a process or even reboot the VM. They could also run Azure PowerShell cmdlets to enable Azure Network Watcher features like capture packets, which we'll explore in the rest of this chapter.
- *Run a logic app*. Azure logic apps allow you to build workflows that run serverless code. You could write information to a support-ticket system or initiate an automated phone call to an on-call engineer. In chapter 21, we'll explore the wonderful world of serverless computing with Azure logic apps and Azure functions.

In the end-of-chapter lab, you'll configure some alerts for your VM. Azure can do more than help to troubleshoot and monitor your VMs, though. Let's discuss another common cause for things to go wrong: the network.

12.3 Azure Network Watcher

VM performance metrics and boot diagnostics are great ways to monitor your Azure IaaS applications. Web app application logs and App Insights provide awareness of the performance of your PaaS applications. Network traffic is often less glamorous, but it's more likely to be the cause of application connectivity issues that you or your customers encounter.

Back in chapter 5, I joked that the network team always gets the blame for problems that the operations team can't explain. Here's where we can try to make friends again, or at least get some solid proof of the network being to blame! Azure Network Watcher is one of those features that helps bring teams together for a nice group hug. With Network Watcher, you can monitor and troubleshoot using features such as these:

- Capturing network packets
- Validating IP flow for NSGs
- Generating network topology

What's great about these features is that they put different teams in the driver's seat to troubleshoot problems. If you create some VMs and then can't connect to them, you

www.dbooks.org

can verify that there's network connectivity. For developers, if your application can't connect to a backend database tier, you can examine the NSG rules to see whether there's a problem. And network engineers can capture packets to examine the complete communication stream among hosts for more in-depth analysis.

Additional network troubleshooting

Network Watcher works in tandem with the diagnostic logs and metrics discussed earlier in the chapter. Network resources such as load balancers and application gateways can also generate diagnostic logs. These logs work the same way as application and system logs from a VM or web app. Logs are collated in the Azure portal for you to determine whether there are errors in the configuration or communications between hosts and applications.

DNS and Traffic Manager also have a Troubleshoot area in the Azure portal. The portal guides you through some common errors that you may encounter, offers configuration advice, and provides links to additional documentation. If all else fails, you can open a support request with Azure Support.

Although it may be easier to build large application deployments with Azure Resource Manager templates or with Azure CLI or PowerShell scripts, the Azure portal has a lot of great tools and features you can use when things go wrong. Especially with complicated network configurations and security policies, a few seconds of your time to review the output from Network Watcher tools can identify an issue and let you resolve it quickly. All these tools help improve the overall health and experience of your applications for your customers.

What are some scenarios in which you may want to use Network Watcher and the troubleshooting features it offers? Let's look at a few common issues and see how Network Watcher could help.

12.3.1 Verifying IP flows

Here's a common problem: customers can't connect to your application. The application works fine when you connect from the office, but customers can't access the application over the public internet. Why?

VPNs and ExpressRoute

Azure virtual private networks (VPNs) provide secure communications between on-premises offices and Azure data centers. Azure ExpressRoute provides high-speed, dedicated private connections from on-premises offices to the Azure data centers and is often used in large organizations.

Both connections are a little more complicated to set up than we can cover in a single lunch break, and they're also often things that you set up and configure only once. The network team is usually responsible for configuring them, and you may not even realize that you access Azure over a private connection.

All the testing of your application works great. You can access the application through a web browser, place orders, and receive email notifications. But when your customers try to place an order, the application doesn't load.

How can Network Watcher help? By verifying IP flows. Network Watcher simulates traffic flow to your destination and reports whether the traffic can reach your VM.

Try it now

To enable Network Watcher and verify IP flows, complete the following steps:

1. In the Azure portal, choose All Resources from the top of the navigation menu on the left.
2. Filter and select Network Watcher from the list of available services. You enable Network Watcher in the region(s) that you wish to monitor. When you enable Network Watcher in a region, Azure uses role-based access controls for the various resources and network traffic.
3. Expand the list of regions for your account. Some regions may already be enabled. If the region your VM was deployed into isn't enabled, select the region, and then enable the network watcher.
4. When Network Watcher is enabled in a region (it takes a minute or two), select IP Flow Verify under Network Diagnostic Tools on the left side of the Network Watcher window.
5. Select your resource group, such as azuremolchapter12, and VM, such as molvm. By default, Protocol is set to TCP, and Direction is Inbound. The Local IP Address of the virtual NIC is also populated.
6. For Local Port, enter port 80. If you accepted the defaults when you created the VM in the previous exercise, you didn't open port 80, so this is a good test of what happens when traffic is denied.
7. Under Remote IP Address, enter `8.8.8.8`. This address may seem familiar: it's an open DNS server provided by Google. You aren't doing anything with this server; you just need to give Network Watcher an external IP address to simulate traffic flow. You could also go to https://whatsmyip.com and enter your real public IP address.
8. Set Remote Port to port 80, and then select Check.

The result of your IP flow check should be `Access denied`. Helpfully, Network Watcher tells you which rule caused the traffic flow to fail: the DenyAllInBound rule. You know that there's a network security rule that blocks traffic, but where is this rule applied? At the subnet, virtual NIC, or application security group? Another Network Watcher feature can tell you!

12.3.2 Viewing effective NSG rules

NSG rules can be applied to a single virtual NIC, at the subnet level, or against a group of VMs in an application security group. Rules are combined, which allows you to specify a common set of rules across an entire subnet and then get more granular for

www.dbooks.org

application security groups (such as "Allow TCP port 80 on all webservers") or an individual VM.

Here are some common examples of how NSG rules may be applied:

- *Subnet level*—Allow TCP port 5986 for secure remote management from management subnet 10.1.10.20/24.
- *Application security group level*—Allow TCP port 80 for HTTP traffic to web applications, and apply the application security group to all web application VMs.
- *Virtual NIC level*—Allow TCP port 3389 for remote desktop access from management subnet 10.1.10.20/24.

These rules are basic, and they explicitly allow certain traffic. If no *allow* rules match a network packet, the default *DenyAll* rules are applied to drop the traffic.

During the testing of the application discussed in the example, you may have configured that HTTP rule to allow only traffic from one of your on-premises subnets. Now customers over the public internet can't connect.

Try it now

To determine where an NSG rule is applied, complete the following steps:

1. In Network Watcher, select Effective security rules on the left.
2. Select your resource group, such as azuremolchapter12, and your VM, such as molvm. It takes a few seconds for the effective rules to be displayed as shown in figure 12.4.

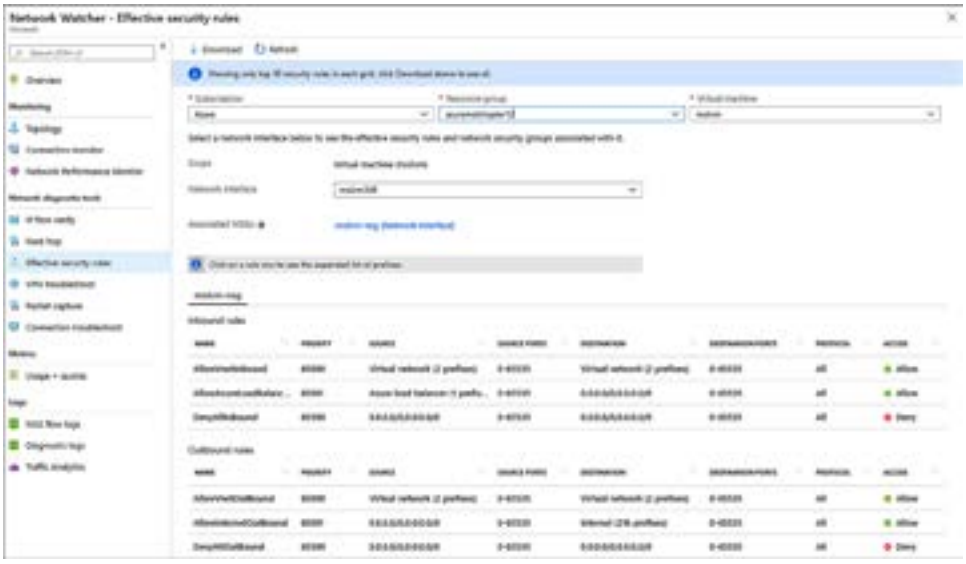


Figure 12.4 When you select a VM, Network Watcher examines how all the NSG rules are applied and the order of precedence, and shows what effective rules are currently applied. Then you can quickly drill down to the subnet, virtual NIC, and default rules to find and edit where a given rule is applied.

The default rules from the VM you created earlier aren't exciting, but you can move through subnet, network interface, and default rules to get a feel for the way that effective rules are combined and how you could identify where rules are applied if you need to make changes.

12.3.3 Capturing network packets

Let's assume that you updated your network security rules to allow access to your application for public internet customers, but one customer reports experiencing odd behavior. The web application sometimes doesn't load or displays broken images. Their connection often appears to time out.

Intermittent problems are often the hardest to troubleshoot, especially if you have limited, or no, access to the computer that encounters a problem. One common troubleshooting approach is to capture the network packets and review them for signs of problems such as network transmission errors, malformed packets, or protocol and communication issues.

With network packet captures, you get the raw stream of data between two or more hosts. There's an art to analyzing network captures, and it's not for the fainthearted! Special third-party tools such as Riverbed's Wireshark, Telerik's Fiddler, and Microsoft's

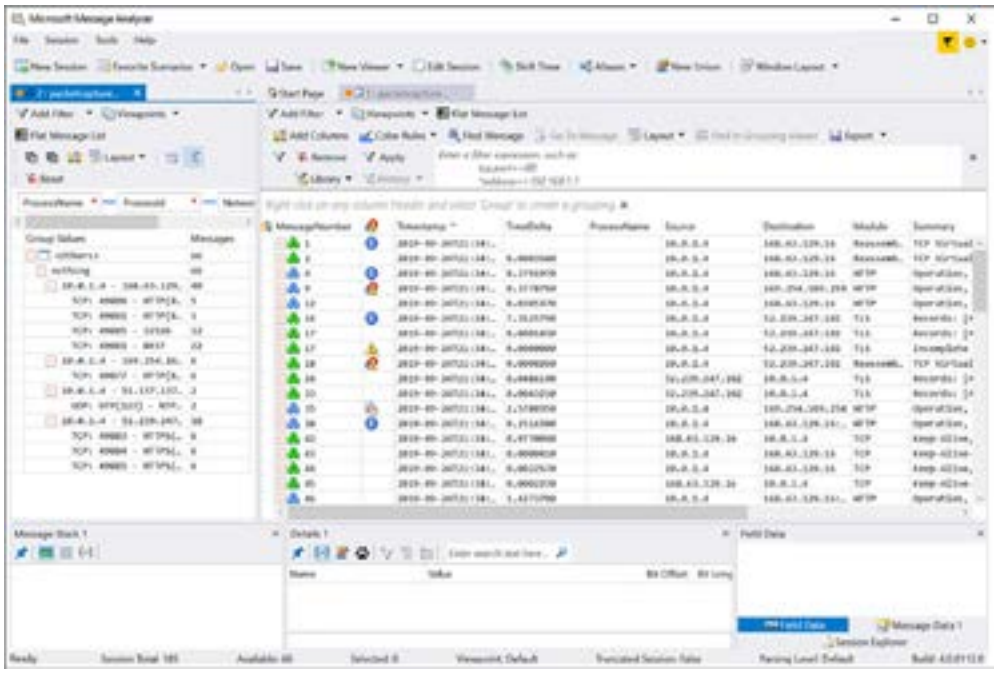


Figure 12.5 A network packet capture when viewed in Microsoft's Message Analyzer. Each individual packet is available for inspection. You can group and filter by communication protocol or client-host. This depth of network data allows you to examine the actual packets that flow between nodes to troubleshoot where an error occurs. A former colleague once told me, "The packets never lie." The puzzle is to figure out what the packets tell you.

www.dbooks.org

Message Analyzer provide a graphical way for you to view and filter the network packets, typically grouping them by related communications or protocols. Figure 12.5 shows an example network packet capture.

To enable Network Watcher to capture packets to and from your VMs, first install the Network Watcher VM extension. As you saw in section 12.3.2, VM extensions provide a way for the Azure platform to reach inside a VM to perform various management tasks. In the case of the Network Watcher extension, it examines network traffic to and from the VM.

Try it now

To install the Network Watcher VM extension and capture network packets, complete the following steps:

1. In the Azure portal, choose Virtual Machines from the menu on the left, and then select your VM, such as molvm.
2. In the Settings category on the left in the VM window, select Extensions.
3. Choose Add an Extension.
4. In the list of available extensions, choose Network Watcher Agent for Windows, and then select Create.
5. To confirm the extension install, select OK. It may take a few minutes for Network Watcher Agent to be installed on your VM.
6. To go back to the Network Watcher menu in the Azure portal, choose All Resources at the top of the Services navigation menu on the left in the portal, and then choose Network Watcher.
7. In the Network Diagnostic Tools section on the left in the Network Watcher window, select Packet Capture, and then choose Add a New Capture.
8. Select your resource group, such as azuremolchapter12, and VM, such as molvm; then enter a name for your packet capture, such as `molcapture`.

   By default, packet captures are saved to Azure Storage. You can also choose to save to file and specify a local directory on the source VM. The Network Watcher Agent extension writes the packet capture file to disk in the VM.
9. If it isn't already selected, choose the storage account name that starts with the name of your resource group, such as azuremolchapter12diag493. This account is the storage account created and used by the VM diagnostics extension that you enabled earlier.
10. You can specify a maximum size for each packet (default is 0 for the entire packet), the maximum file size for the packet capture session (default is 1 GB), and the time limit for the packet capture (default is 5 hours). To capture traffic only from specific sources or ports, you can also add a filter to narrow the scope of your packet captures.

11 Set a time limit of 60 seconds.

12 To start the packet capture, select OK.

It takes a minute or two to start the capture. When the capture is in progress, the data is streamed to the Azure Storage account or local file on the VM. The list of captures is shown on the Network Watcher portal page. If you stream the logs to Azure Storage, you can have the capture go straight to the Storage account and download the .cap capture file. Then you can open the packet capture in an analysis program, as discussed in section 12.3.3. In fact, the example network capture shown in figure 12.5 earlier in this chapter is from an Azure Network Watcher packet capture!

12.4 Lab: Creating performance alerts

I hope that the VM diagnostics, metrics, and Network Watcher features covered in this chapter have given you some insight into what's available in Azure to troubleshoot application problems. Some things, such as boot diagnostics and the VM diagnostics extension, make the most sense when you enable and configure them as you deploy VMs.

In this lab, you'll configure some metric alerts to see what you can be notified about and what the alerts look like when you receive them:

1 In the Azure portal, browse to the VM you created in the previous exercises.

2 In the Monitoring section for the VM, select Alerts.

3 Choose to create an alert rule, and then add a condition for when the CPU percentage is greater than an average of 10% in the past 5 minutes. A chart should show you what the latest metrics are, so adjust the threshold if 10% wouldn't trigger an alert.

4 Add an action group, and give it a name and short name. For this lab, set both names to azuremol. Action groups let you define reusable sets of steps to perform when an alert is generated, such as email a set of users, or run an automated PowerShell script or Azure Logic App.

5 Explore the available action types, and then select Email/SMS/Push/Voice.

6 Choose how you want to be notified, such as via email or text message. Some carrier charges may apply for SMS or voice notifications.

7 When the action group has been created, give the alert a name, and then specify a severity. This severity is useful when you have lots of alerts defined to help you triage and prioritize what to resolve first.

8 Create the rule when you're ready. It takes 10 to 15 minutes for the rule to become active and generate the notifications you defined.

This example is a basic one, so think of any existing alerts and notifications you have for applications and services and how you could use this feature when you run workloads in Azure.

www.dbooks.org

Part 3

Secure by default

In an online world in which applications are typically connected to the internet 24/7, the threat of a digital attack is all too real. These attacks cost time, money, and customer trust. A central part of building highly redundant and distributed applications includes securing them and protecting your data. Azure has several built-in features to secure your data, including encryption, monitoring, digital key vault, and backups. In this part of the book, you learn how to secure and protect your applications right from the start.

www.dbooks.org

13 Backup, recovery, and replication

The next few chapters introduce some of the core Azure features and services that allow you to build security into your applications. That's probably too subjective: security shouldn't be an add-on feature or consideration. Rather, security should be inherently built into the heart and soul of your application from the start. In this chapter, you'll begin your journey into Azure security with backing up and recovering your data. Backups may not seem like a common security topic, but think about security as being more than data encryption or website SSL certificates. What about the protection of your data from outages, data loss, and hacking? A discussion of backups and replication also acts as a good topic to bridge between the chapter on high availability and this chapter.

Backups may seem trivial, and as a former backup administrator, I can tell you there isn't much exciting about backup jobs and rotations! But timely backups that work are crucial to protect your applications and ensure that in the worst-case scenario, you can restore your data quickly and reliably. You can also replicate your VMs from one Azure region to another. This ability builds on the high-availability concepts we looked at back in chapter 7.

In this chapter, you'll learn how to back up and restore VMs and then replicate VMs automatically across Azure. All these backups and restore points are encrypted to secure your data.

13.1 Azure Backup

One of the cool things about Azure Backup is that it's both a service and a big bucket of storage for the actual backups. Azure Backup can protect VMs in Azure, on-premises VMs or physical servers, and even VMs in other providers such as Amazon

Web Services (AWS). The data backups can be stored on your own on-premises storage arrays or within an Azure recovery vault. Figure 13.1 shows how the Azure Backup service can protect and orchestrate all of your backup needs.

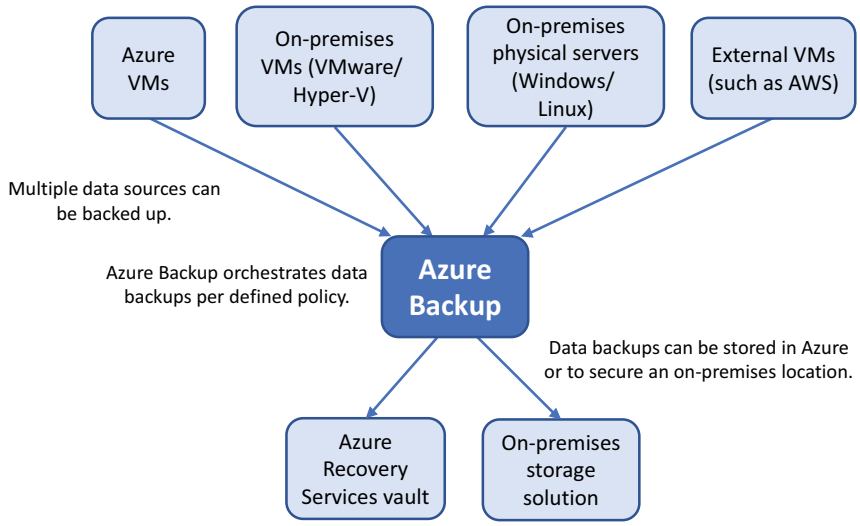


Figure 13.1 Multiple VMs or physical servers, from various providers and locations, can be backed up through the central orchestration service. Azure Backup uses defined policies to back up data at a given frequency or schedule. Then these backups can be stored in Azure or an on-premises storage solution. Throughout, data is encrypted for added security.

At its core, Azure Backup manages backup schedules and data retention, and orchestrates the backup or restore jobs. To back up Azure VMs, you have no server component to install and no agent to install manually. All the backup and restore operations are built into the Azure platform.

To back up on-premises VMs or physical servers, or VMs in other providers such as AWS, you install a small agent that enables secure communication back and forth with Azure. This secure communication ensures that your data is encrypted during transfer.

For data stored in Azure, the backups are encrypted with an encryption key that you create. Only you have access to those encrypted backups. You can also back up encrypted VMs (which we'll look at in chapter 14) to really make sure your data backups are safe.

There's no charge for the network traffic flow to back up or restore data. You pay only for each protected instance and then for however much storage you consume in Azure. If you use an on-premises storage location, the cost to use Azure Backup is minimal because there are no Azure Storage or network traffic costs.

www.dbooks.org

13.1.1 Policies and retention

Azure Backup uses an incremental backup model. When you protect an instance, the first backup operation performs a full backup of the data. After that, each backup operation performs an incremental backup of the data. Each of these backups is called a *recovery point*. Incremental backups are a time-efficient approach that optimizes the storage and network bandwidth usage. Only data that has changed since the previous backup is securely transferred to the destination backup location. Figure 13.2 details how incremental backups work.

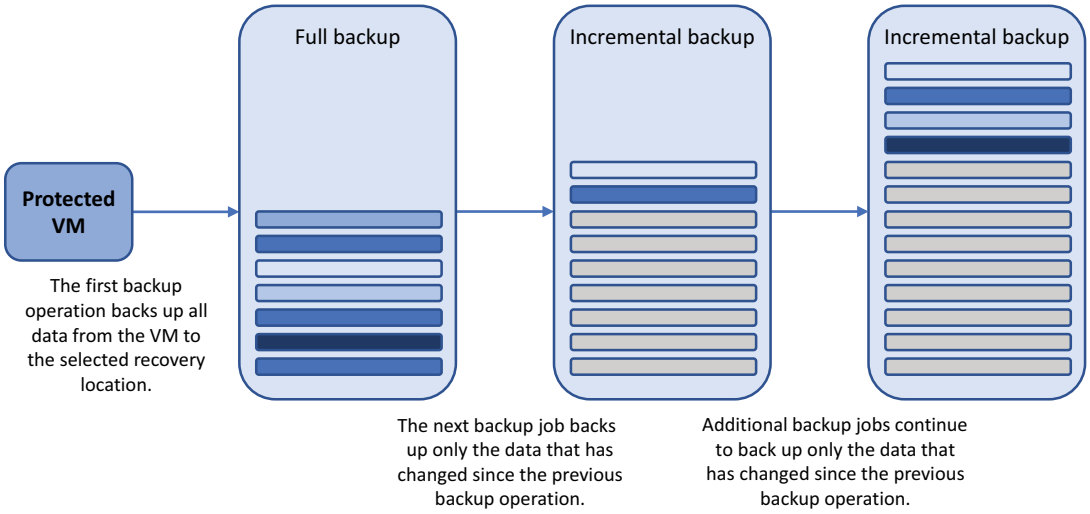


Figure 13.2 Incremental backups back up only the data that has changed since the previous operation. The first backup is always a full backup. Each subsequent backup job backs up only data that has changed since the previous job. You control the frequency of full backups with policies. This approach minimizes the amount of data that needs to travel securely across the network and be housed in the destination storage location. Azure Backup maintains the relationships of incremental backups to ensure that when you restore data, that data is consistent and complete.

With Azure Backup, you can store up to 9,999 recovery points for each instance that you protect. For some context, if you made a regular daily backup, you'd be set for more than 27 years. And you could keep weekly backups for almost 200 years. I think that would cover most audit situations! You can choose to retain backups on a daily, weekly, monthly, or yearly basis, which is in line with most existing backup policies.

To implement the optimal backup strategy for your workload, you need to understand and determine your acceptable *recovery point objective* (RPO) and *recovery time objective* (RTO).

RECOVERY POINT OBJECTIVE

The RPO defines the point that your latest backup allows you to restore. By default, Azure Backup makes a daily backup. Then you define retention policies as to how many days, weeks, months, or years you wish to keep these recovery points. Although the RPO is typically used to define the maximum amount of acceptable data loss, you should also consider how far back in time you may wish to go. Figure 13.3 shows how the RPO defines the amount of acceptable data loss.

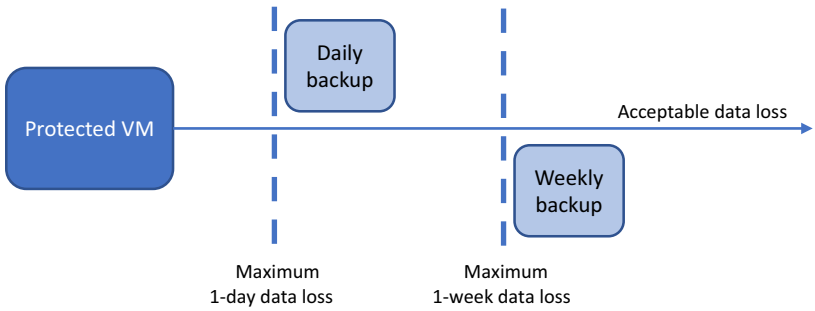


Figure 13.3 The RPO defines how much data loss you can sustain for a protected instance. The longer the RPO, the greater the acceptable data loss. An RPO of 1 day means that up to 24 hours of data could be lost, depending on when the data loss occurred in relation to the last backup. An RPO of one week means up to seven days' worth of data could be lost.

Major outages and large amounts of data loss are rare occurrences. More common are incidents of small data loss or overwrites. These incidents often aren't noticed or reported until sometime after the data loss occurred. This is where the retention policy for your protected instances becomes important. If you have a short retention policy, you may be unable to restore data from the required point in time. You need to determine a balance between retaining multiple recovery points and the storage costs to retain all those recovery points.

Azure storage is relatively cheap, typically less than $0.02 per gigabyte of storage. This cost equates to approximately $2 per month for a 100 GB VM data backup (plus a charge for the actual Azure Backup service). Depending on how much your data changes, the size of the incremental recovery points could add up quickly. Retaining recovery points for weeks or months could cost tens of dollars per month per protected instance. This isn't to discourage you, but it's important to plan your needs and be smart about your costs. Storage looks cheap at less than $0.02 per gigabyte until you have hundreds of gigabytes per protected instance and dozens or even hundreds of instances to protect.

I'm a former backup administrator, and storage capacity was often a central factor when I determined how many recovery points to retain. That storage capacity often

www.dbooks.org

created compromises with those RPOs. If you use Azure Storage rather than an on-premises storage solution, you don't need to worry about available storage capacity. I can all but guarantee that there's more storage than your credit card limit!

RECOVERY TIME OBJECTIVE

The RTO dictates how quickly you can restore your data. If you choose to back up Azure VMs and store the recovery points in an on-premises storage solution, it takes much longer to restore those backups than if they were housed directly in Azure Storage. The inverse would be true if you backed up on-premises VMs or physical servers to Azure Storage. Figure 13.4 outlines the RTO.

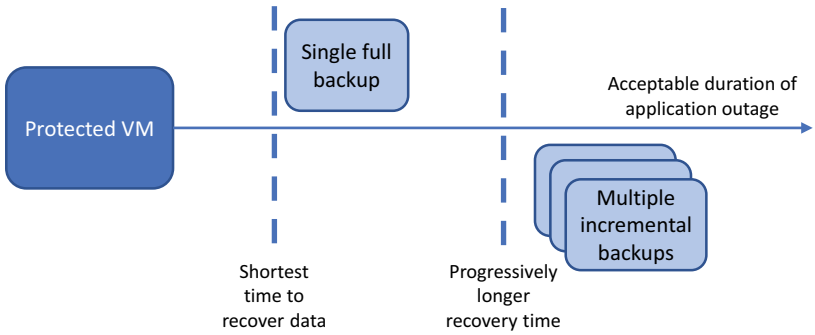


Figure 13.4 The RTO defines how long it's acceptable for the data-restore process to take and the application to be unavailable. The more recovery points are involved in the restore process, the longer the RTO. In a similar manner, the closer the backup storage is to the restore point, the shorter the RTO.

In either scenario, the recovery-point data would need to be transferred from the recovery-point storage location to the restore location. For large restore operations, in which you may need to transfer hundreds of gigabytes, your network bandwidth becomes a real bottleneck that controls how quickly you can make applications available again.

The same is true for long retention policies with many successive incremental recovery points. Restoring the data may require that multiple recovery points be mounted and restored. Your job is to determine how far back in time you need to be able to travel and how much time you can take to restore the data.

Both RPO and RTO vary based on your applications and business use. An application that processes real-time orders can't tolerate much outage or downtime, so the RPO and RTO are likely to be very low. You typically use a database to hold your data, so you'd typically design tolerances into the application rather than rely on recovery points. If you think back to Cosmos DB, there isn't anything to back up: the Azure platform performs the replication and data protection for you. If you built a custom solution on MySQL or Microsoft SQL Server, you'd typically use a similar type of clustering

and replication to ensure that multiple copies of the database exist, so the loss of one instance wouldn't require you to restore from a backup. Backups primarily protect against a major outage or data corruption.

13.1.2 Backup schedules

How do you control the frequency of your backups and the retention of the recovery points? In Azure Backup, these settings are defined in policies. You build these policies to cover the various scenarios you wish to protect against, and you can reuse the policies for multiple protected instances.

A backup policy may define that you want to make a backup at 6:30 p.m. each day, for example. You wish to keep daily backups for 6 months and rotate them to retain weekly backups for 2 years. For compliance purposes, you retain monthly backups for 5 years. A yearly backup is retained for 10 years. These retention values may appear excessive, but for an application that involves communication and messaging, you often need to retain backups for regulatory and compliance purposes for these long time frames. Azure Backup provides the flexibility to define policies to suit different application workloads and quickly enforce compliance.

Try it now

All your Azure backups are stored in a Recovery Services vault. To create a vault and backup policy, complete the following steps:

1. Open the Azure portal, and choose Create a Resource from the menu at top left.
2. Search for and select Backup and Site Recovery, and then choose Create.
3. Create a resource group, such as `azuremolchapter13`, and then enter a name for the vault, such as `azuremol`.
4. Select a location, and then review and create the vault.
5. When the vault has been created, choose Resource Groups from the menu at left in the portal, and then choose the resource group you created.
6. Select your Recovery Services vault from the list of available resources, choose Backup Policies from the menu at left, and then choose to add a policy.
7. Select the Azure Virtual Machine policy type, and provide a name for your new policy, such as `molpolicy`. By default, a backup is created each day.
8. Choose the most appropriate time zone from the drop-down menu. By default, Azure uses Universal Coordinated Time (UTC).

   If you wish, review and adjust the retention policies for daily, weekly, monthly, and yearly. The section on the concepts of backup and retention schedules detailed how you would select these values. These values typically vary as you create and apply backup policies to protect your VM instances.
9. When you're ready, select Create.

www.dbooks.org

The simple life

You can also configure VM backups when you create a VM in the Azure portal. On the Settings page where you configure virtual network settings or diagnostics and troubleshooting options, you can enable Azure Backup. You can pick an existing Recovery Services vault or create one, and then create or use a backup policy. You can't currently enable backups as part of the VM deployment in the Azure CLI or Azure PowerShell, but it usually takes a single command postdeployment to do so.

I like to plan a backup strategy, retention policies, and schedules, which is why these exercises created the Recovery Services vault and policies first. But if you want to create a VM and enable backups quickly, you can do that in the Azure portal in one step.

Now you have a backup policy, which also defines retention policies for various periods, but you have nothing to back up yet. Let's create a VM with Cloud Shell so that you can create a backup and, in a later exercise, replicate the data.

Try it now

To create a test VM for backup and replication, complete the following steps:

1. Select the Cloud Shell icon at the top of the Azure portal.
2. Create a VM with `az vm create`; provide the resource group name you created in the previous lab, such as `azuremolchapter13`; and then enter a VM name, such as `molvm`:

```
az vm create \
  --resource-group azuremolchapter13 \
  --name molvm \
  --image win2019datacenter \
  --admin-username azuremol \
  --admin-password P@ssw0rdMoL123
```

A backup policy is defined, and a test VM is ready. To see Azure Backup in action, let's apply your backup policy to the VM.

Try it now

To back up a VM with your defined policy, complete the following steps:

1. Select Resource Groups from the menu at left in the portal.
2. Choose the resource group, and then the VM you created.
3. Under Operations, select Backup.
4. Make sure that your Recovery Services vault is selected, and choose your backup policy from the drop-down menu.

5 Review the schedule and retention options, and then enable the backup. It takes a few seconds for the backup policy to be applied.

6 When the policy is enabled, return to the backup settings. The VM status reports `Warning (Initial backup pending)`.

7 To create the first backup, choose the Backup Now button, as shown in figure 13.5.

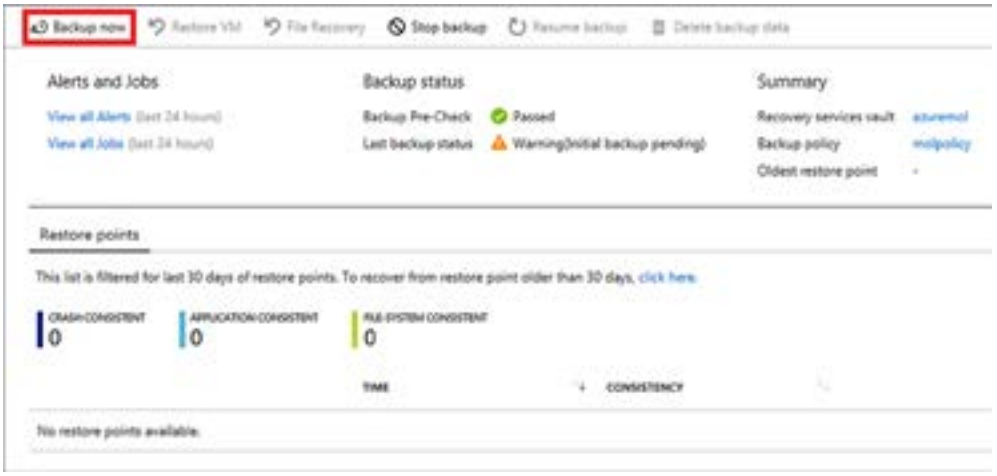


Figure 13.5 To create the first backup, select the Backup Now button. The status updates when complete and shows the latest backup time, latest restore point, and oldest restore point.

It can take 15 to 20 minutes for the first complete backup operation to finish. To see the progress of the backup job, you can select the View All Jobs option. There's no progress bar or percentage indicator, but you can make sure that the job is still running.

That's all it takes to back up VMs and protect your data in Azure! Keep reading to see how you can restore the data should something go wrong.

13.1.3 Restoring a VM

Azure Backup allows you to restore a complete VM or perform a file-level restore. In all my years in the field, file-level restore operations were the more common of the two. This type of restore job is usually performed when files are deleted or accidentally overwritten. File-level restores usually determine the retention policies for your backups. The more important the data, the more likely you want to retain backups longer, in case you get a late-night call to restore a file from six months ago.

A complete VM restore, as you might expect, restores the entire VM. Rarely have I performed a complete VM restore to bring a deleted VM back online. A great use case for a complete VM restore is to provide a test VM that's functionally equivalent to the original. You can restore a VM and then test a software upgrade or other maintenance

www.dbooks.org

procedure, which can help you identify potential problems and create a plan for handling the real production VM.

It's also important to test your backups regularly. Don't wait until you need to restore data in a real-world scenario. Trust Azure Backup, but verify that you know how and where to restore the data when needed!

File-level restore

A file-level restore is a pretty cool process in Azure Backup. To give you flexibility in how and where you restore files, Azure creates a recovery script that you download and run. This recovery script is protected by a password so that only you can execute the recovery process. When you run the recovery script, you're prompted to enter the password before you can continue. The window for downloading the recovery script is shown in figure 13.6.

When you run the recovery script, your recovery point is connected as a local filesystem on your computer. For Windows VMs, a PowerShell script is generated, and a local volume is connected, such as F:. For Linux VMs, the recovery point is mounted as a data disk, such as /dev/sdc1 in your home volume. In both cases, the recovery script clearly indicates where you can find your files.

When you've finished restoring files from the recovery vault, return to the Azure portal, and select the Unmount Disks option. This process detaches the disks from your local computer and returns them for use in the recovery vault. Don't worry if you forget to perform this unmount process in the heat of the moment when you need to restore files quickly for a production VM! Azure automatically detaches any attached recovery points after 12 hours.



Figure 13.6 When you perform a file-level restore, you choose a recovery point to restore. Then a recovery script is downloaded to your computer. You can execute this script only by entering the generated password. The recovery script mounts the recovery point as a local volume on your computer. When you've restored the files you need, unmount the disks from your computer, which returns them for use in the recovery vault.

Complete VM restore

A complete VM restore creates a VM, connects the VM to the virtual network, and attaches all the virtual hard disks. Let's try the process for a complete VM restore. Because it's always best to test maintenance updates before you perform them for real, this restore exercise is good practice.

Try it now

To restore a complete VM, complete the following steps:

1. From your resource group, select the VM that you backed up in the previous exercise.
2. Select the Backup option from the menu at left in the VM. The backup overview should report that a recovery point has been created, as shown in figure 13.7. If not, wait a few minutes, and then come back to this exercise. Or just read what the process entails.

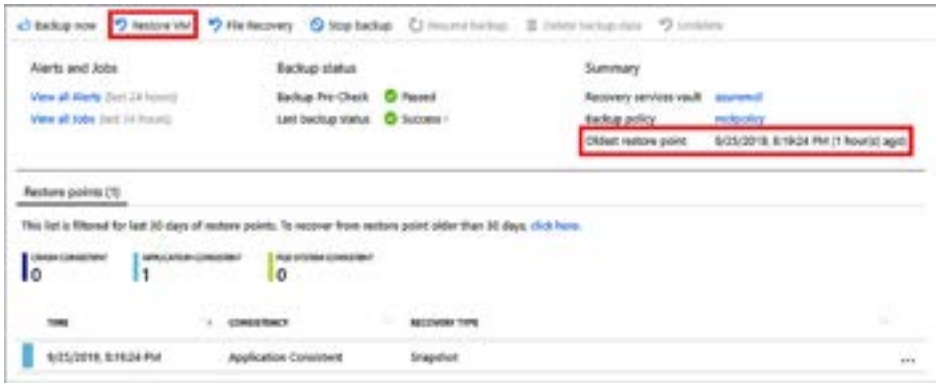


Figure 13.7 When the VM backup is complete, the overview page shows the data from the last backup and available restore points. To start the restore process, select Restore VM.

3. Select the Restore VM button, choose a restore point from the list, and then select OK.
4. Choose a restore point, and choose how to restore the VM. You can choose to create a new VM or replace an existing VM.

   The default option is to create a new VM. In this configuration, a new VM is created and connected to the specified virtual network, and the disks are restored and connected.

   You can also choose to replace an existing VM. In this scenario, the disks are restored from the backup and attached to the existing VM. Whatever virtual network or other configuration options applied to the VM are retained.
5. For this exercise, chose to restore to a new VM. Provide a name for the restored VM, such as `restoredvm`, and then review the settings for virtual network and storage. In production, you typically connect the restored VM to an isolated virtual network so that you don't affect production traffic.
6. Select OK and then Restore.

www.dbooks.org

It takes a few minutes to connect the recovery point and create a restored VM with the previous disks attached. At this point, you could connect to the restored VM to test software upgrades or restore large amounts of data as needed.

You can also back up a web app, so this approach isn't just for VMs. The process is a little different, but the concepts are the same. Moving your application model to a PaaS solution like a web app doesn't mean you can forget the basics of data backups and retention!

13.2 Azure Site Recovery

Remember when we discussed Cosmos DB, and you learned that with the click of a button, your data is replicated to a completely different Azure region for redundancy and fault tolerance? You can do that with entire VMs too! Azure Site Recovery is a powerful service that can do way more than just replicate VMs to a different region. Figure 13.8 outlines how Azure Site Recovery acts to orchestrate workloads between locations.

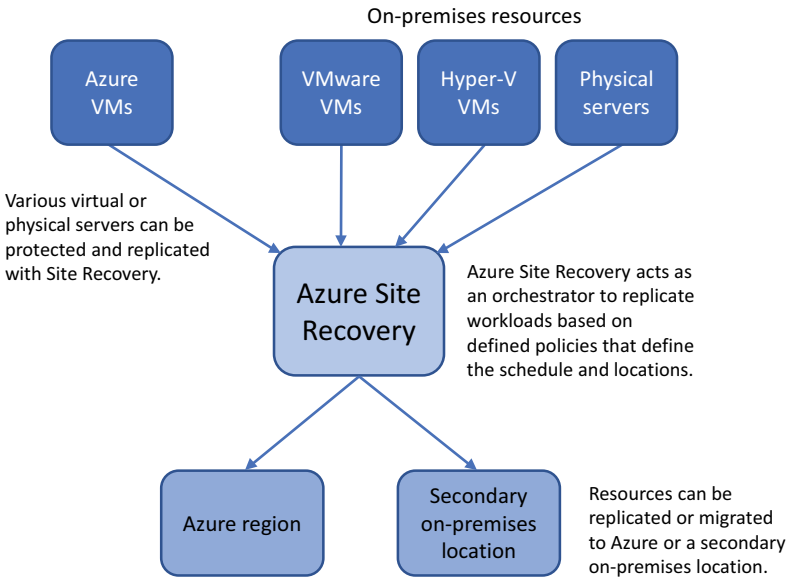


Figure 13.8 Azure Site Recovery orchestrates the replication and migration of physical or virtual resources to another location. Both on-premises locations and Azure can serve as source and destination points for protection, replication, or migration.

An important aspect is that Azure Site Recovery is for more than just Azure VMs; Site Recovery can be used to replicate on-premises VMware or Hyper-V VMs to Azure for disaster recovery (DR) or as part of a migration to Azure. You can also use Azure Site Recovery purely as the orchestrator to replicate on-premises VMs from one location to a secondary on-premises location.

In the same way that Azure Backup doesn't mean "works only with Azure," Azure Site Recovery doesn't mean "replicates only Azure VMs." Both Azure Backup and Azure Site Recovery can be used as hybrid solutions for backup and disaster recovery. These Azure services can be used to protect all your workloads, both on-premises and in Azure. Then a single reporting structure for compliance and validation can be generated to make sure that all the workloads you think are protected are indeed safe from data loss.

Why would you use Azure Site Recovery? Two primary reasons are most common: replication and migration.

Replication protects you from a complete Azure region outage. It would take a catastrophic event for an entire region to go offline, but when you work in IT, you know that anything is possible. Even Availability Sets and Availability Zones, which we talked about in chapter 7, typically protect you only from a smaller outage within an Azure region. If the entire region goes down, your app will go down. With Site Recovery, your entire application environment, including virtual network resources, is replicated to a secondary Azure region. At the click of a button, that secondary location can be brought online and made active. Then traffic can route to this secondary location and begin to serve your customers. Figure 13.9 shows a high-level overview of how Azure Site Recovery protects your environment.

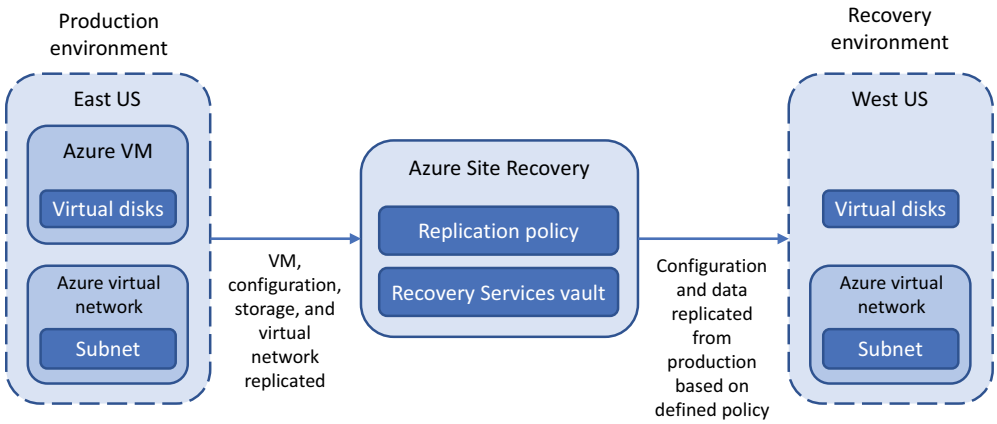


Figure 13.9 Azure Site Recovery replicates configuration, data, and virtual networks from the production environment to a recovery environment. The VMs aren't created in the recovery environment until a failover is initiated. Only the data replicates.

The VM is just metadata that defines what the VM size is, what disks are attached, and what network resources the VM connects to. This metadata is replicated, which allows the VMs to be created quickly when a failover is initiated. The virtual disks are replicated to the recovery environment and are attached when a recovery VM is created during a failover event.

www.dbooks.org

For Azure-to-Azure replication, there's no defined replication schedule. The disks replicate in almost real time. When data on the source virtual disks changes, it's replicated to the recovery environment. For hybrid workloads, where you protect on-premises VMware or Hyper-V VMs, you define policies that control the replication schedule.

If we focus on Azure-to-Azure replication, how does the data replicate in near real time? A storage account cache is created in the production environment location, as shown in figure 13.10. Changes written to the production virtual disks are immediately replicated to this storage account cache. Then the storage account cache is replicated to the recovery environment. This storage account cache acts as a buffer so that any replication delays to the distant recovery location don't affect performance on the production workload.

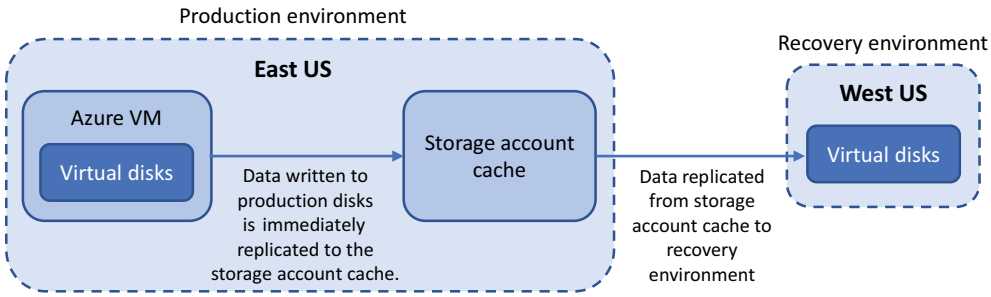


Figure 13.10 Changes on the production disks are immediately replicated to a storage account cache. This storage account cache prevents performance effects on the production workloads as they wait to replicate changes to the remote recovery location. Then the changes from the storage account cache are replicated to the remote recovery point to maintain data consistency.

The process of configuring Site Recovery for Azure-to-Azure replication is straightforward, but it takes some time to create all the necessary replicated resources and complete the initial data replication. In the end-of-chapter lab, you'll configure this Azure-to-Azure replication.

What can you do with VMs replicated to a secondary location with Azure Site Recovery? For the most part, cross your fingers and hope that you don't need them! But there are a couple of scenarios in which you would need them.

The first should be obvious: a major outage. If an Azure region becomes totally unavailable, such as because of a natural disaster in the area, you can initiate a failover of your resources. This failover tells Azure Site Recovery to create VMs in the recovery location based on the replicated VM metadata and then attach the appropriate virtual hard disks and network connections. You could also be proactive here: if a natural disaster is forecast to hit an Azure region, you could initiate a failover *before* the event takes place. This approach lets you decide when to incur some potential downtime as

the resources fail over to the secondary location, typically outside primary business hours. When the forecast event has passed in the primary Azure region, you can fail back your resources and continue to run as normal.

The second scenario in which you may fail over is to test that the process works. In the same way that backups should be regularly tested, you should test a replication and failover plan. It would be pretty embarrassing and stressful to find that when you need to bring a secondary location online, there's some misconfiguration on the virtual networks, or one of the applications doesn't fail over gracefully. Helpfully, Azure provides an option specifically for testing failover. An isolated Azure virtual network is typically used in the secondary location, and the production workloads continue to run as normal in the primary location. If you use Azure Site Recovery, be sure to test the failover process regularly!

13.3 Lab: Configuring a VM for Site Recovery

There are several prerequisites for configuring on-premises VMware or Hyper-V replication with Azure Site Recovery. It's a great feature, both for disaster-recovery purposes and to migrate VMs to Azure, but it takes up way more than your lunch break! If you want to learn more about those scenarios, head over to http://mng.bz/x71V.

Let's set up Azure-to-Azure replication with the test VM you created and backed up earlier:

1. In the Azure portal, choose Resource Groups from the menu at left.
2. Select the resource group you used in the previous exercises, such as azuremolchapter13.
3. Select the VM you created in the earlier exercises, such as molvm.
4. Choose Disaster Recovery from the menu at left in the VM window.
5. Under the advanced settings, look at the default settings used by Azure Site Recovery to create a resource group and a virtual network at the destination location. A storage account cache is created to replicate from the source virtual disks, and a Recovery Services vault and policy are created to control the replication process.
6. There's nothing you need to change here, although if you use Site Recovery in production and have multiple VMs to protect, you'll need to review how the VMs map to existing replicated virtual networks and subnets. For this lab, review and enable the replication using the default values.

Now go back to work. Seriously! It takes a while to configure all the replicated resources and complete the initial data sync. Don't wait around unless your boss is fine with your taking a really long lunch break today!

www.dbooks.org

Keeping your backups safe from deletion

I hope that as a best practice, you've deleted resource groups and their resources at the end of each chapter to keep your free Azure credits available for use in the rest of the book.

If you have VMs that are protected with Azure Backup or Site Recovery, you can't delete the Recovery Services vault or resource group for the VM. The Azure platform knows you have active data that's backed up or replicated and prevents those resources from being deleted.

To delete protected VMs, first disable any active backup jobs or replicated VMs. When you do so, you can choose to retain the protected data or remove it. For the lab exercises in this chapter, choose to delete the restore points. As a security feature, Azure automatically soft-deletes these restore points and lets you undelete them for 14 days. There's nothing for you to configure here, and you can't forcefully remove these soft-deleted restore points. I don't recommend it, but you can also disable the soft-delete function of a Recovery Services vault by selecting the Properties of the vault in the Azure portal.

The good news is that the rest of the resource group can be deleted, and you don't pay for these soft-deleted restore points. When the 14-day soft-delete period is up, the Recovery Services vault can be deleted as normal. The goal here is to protect you from accidental, or malicious, deletion of restore points and give you time to realize that they're actually needed and recover them.

14 Data encryption

The security of your data is important. More specifically, the security of your customers' data is critical. We hardly go a week without reading in the news about a major company that encountered a data breach. Often, these incidents are caused by a lack of security, misconfiguration, or plain carelessness. In this digital age, it's all too easy for attackers to automate their attempts to gain access to your data. The time to recover from a security incident at an application level may be nothing compared with how long it takes the business to regain the trust of its customers if *their* data was exposed.

Azure includes encryption features that make it hard to claim you don't have the time or expertise to secure your data. In this chapter, we examine how to encrypt data stored in Azure Storage, on managed disks, or the complete VM. Entire books have been written about data encryption, and this chapter doesn't dive deep into encryption methods and considerations. Instead, you see how to enable some of the core Azure features and services to secure your data throughout the application lifecycle.

14.1 What is data encryption?

When you purchase something online, do you check that there's a little padlock icon on the address bar to indicate that the website uses HTTPS? Why is it bad to send your credit details over a regular, unsecured HTTP connection? Every bit of data in a network packet that flows between devices could potentially be monitored and examined. Figure 14.1 shows how shopping online without an HTTPS connection could be bad for your credit card statement.

There's no excuse for web servers to use unsecure connections. Every web app that you create in Azure automatically has a wildcard SSL certificate applied to it.

www.dbooks.org

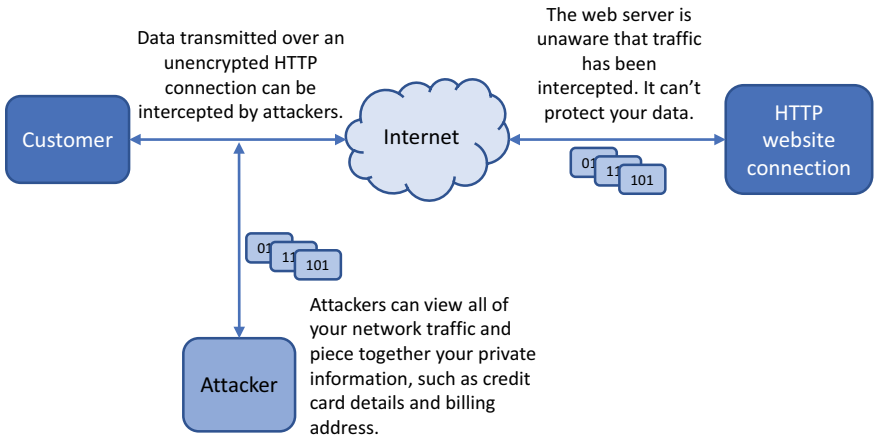


Figure 14.1 In this basic example, an attacker could intercept network traffic that's sent over an unencrypted HTTP connection. Because your data isn't encrypted, the attacker could piece together the network packets and obtain your personal and financial information. If you instead connect to the web server over an encrypted HTTPS connection, an attacker can't read the contents of the network packets and view the data.

An *SSL certificate* is a digital component that's used to secure the web server and allow a web browser to validate the connection. A wildcard SSL certificate can be used across an entire domain, such as *.azurewebsites.net, the default domain for web apps. When you created a web app in chapter 3, you could have added https:// to the web address and started to use encrypted communications with your web apps. That's all there is to it!

Custom SSL certificates are relatively cheap and easy to implement. Through projects such as Let's Encrypt (https://letsencrypt.org), you can obtain a certificate for free and automatically configure your web server in minutes. You can also buy and use an App Service Certificate that integrates directly into Web Apps. App Service Certificates are stored in Azure Key Vault, which we'll look at more in chapter 15.

As you design and build applications in Azure, you should implement secure communications wherever possible. This approach helps secure the data while it's in transit. But what about when that data is written to disk? A similar process exists for disks and VMs that secures and protects your data at rest. Figure 14.2 shows how disk and VM encryption works.

I hope that these simplified examples of data encryption in Azure motivate you to implement encryption as you design and build applications in Azure. Most customers expect their data to be secured, and many companies have regulatory and compliance mandates that require encryption. Don't think only about the potential fines to the business for a data breach or the loss of customer trust. Consider the risk that the customers' personal and financial data will be exposed and how that exposure could

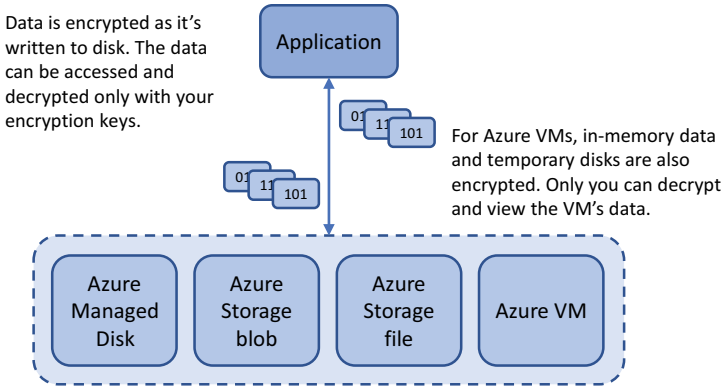


Figure 14.2 When you encrypt your data, only you can decrypt and view the contents. If an attacker were to gain access to a virtual disk or individual files, they wouldn't be able to decrypt the contents. Encryption methods can be combined: customers can connect to your web over HTTPS, you can force traffic to storage accounts to be over HTTPS, and you can encrypt the data that's written to disk.

affect their daily lives. You probably don't like the idea of your own data being exposed, so do all you can to protect the data of your customers.

14.2 Encryption at rest

If data encryption is so important, how do you use it in Azure? Just keep doing what you've already learned in this book! Right at the start, I mentioned that all of your VMs should use managed disks, right? There are many good reasons for that, one of which is security. A managed disk is automatically encrypted. There's nothing for you to configure, and there's no performance impact when it's enabled. There's no opt-out here; your data is automatically encrypted at rest with managed disks.

What does it mean for the data to be *encrypted at rest*? When you use managed disks, your data is encrypted when it's written to the underlying Azure storage. The data that resides on the temporary disks, or data that exists in memory on the VM, isn't encrypted. Only when the OS or data disk's data *rests* on the underlying physical disk does it become encrypted. Figure 14.3 shows how the data is encrypted as it's written to a managed disk.

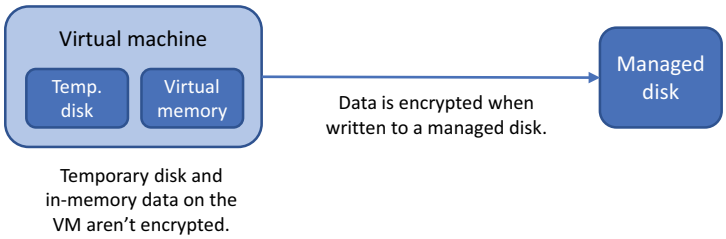


Figure 14.3 As data is written to a managed disk, it's encrypted. In-memory data on the VM, or data on temporary disks local to the VM, isn't encrypted unless the entire VM is enabled for encryption, which we'll look at in section 14.4.2. The automatic encryption of data written to managed disks causes no overhead to the VM. The Azure platform performs the encryption operation on the underlying storage. The VM doesn't need to handle any encrypt/decrypt processes.

www.dbooks.org

This encryption at rest for managed disks means that there's no performance impact on the VMs. There's no additional processing for the VM to perform to encrypt and decrypt the data, so all the available CPU power can be used to run applications. In typical VM encryption scenarios, the VM uses a certain amount of compute power to process and manage the data encryption. The trade-off to the automatic managed disk encryption is that only the OS and data disks are secured. Potentially, other in-memory or temporary disk data on the VM could be exposed.

Microsoft manages the digital encryption keys within the Azure platform with the automatic encryption of managed disks. This does create another trade-off in that you can automatically encrypt your data without the need to create, manage, rotate, or revoke keys, but you have to trust Microsoft to protect those keys.

14.3 Storage Service Encryption

Automatic managed disk encryption is great, but what if you use Azure Storage for blob or file storage? Azure Storage Service Encryption (SSE) lets you encrypt data at the storage account level. Data is encrypted as it's written to the account. Again, Microsoft handles the encryption keys, so no management overhead or configuration is required. The Azure platform abstracts the key generation and management for you. If you prefer, you can create and use your own encryption keys, with a little additional management overhead. Like automatic managed disk encryption at rest, Azure storage encryption is automatically enabled when you create an account.

The goal of both automatic managed disk encryption and SSE is to make it as easy as possible for you to encrypt your data and spend more time designing, building, and running your applications. Figure 14.4 shows how SSE protects your data and can also force secure communications when data is in transit.

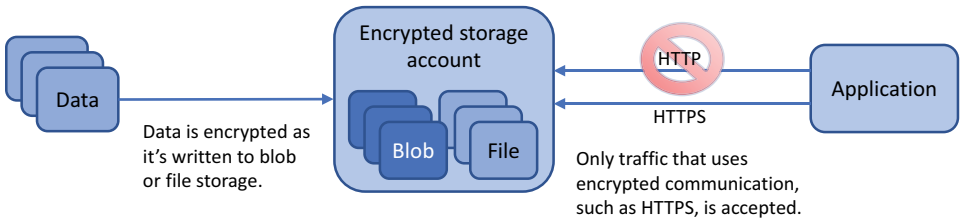


Figure 14.4 When you enable SSE, Azure blobs and files are encrypted as the data is written to disk. Azure tables and queues aren't encrypted. For additional data security, you can force all communications with a Storage account to use secure communication protocols, such as HTTPS. This protects the data in transit until the moment it's encrypted on disk.

Forcing storage traffic to use secure transfers

Along with enabling SSE, you can force all storage requests and transfers to use a secure communication method. This setting forces all REST API calls to use HTTPS and all Azure file connections that don't enable encryption, such as older versions of the SMB protocol, to be dropped.

(continued)

Azure SDKs, such as the Python examples we examined in chapter 4, can use encrypted connections. The reference docs for each language-specific SDK provides guidance on how to implement secure communications.

The use of secure communications should be built into applications from the start. It may cause problems to enable secure communications on an existing application if some components weren't originally configured appropriately. At the very least, test secure communications for an existing application in a development environment first.

Try it now

To create a storage account and enable both encryption and secure communications, complete the following steps:

1. Open the Azure portal, and choose the Cloud Shell icon from the top menu.
2. Create a resource group; provide a name, such as `azuremolchapter14`; and provide a location, such as `eastus`:

```
az group create --name azuremolchapter14 --location eastus
```

3. Create a storage account with `az storage account create`. Provide a unique name, such as `azuremolstorage`, and enter the resource group that you created in step 2. Enter a storage account type, such as `Standard_LRS` for locally redundant storage. To force secure communications, set `--https-only`.

```
az storage account create \
  --name azuremolstorage \
  --resource-group azuremolchapter14 \
  --sku standard_lrs \
  --https-only true
```

4. Verify that the storage account is encrypted and enabled for secure communications by querying for `enableHttpsTrafficOnly` and the encryption parameters:

```
az storage account show \
  --name azuremolstorage \
  --resource-group azuremolchapter14 \
  --query [enableHttpsTrafficOnly,encryption]
```

The output is similar to the following:

```
[
  true,
  {
    "keySource": "Microsoft.Storage",
```

www.dbooks.org

```
    "keyVaultProperties": null,
    "services": {
      "blob": {
        "enabled": true,
        "lastEnabledTime": "2019-09-27T03:33:17.441971+00:00"
      },
      "file": {
        "enabled": true,
        "lastEnabledTime": "2019-09-27T03:33:17.441971+00:00"
      },
      "queue": null,
      "table": null
    }
  }
]
```

14.4 VM encryption

The automatic encryption of Azure Managed Disks helps provide a level of VM security. For a comprehensive approach to VM data security, you can encrypt the VM itself. This process involves more than encrypting the underlying virtual hard disks. The OS disk and all attached data disks, along with the temporary disk, are encrypted. The VM memory is also encrypted to further reduce the attack surface. You use digital keys to encrypt VMs.

One advantage of encrypting the entire VM is that you manage the encryption keys. These encryption keys are securely stored in Azure Key Vault, and you can choose between using software- and hardware-generated keys. You control these keys, so you can define access to them and use role-based access controls and auditing to track usage. You can also rotate the encryption keys on a defined schedule, much like changing your password every 60 or 90 days. These additional controls and management tasks for encryption keys add some management overhead but provide maximum flexibility for securing your data, and they may be required for certain regulatory purposes. Let's look a little more at Azure Key Vault.

14.4.1 Storing encryption keys in Azure Key Vault

We'll spend chapter 15 on Azure Key Vault, but I want to show you the power of data encryption and VM encryption first. As a quick overview, Azure Key Vault is a digital vault that allows you to securely store encryption keys, SSL certificates, and secrets such as passwords. For redundancy, key vaults are replicated across Azure regions. This replication protects your keys and secrets, and ensures that they're always available for use.

Only you have access to your key vaults. You generate and store objects in key vaults and then define who has access to those vaults. Microsoft manages the underlying Key Vault service but has no access to the contents of the vaults. This security boundary means when you encrypt your data in Azure, you are the only one who can decrypt and view it.

Try it now

To create a key vault and encryption key, complete the following steps:

1. Open the Azure portal, and choose the Cloud Shell icon from the top menu.
2. Create a key vault with the `az keyvault create` command; specify the resource group you created in the previous exercise, such as azuremolchapter14; and then provide a unique name for your key vault, such as `azuremolkeyvault`:

```
az keyvault create \
  --resource-group azuremolchapter14 \
  --name azuremolkeyvault \
  --enabled-for-disk-encryption
```

Let's pause and think about why you add a parameter for `--enabled-for-disk-encryption`. When you encrypt a VM, the Azure platform needs to be able to start and decrypt the VM so that it can run. The Azure platform doesn't have any permissions to access that data, and Microsoft doesn't have access to view and use those encryption keys for anything other than starting a VM. When you enable a key vault for disk encryption, you grant permissions for Azure to access the key vault and use the encryption key associated with a VM.

Again, Microsoft doesn't have access to these keys or your data—only the ability to start your encrypted VM. It's pretty hard to do much with an encrypted VM when it can't boot. Figure 14.5 shows how the Azure platform uses the encryption key to start an encrypted VM.

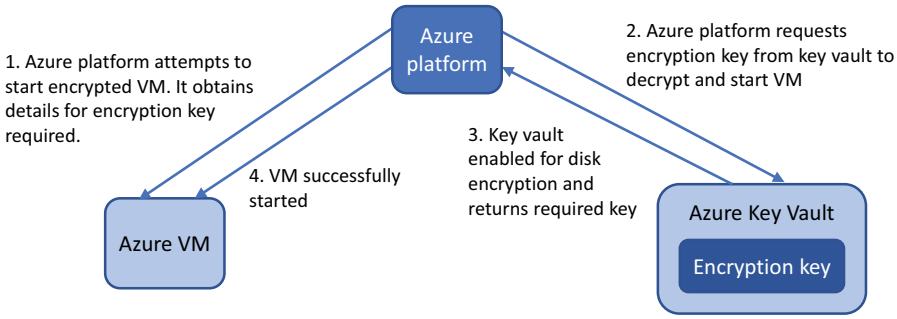


Figure 14.5 When a key vault is enabled for disk encryption, it grants permission for the Azure platform to request and use the encryption key to successfully start an encrypted VM.

Keys can be created and stored in software, or they can be stored in hardware security modules (HSMs) for additional security. For many purposes, software keys work great, although you may have security mandates that require the use of HSMs. We'll discuss this topic more in chapter 15.

www.dbooks.org

3 To create a key, specify the vault you created in step 2, such as azuremolkeyvault, and then provide a key name, such as `azuremolencryptionkey`:

```
az keyvault key create \
  --vault-name azuremolkeyvault \
  --name azuremolencryptionkey \
  --protection software
```

14.4.2 Encrypting an Azure VM

The encryption key you created in section 14.4.1 can be used to encrypt many VMs, if desired. This approach minimizes the overhead of key management and, if you use virtual machine scale sets, allows you to autoscale the number of VM instances without the need to generate encryption keys each time. The alternative is that each VM has its own encryption key, which adds complexity but provides a layer of security for your VMs. If you have the same encryption key used for backend application VMs and then database VMs, for example, a theoretical attacker with that one key could gain access to the data of both sets of VMs. If different keys are used, the number of potentially compromised VMs is lower. In the end-of-chapter lab, you'll encrypt a single VM, although the same process can work with a scale set that has multiple VMs but uses just the one key. Especially when you work with larger, autoscaling applications, be sure to design and build in security features.

When you encrypt a VM, an Azure VM extension is installed. The extension controls the encryption of the OS disk, temporary disk, any attached data disks, and in-memory data, as shown in figure 14.6. For Windows VMs, the BitLocker encryption mechanism is used. For Linux VMs, dm-crypt is used to process the encryption. Then the VM extension can report back on the status of encryption and decrypt the VM as desired.

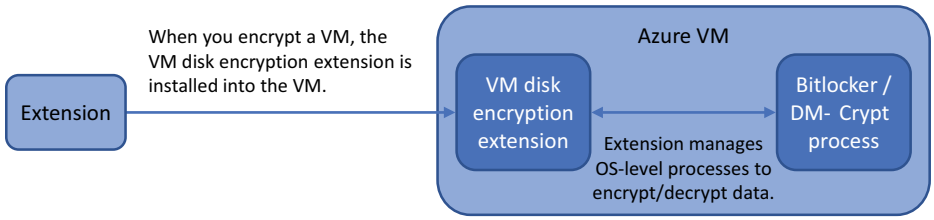


Figure 14.6 When you encrypt a VM, the Azure disk encryption extension is installed. This extension manages the use of BitLocker on Windows VMs or dm-crypt on Linux VMs, to perform the data encryption on your VM. The extension is also used when you query the encryption status for a VM.

Because the VM disk encryption extension relies on BitLocker or dm-crypt, there are some limitations on the use of VM encryption. Most Azure Marketplace images support disk encryption, although some restrictions exist on VM sizes that support encryption or encryption of connected network file shares such as Azure files. For the

most comprehensive information on supported limitations and considerations for VM encryption, read the latest Azure docs at http://mng.bz/yyvd.

This chapter provided a quick intro to the data security and encryption features in Azure. Automatic encryption for managed disks and SSE doesn't require much configuration, so there's no real barrier to prevent you from using them.

14.5 Lab: Encrypting a VM

Let's see all this in action by encrypting a VM with the encryption key you stored in your key vault:

1. Create a VM. Most Linux images in the Azure Marketplace support encryption, as do the Windows Server images from Server 2008 R2 and later. To make it quick and easy, create an Ubuntu LTS VM, just as you have for most of this book. As the VM requires enough memory to perform the disk encryption operation, specify a size of Standard_D2s_v3:

```
az vm create \
  --resource-group azuremolchapter14 \
  --name molvm \
  --image ubuntults \
  --size Standard_D2s_v3 \
  --admin-username azuremol \
  --generate-ssh-keys
```

2. Enable encryption on the VM, and provide the name of the Azure Key Vault and digital key you created in a previous exercise:

```
az vm encryption enable \
  --resource-group azuremolchapter14 \
  --name molvm \
  --disk-encryption-keyvault azuremolkeyvault \
  --key-encryption-key azuremolencryptionkey
```

   It takes a few minutes to install the Azure VM disk encryption extension and begin the process of encrypting the VM.

3. When encryption has started, monitor the progress, and be ready to restart the VM to complete the encryption process. View the status as follows:

```
az vm encryption show \
--resource-group azuremolchapter14 \
--name molvm \
--query 'status'
```

   Here's some example output of a VM in the process of being encrypted. At the start, the status message reports as

```
[
  {
    "code": "ProvisioningState/succeeded",
    "displayStatus": "Provisioning succeeded",
    "level": "Info",
```

www.dbooks.org

```
    "message": "OS disk encryption started",
    "time": null
  }
]
```

It can take a while to complete the disk encryption, so this may be another good lab exercise to come back to in an hour or so—unless you want a long lunch break! Hey, I'm *still* not your boss, but it gets boring looking at the same encryption status message.

4 When the encryption status reports as `Encryption succeeded for all volumes`, restart the VM:

```
az vm restart --resource-group azuremolchapter14 --name molvm
```

You can then check the status of VM encryption again with `az vm encryption show` to confirm that the VM reports as Encrypted.

Remember your housecleaning chores

These last two end-of-chapter labs didn't take long to complete, but they may have taken a while to finish. Don't forget to go back and delete resources when you're done with them.

As discussed in chapter 13, remember that you need to disable Azure Backup or Site Recovery protection before you can delete the Recovery Services vault or resource group (after waiting the 14 days for the free soft-delete recovery points to expire). Make sure to go back and clean up those lab resources before they start to use up too many of your free Azure credits.

15 Securing information with Azure Key Vault

Almost every week, there's news of a cybersecurity incident with a major company. In the same way that you've used various forms of automation to grow or replicate your applications and data, attackers automate their own actions. It's unlikely that a single person will try manually to compromise the security of your systems. This concept makes it difficult to defend your systems 24 hours a day, 7 days a week, 365 days a year (okay, or 366 days!).

Chapter 14 discussed how to encrypt your data and VMs. This process is a great first step, and we briefly looked at how to create and use encryption keys stored with the Azure Key Vault service. Secure data such as keys, secrets, and certificates is best stored in a digital vault like a key vault, which can centrally manage, issue, and audit the use of your critical credentials and data. As your applications and services need access to different resources, they can automatically request, retrieve, and use these keys, secrets, and credentials. In this chapter, you'll learn why and how to create a secure key vault, control access, and then store and retrieve secrets and certificates.

15.1 Securing information in the cloud

As applications become more complex and the risk of cyberattacks grows, security becomes a critical part of how you design and run your services. Especially as you run more internet-facing applications, either on-premises or in the cloud, making sure you minimize the risk of unauthorized data access should be one of the main design areas you focus on. There's no point having the greatest pizza store in the world if customers don't trust you with their payment details or personal information.

A common way to provide security for applications and services is through the use of digital keys, secrets, and certificates, as shown in figure 15.1. Rather than using a username and password that must be entered manually time and again—or,

www.dbooks.org

maybe worse, written in an unencrypted configuration file—you use a digital vault to store these secure credentials and data. When an application or service requires access, it requests the specific key or secret it needs, and an audit trail is created to trace any possible security misuse or breach.

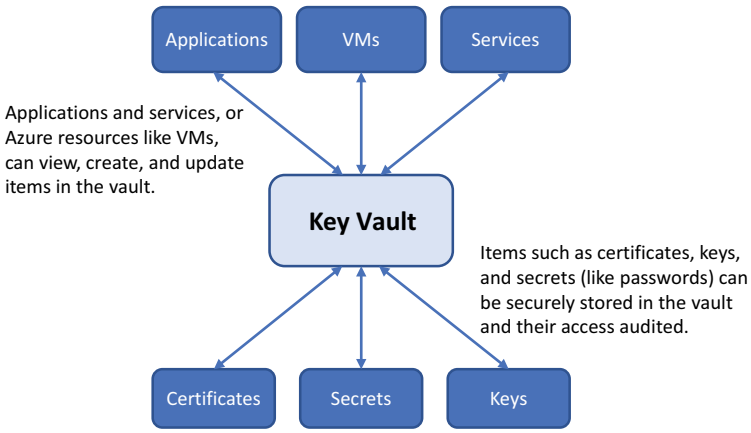


Figure 15.1 Azure Key Vault provides a secure way to store digital information such as certificates, keys, and secrets. Then these secure items can be accessed directly by your applications and services, or Azure resources such as VMs. With minimal human interaction, you can centrally distribute secure credentials and certificates across your application environments.

When designed and implemented correctly, these digital vaults are almost fully automated and secure. Services can request a new digital certificate, be issued one that's then securely stored in the vault, and use it to authorize themselves against other application components. Servers can configure software by retrieving secrets such as passwords from the digital vault and then installing application components without the credentials being stored in a text-based configuration file. An application administrator can centrally manage all the secrets, keys, and certificates for a service and update them regularly as needed.

Azure Key Vault provides all these digital security features and allows you to tightly control which users and resources can access the secure data. Key vaults can be securely replicated for redundancy and improved application performance, and integrate with common Azure resources such as VMs, web apps, and Azure Storage accounts.

15.1.1 Software vaults and hardware security modules

Before we jump into a hands-on example of how to create and use a key vault, it's important to understand the way your secure information is stored in a vault. As shown in figure 15.2, all the keys, secrets, and certificates in a key vault are stored in a hardware security module (HSM). These devices aren't unique in Azure; they're industrywide hardware devices that provide a high level of security for any data stored on them.

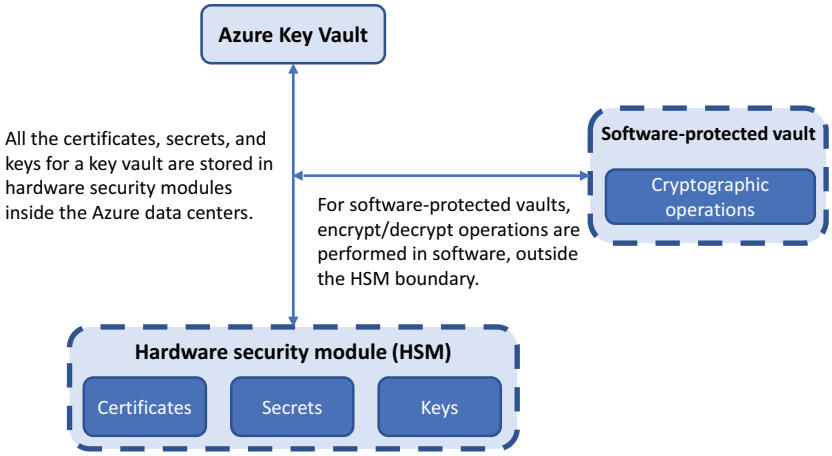


Figure 15.2 Azure Key Vault is a logical resource in Azure, but any certificates, secrets, and keys are stored in an HSM. For development or test scenarios, a software-protected vault can be used, which performs any cryptographic operations—such as encrypting or decrypting data—in software, not in hardware on the HSM. For production, you should use an HSM-protected vault, where all the processing is done on hardware.

Currently, you can use two types of key vaults: software-protected and HSM-protected. The difference may be confusing, which is why I want to clear it up before we get started:

- A *software-protected vault* stores keys, secrets, and certificates in an HSM, but any cryptographic operations that are required to encrypt or decrypt its contents are performed by the Azure platform in software. Software-protected vaults are great for development and test scenarios, although you may decide that production workloads require a slightly more secure way to perform the cryptographic operations.
- An *HSM-protected vault* stores keys, secrets, and certificates in an HSM, and cryptographic operations that are required to encrypt or decrypt its contents are performed directly on the HSM. You can also generate your own secure keys in an on-premises HSM and then import them into Azure. There are some additional tools and processes to follow, but this way, you ensure that you completely control the keys and that they never leave the HSM boundary.

  To maximize the security and integrity of your data, hardware-protected vaults are the preferred approach for production workloads.

Regardless of which type of vault you use, it's important to remember that all of your data is stored securely on a Federal Information Processing Standard (FIPS) 140–2 Level 2 validated (at a minimum) HSM and that Microsoft can't access or retrieve your keys. There's an additional cost for HSM-protected vaults, so as with anything in Azure and cloud computing, balance the cost versus the risk of your data being compromised.

www.dbooks.org

15.1.2 Creating a key vault and secret

A digital vault sounds great, but you may be a little unsure how to make use of the power that Azure Key Vault provides. Let's build an example of a basic server that runs a database such as MySQL Server, as shown in figure 15.3.

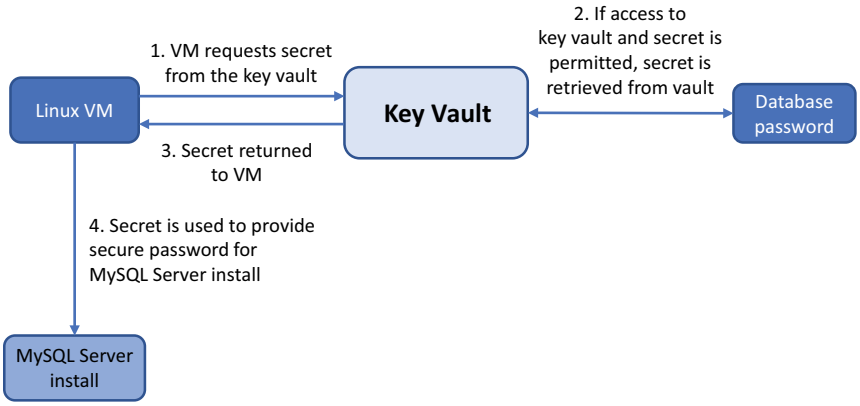


Figure 15.3 In the next few exercises, you'll build an example of a secret stored in a key vault that can be used as the database password for a MySQL Server install. A VM is created that has permissions to request the secret from the key vault. Then the retrieved secret is used to automatically enter a secure credential during the application install process.

One of the first exercises in this book was to create a VM and then install the LAMP web server stack. You were likely prompted for a MySQL Server password, or a blank password was automatically used. Now that you know all about key vaults, you can retrieve a password from the vault automatically and use it dynamically to install and configure the server.

Try it now

To create a key vault and add a secret, complete the following steps:

1. Open the Azure portal; launch Cloud Shell; and create a resource group, such as `azuremolchapter15`:

```
az group create --name azuremolchapter15 --location eastus
```

2. Create a key vault with a unique name, such as `azuremol`, and enable it for deployment so that you can use the vault to inject keys and certificates into a VM:

```
az keyvault create \
  --resource-group azuremolchapter15 \
```

```
  --name azuremol \
  --enable-soft-delete \
  --enabled-for-deployment
```

By default, your Azure user account is assigned full permissions to the key vault. For these exercises, this is fine, although as a security best practice, you should consider limiting who can access your key vault. You can add the `--no-self-perms` parameter to skip permission assignment to your account

3 Create a secret, such as `databasepassword`, and assign a password value, such as `SecureP@ssw0rd`. (Yep, really secure, right?) This secret can be used as the credentials for a database server, which you'll deploy in the following exercises:

```
az keyvault secret set \
  --name databasepassword \
  --vault-name azuremol \
  --description "Database password" \
  --value "SecureP@ssw0rd"
```

4 You have full permissions to the key vault, so you can view the contents of your secret:

```
az keyvault secret show \
--name databasepassword \
--vault-name azuremol
```

From a management perspective, you can also perform common actions such as backing up and restoring, downloading, updating, and deleting items stored in a key vault. One additional property that you set when the key vault was created is the option to `enable-soft-delete`. If your applications and services can't retrieve the secrets they need from the key vault, you could have a pretty large application outage to deal with! A key vault can store metadata for secrets for up to 90 days after they're truly deleted, which allows you to recover data that's incorrectly or maliciously deleted.

5 Delete the key you just created to simulate a mistake or possibly someone with malicious intent:

```
az keyvault secret delete \
  --name databasepassword \
  --vault-name azuremol
```

6 Recover the secret so that you can continue to use the database password with your application and services:

```
az keyvault secret recover \
  --name databasepassword \
  --vault-name azuremol
```

If you truly want to remove a secret, you also have the option to purge a deleted secret. This option permanently removes the secret without waiting for the default 90-day recovery period to elapse.

www.dbooks.org

Feel free to use `az keyvault secret show` again to view the information on your secret and confirm that the password you stored is there after you deleted and restored it. Now let's move on to see how a VM can access a key vault and use the secret to install the MySQL Server.

15.2 Managed identities for Azure resources

The ability to use Azure Key Vault to store secrets or keys is great, but how do you access these secrets? The Azure CLI or Azure PowerShell can access the information stored in a key vault, but it's often more convenient to allow your VMs or applications to retrieve secrets or keys directly when they need them. One way to do this is with managed identities for Azure resources, as shown in figure 15.4.

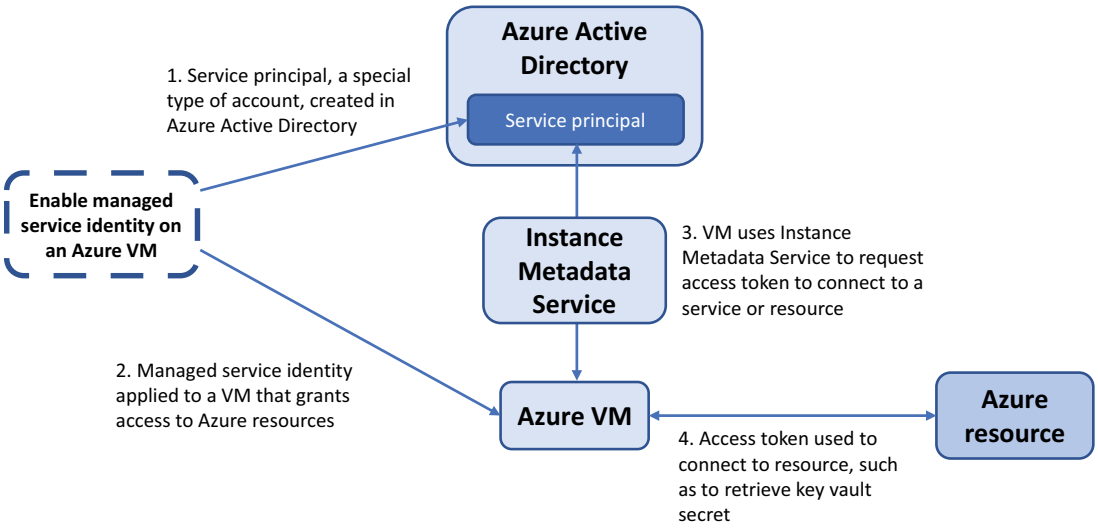


Figure 15.4 When you create a managed identity for a VM, a service principal is created in Azure Active Directory. This service principal is a special type of account that can be used for resources to authenticate themselves. Then this VM uses the Instance Metadata Service endpoint to makes requests for access to resources. The endpoint connects to Azure AD to request access tokens when the VM needs to request data from other services. When an access token is returned, it can be used to request access to Azure resources, such as a key vault.

A *managed identity* lets you create a special kind of account that can be used by an Azure resource, like a VM. If you've used a directory service such as Active Directory, a computer account is often used to identify and grant access to various network resources that a computer needs. You don't create and use regular user accounts for this type of authentication, which improves security: you can grant a restrictive set of permissions just to a computer rather than also worrying about user permissions and shared folder access, for example.

A managed identity is like a computer account, but it's stored in Azure Active Directory (Azure AD). The identity, called a *service principal*, is unique to each VM and can be used to assign permissions to other Azure resources, such as an Azure Storage account or key vault. The VM has permissions to access those resources, so you can script tasks (such as with Azure Automation, which we'll explore in chapter 18) that require no user intervention or prompts for usernames and passwords. The VMs authenticate themselves, and the Azure platform authorizes access to their assigned resources.

You can create two types of managed identities:

- *System-assigned*—This type of managed identity is applied directly to a resource, like a VM, and is used only by that resource. Each resource has its own unique identity when it comes to auditing or troubleshooting access. When the resource is deleted, the managed identity is deleted automatically.
- *User-assigned*—A separate Azure resource is created and managed for the specified managed identity. This managed identity can be shared across other resources to define access. When any resources that use the identity are deleted, the managed identity remains available for use.

Let's see how you can use a system-assigned managed identity to request the `databasepassword` secret from a key vault. Once the VM can retrieve the secret, the password can be used to install a MySQL database server automatically. With a key vault and MSIs, you can run a couple of commands to retrieve the secret from the key vault, run the MySQL Server installer, and automatically provide the secure password.

Azure Instance Metadata Service

A VM that's enabled with an managed identity uses a REST endpoint through the Instance Metadata Service (IMDS) to request an access token from Azure AD that it can then use to request data from Azure Key Vault. But what is the Instance Metadata Service?

IMDS is a REST endpoint that's accessible only internally to VMs. The endpoint is available at the nonroutable address of 169.254.169.254. A VM can make a request to the IMDS endpoint to retrieve information about itself, such as Azure region or resource group name. This ability allows the VM to understand how and where in the Azure platform it's running. The IMDS endpoint can be accessed from many languages, including Python, C#, Go, Java, and PowerShell.

For maintenance events, the IMDS endpoint can also be queried so that the VM becomes aware of a pending update or reboot event. Then any preupdate or reboot tasks that are required can be carried out. Because IMDS is a REST endpoint on a nonroutable IP address, there's no agent or extension for the VM to install, and there are no network security or routing concerns.

For managed-identity purposes, the IMDS endpoint is used to relay the request for an access token to Azure AD. This approach provides a secure way for VMs to request access without needing to talk to Azure AD directly.

www.dbooks.org

Try it now

To create a VM with an MSI, complete the following steps:

1. Create an Ubuntu VM; then provide your resource group, such as `azuremolchapter15`, and a name for the VM, such as `molvm`. A user account named azuremol is created, and the SSH keys that you've used in previous chapters are added to the VM:

```
az vm create \
  --resource-group azuremolchapter15 \
  --name molvm \
  --image ubuntults \
  --admin-username azuremol \
  --generate-ssh-keys
```

2. As a security best practice, you shouldn't allow accounts to access all the resources across your entire Azure subscription. Especially for managed identities, grant only the minimum amount of permissions needed.

   For this exercise, scope access to only your resource group, such as `azuremolchapter15`. You set the scope by querying for the ID of the resource group with `--query id`. Then this ID is assigned to a variable named `scope`:

```
scope=$(az group show --resource-group azuremolchapter15
➥--query id --output tsv)
```

3. Create a system-assigned managed identity for the VM with the reader role so that it can only read resources, not make changes to them. Scope the identity to the resource group. The variable you created in the previous step that contains the resource group ID is provided:

```
az vm identity assign \
  --resource-group azuremolchapter15 \
  --name molvm \
  --role reader \
  --scope $scope
```

4. Apply permissions on the Azure Key Vault that grants access to the service principal for the managed identity. You can do this through the portal under Access Policies for the Key Vault resource, or you can use the Azure CLI. Let's use the CLI to see how to get the information programmatically.

   First, get information on the Azure AD service principal for your managed identity. Filter on the `display-name` of the VM you created in step 3, such as `molvm`:

```
az ad sp list \
  --display-name molvm \
  --query [].servicePrincipalNames
```

The output is similar to the following condensed example. Don't worry too much about what these values mean; you don't need to work with them beyond assigning the initial permissions here. Again, you can use the Azure portal to avoid the CLI if you're uncomfortable.

Make a note of the first `servicePrincipalName`. This value is used to assign permissions on Azure resources such as your key vault and is needed in the next step:

```
[
  "887e9665-3c7d-4142-b9a3-c3b3346cd2e2",
  "https://identity.azure.net//
  ➥ihxXtwZEiAeNXU8eED2Ki6FXRPkklthh84S60CiqA4="
]
```

5 Now set the access policy on the key vault such that the service principal for your VM can read secrets, and enter your first `servicePrincipalName` from step 4:

```
az keyvault set-policy \
  --name azuremol \
  --secret-permissions get \
  --spn 887e9665-3c7d-4142-b9a3-c3b3346cd2e2
```

One point to make here is that when the managed identity was created and scoped to the resource group, that didn't mean the VM could do anything it wanted. First, the only role created for the identity was read permissions to resources. But you still had to assign permissions to the key vault itself. These layers of security and permissions give you fine-grained control over the exact resources each identity can access.

Now that you have access to a key vault, you probably want to know how to retrieve the secret, right?

15.3 Obtaining a secret from within a VM with managed identity

You've stored a secret in a key vault for a database password, and you have a VM with a managed identity that provides access to read that secret from the key vault. Now what? How do you retrieve the secret and use it? Figure 15.5 shows how a VM uses the IMDS to request access to a resource, such as a key vault. Let's go through the steps to see how the VM retrieves the secret.

Most use cases for Azure Key Vault wouldn't have a VM connecting and retrieving the secrets this way. Key Vault really shines when applications themselves, within the code, reach out to retrieve secrets. The application code would use the appropriate Azure SDK, such as Python, .Net, or Java. To avoid complexities of code abstracting what's happening, the following exercise uses a VM and some command-line work. As you work through this exercise, remember that this magic usually would happen within the application code.

www.dbooks.org

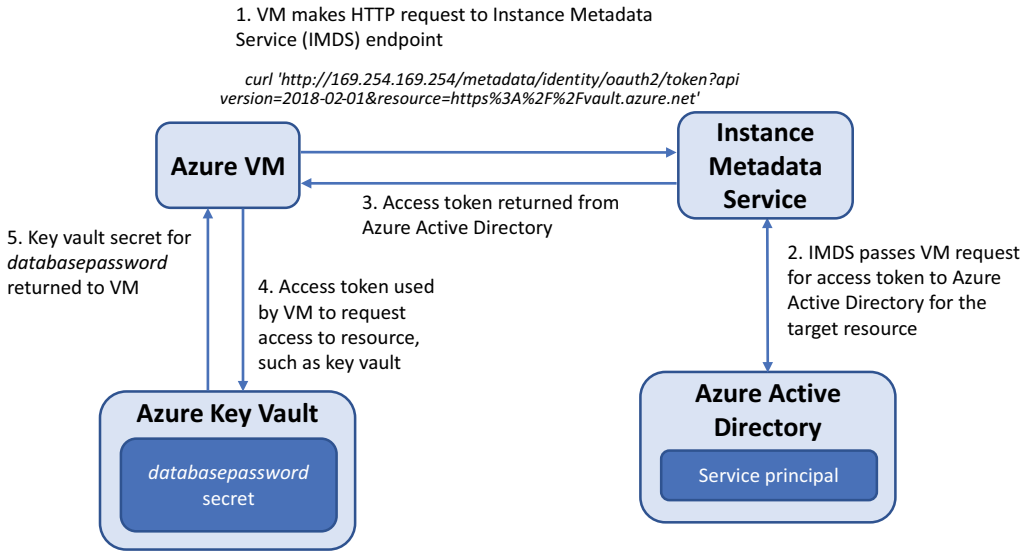


Figure 15.5 The VM uses the IMDS to request access to a key vault. The endpoint communicates with Azure AD to request an access token. The access token is returned to the VM, which is then used to request access from the key vault. If access is granted by the key vault, the secret for `databasepassword` is returned to the VM.

Try it now

To retrieve and use a secret on a VM with a managed identity, complete the following steps:

1. Get the public IP address of the VM you created in the previous exercise, such as `molvm`:

```
az vm show \
  --resource-group azuremolchapter15 \
  --name molvm \
  --show-details \
  --query [publicIps] \
  --output tsv
```

2. SSH to your VM, such as `ssh azuremol@publicIps`.
3. To access a key vault, you need an access token. This access token is requested from the IMDS. It's an HTTP request, and on a Linux VM you can use the `curl` program to make the request. The IMDS passes your request on to AAD:

```
curl 'http://169.254.169.254/metadata/identity/oauth2/token?
➥api-version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net'
➥-H Metadata:true
```

4 The output is a little hard to read because it looks like a jumble of text. It's in the JSON Web Token (JWT) format. To process the JSON output and make things more human-readable, install a JSON parser called `jq`:

```
sudo apt-get update && sudo apt-get -y install jq
```

5 Make your `curl` request again, but this time, view the output with `jq`:

```
curl 'http://169.254.169.254/metadata/identity/oauth2/token?
➥api-version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net'
➥-H Metadata:true --silent | jq
```

These first few steps show you how the requests are made and what the output looks like, as shown in figure 15.6. If you still log in to the VM and manually request an access token, what's the point of using a managed identity? You could just provide your own credentials. In production use, you'd likely use a script that runs on the VM to make the request for an access token automatically and then retrieve the secret from the key vault. Let's keep going to see how you automate this process and retrieve the secret.

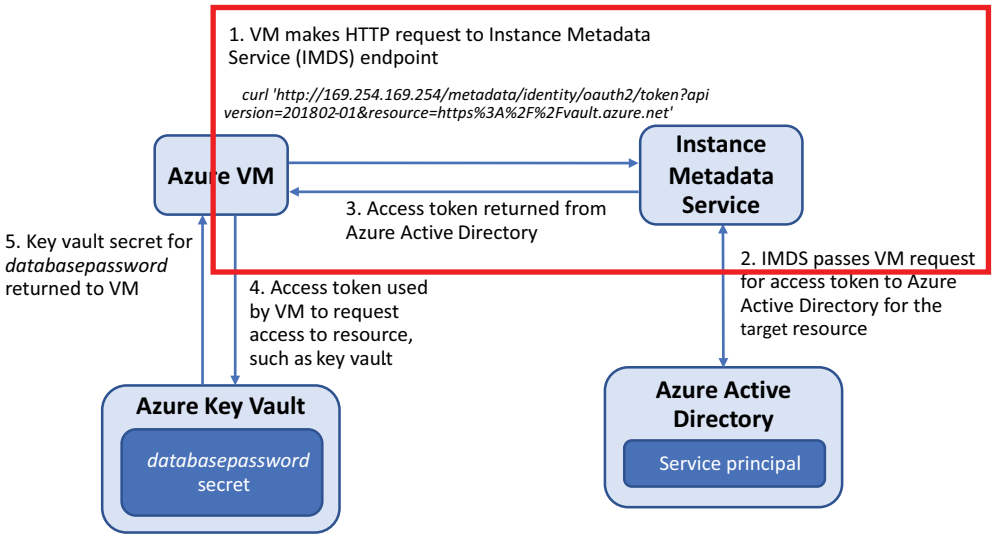


Figure 15.6 The `curl` request covers the first three steps of this diagram. The `curl` request is made, the endpoint communicates with Azure AD, and an access token is issued.

6 To make things easier—and if you were going to do all this in a script—you can use `jq` to process the curl response, extract only the access token, and set it as a variable named `access_token`:

```
access_token=$(curl
➥'http://169.254.169.254/metadata/identity/oauth2/token?
➥api-version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net'
➥-H Metadata:true --silent | jq -r '.access_token')
```

www.dbooks.org

7 As a manual step to help you understand what this looks like, view the `access_token` variable:

```
echo $access_token
```

8 Now the fun part! Use the access token to request your secret from the key vault. First, do this manually so you understand what happens.

9 Retrieve the secret with another `curl` request, and format the output with `jq`. Enter your own key vault name at the start of the https:// address:

```
curl https://azuremol.vault.azure.net/secrets/databasepassword?
➥api-version=2016-10-01 -H "Authorization: Bearer $access_token"
➥--silent | jq
```

The output is similar to the following, which shows the value of the password stored in the secret, along with some additional metadata about the secret that you don't need to worry about for now:

```
{
  "value": "SecureP@ssw0rd!",
  "contentType": "Database password",
  "id":
➥"https://azuremol.vault.azure.net/secrets/databasepassword/
➥87e79e35f57b41fdb882c367b5c1ffb3",
}
```

This `curl` request is the second part of the workflow, as shown in figure 15.7.

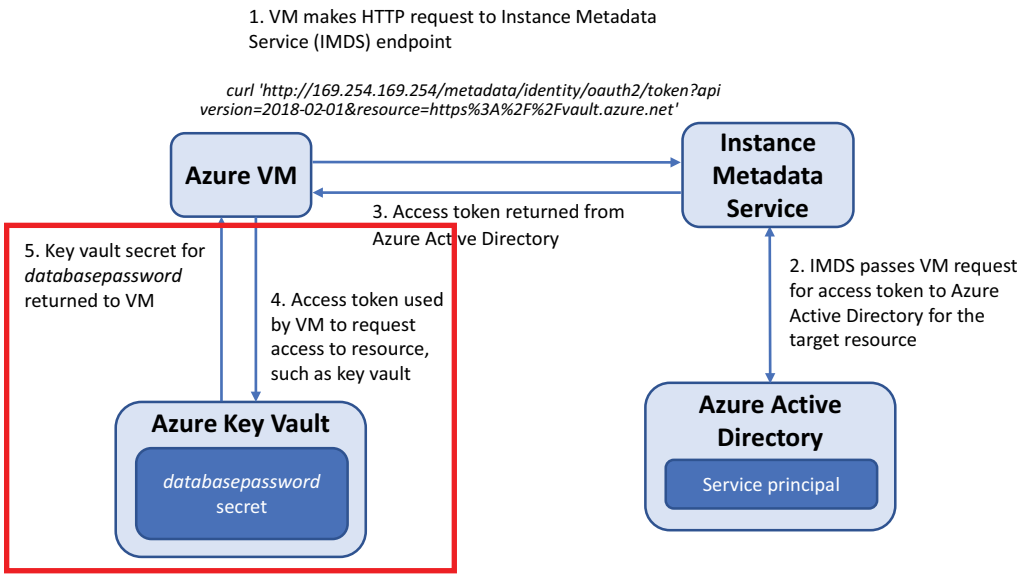


Figure 15.7 This second `curl` request covers the last two steps in the diagram. The access token is used to request the secret from the key vault. The JSON response is returned, which includes the value of the secret.

10 In the same way that you used a variable to store the access token, in a script you can assign the value of the secret to a variable as well. This time, use `jq` to process the response, extract only the value secret, and set it as a variable named `database_password`:

```
database_password=$(curl
➥https://azuremol.vault.azure.net/secrets/databasepassword?
➥api-version=2016-10-01 -H "Authorization: Bearer $access_token"
➥--silent | jq -r '.value')
```

11 Again, as a manual step to help you understand the process, view the contents of the `database_password` variable:

```
echo $database_password
```

I hope that you're following along! If you write an application in Python, ASP.NET, or Node.js, for example, the process will be similar as you make a request for the access token and then use the token to request a secret from a key vault. There are likely other libraries you could use in your code rather than the `jq` utility from the command line.

As a quick recap, all these steps can be condensed to two lines, as shown in the following listing.

Listing 15.1 Requesting an access token and then a secret from a key vault

```
access_token=$(curl
➥'http://169.254.169.254/metadata/identity/oauth2/token?
➥api-version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net'
➥-H Metadata:true --silent | jq -r '.access_token')
database_password=$(curl
➥https://azuremol.vault.azure.net/secrets/databasepassword?
➥api-version=2016-10-01 -H "Authorization: Bearer $access_token"
➥-silent | jq -r '.value')
```

Now what? The managed identity for your VM can retrieve a secret from a key vault. Let's see how you can use that managed identity to install and configure MySQL Server.

In Ubuntu, you can set configuration selections for package installers, such as MySQL Server. These configuration selections let you provide values such as usernames and passwords and have them used automatically at the relevant part of the install process. The manual prompts to provide a password, as you may have seen back in chapter 2, are gone.

12 Set the configuration selections for the MySQL Server passwords with the `database_password` variable you created in step 10:

```
sudo debconf-set-selections <<< "mysql-server mysql-server/root_password
➥password $database_password"
sudo debconf-set-selections <<< "mysql-server mysql-
➥server/root_password_again password $database_password"
```

www.dbooks.org

13 Install MySQL Server. There are no prompts because the password is provided by configuration selections:

```
sudo apt-get -y install mysql-server
```

14 Let's prove that all this worked! View the `database_password` variable so you can clearly see what your password should be:

```
echo $database_password
```

15 Log in to MySQL Server. When prompted for a password, enter the value of `database_password`, which is the value of the secret from the key vault:

```
mysql -u root -p
```

You're logged in to the MySQL Server, which confirms that the secret from the key vault was used to create the SQL server credentials!

16 Type `exit` twice to close out of the MySQL Server command prompt, and then close your SSH session to the VM.

This example is a basic one; you'd still need to secure the MySQL Server and provide additional credentials for applications to access databases or tables, for example. The advantage of using a secret from a key vault is that you guarantee all the passwords are the same. If you use virtual machine scale sets, for example, each VM instance can automatically request the secret and install MySQL Server so that it's ready to serve your application data. Those passwords are never defined in scripts, and no one needs to see what the passwords are. You could even generate passwords at random and rotate them as secrets in a key vault.

Storing passwords in a key vault is great, but can you use a key vault to store certificates and retrieve them automatically from your applications or VMs? Of course you can!

15.4 Creating and injecting certificates

Digital certificates are a common form of security and authentication in web services and applications. Certificates are issued by a certificate authority (CA), which is (we hope!) trusted by end users. The certificate allows users to verify that a website or application is indeed what it says it is. Every time you see a website with a web browser address that begins with https:// and has a padlock symbol, the traffic is encrypted and secured by a digital certificate.

Managing digital certificates can become a major management task. A common problem is how to store and grant access to certificates as services and applications need them. In the previous exercises, we examined how a key vault can be used to share secure secrets and keys with services and applications, but a key vault can do the same with certificates. As shown in figure 15.8, a key vault can be used to request, issue, and store certificates.

In production use, you should always use a trusted CA to issue your certificates. For internal use, you can issue self-signed certificates that you create yourself. These

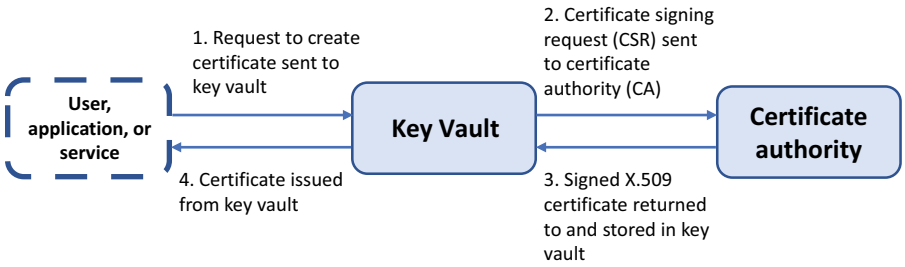


Figure 15.8 A user, application, or service can request a new certificate from a key vault. A certificate signing request (CSR) is sent by the key vault to an external third-party CA or a trusted internal CA. Azure Key Vault can also act as its own CA to generate self-signed certificates. Then the CA issues a signed X.509 certificate, which is stored in the key vault. Finally, the key vault returns the certificate to the original requestor.

self-signed certificates aren't trusted by other services and applications, so they typically generate a warning, but self-signed certificates let you get up and running quickly and make sure your code works as expected with encrypted traffic.

Azure Key Vault can generate self-signed certificates for you. Under the hood, Key Vault acts as its own CA to request, issue, and then store certificates. Let's use this ability to generate a self-signed certificate and see how to easily inject it into a VM. Then the certificate is used for a basic web server to show you how to quickly enable SSL to secure your web traffic.

Try it now

To create and inject a certificate into a VM, complete the following steps:

1. Create a self-signed certificate in Azure Key Vault, and enter a name, such as `molcert`. Policies are used to define properties such as expiration time periods, encryption strength, and certificate format. You can create different policies to suit the needs of your applications and services. For this exercise, use the default policy that creates a 2,048-bit certificate and is valid for one year:

```
az keyvault certificate create \
  --vault-name azuremol \
  --name molcert \
  --policy "$(az keyvault certificate get-default-policy)"
```

2. To see the certificate in action, create another VM, such as `molwinvm`. This time, create a Windows VM that uses Windows Server 2019, so you spread around the OS love and see that these Key Vault features aren't dependent on a specific OS! Provide your own admin username and password:

```
az vm create \
  --resource-group azuremolchapter15 \
```

www.dbooks.org

```
  --name molwinvm \
  --image win2019datacenter \
  --admin-username azuremol \
  --admin-password P@ssw0rd1234
```

3 You can automatically add the certificate to the VM straight from the Azure CLI. This approach doesn't rely on a managed identity; the Azure platform injects the certificate using the Windows Azure VM agent.

Add your certificate, such as `molcert`, to the VM you created in step 2, such as `molwinvm`:

```
az vm secret add \
  --resource-group azuremolchapter15 \
  --name molwinvm \
  --keyvault azuremol \
  --certificate molcert
```

4 Connect to the VM, and verify that the certificate was injected correctly. To connect to your VM, first get its public IP address:

```
az vm show \
  --resource-group azuremolchapter15 \
  --name molwinvm \
  --show-details \
  --query [publicIps] \
  --output tsv
```

Use a local Microsoft Remote Desktop connection client on your computer to connect to your VM. Use the credentials to connect to localhost\azuremol, not the default credentials of your local computer that your Remote Desktop client may try to use, as shown in figure 15.9.

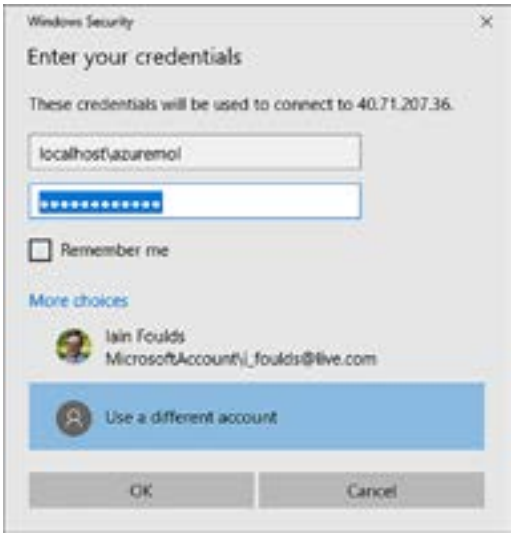


Figure 15.9 Your Remote Desktop client may try to use your default local computer credentials. Instead, select Use a Different Account, and then provide the localhost\azuremol credentials that you specified when you created the VM.

5 When you're logged in, select the Windows Start button, type `mmc`, and open the Microsoft Management Console.

6 Choose File > Add / Remove Snap-in, and then select the option to add the Certificates snap-in.

7 Choose to add certificates for the Computer account, select Next, and then select Finish.

8 Choose OK to close the Add / Remove Snap-in window.

9 Expand the Certificates (Local Computer) > Personal > Certificates folder. The certificate from Azure Key Vault that you injected into the VM is listed, such as CLIGetDefaultPolicy, as shown in figure 15.10.

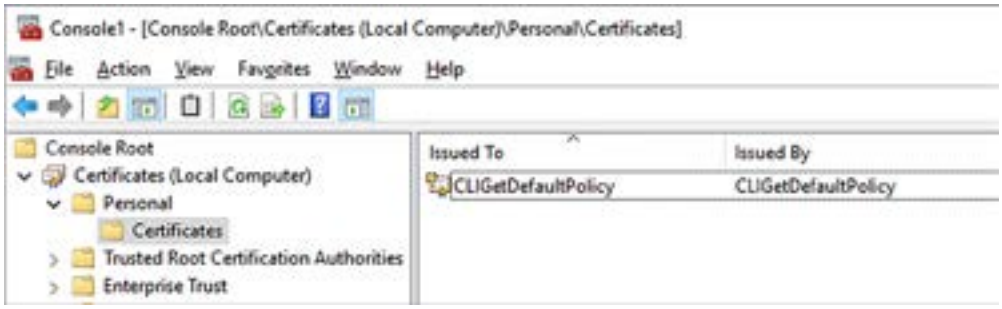


Figure 15.10 In the Microsoft Management Console, add the Certificates snap-in on the local computer. Expand the Personal > Certificates store to view installed certificates. The certificate injected from Key Vault is listed.

That's all there is to it! Create the certificate in Key Vault, and then add the certificate to the VM. The certificate is placed in the local certificate store of the computer, which allows any service or application to access it. On a Windows VM, the certificates are stored in the local certificate cache, as shown in this exercise. On Linux VMs, .prv and .crt files for the private and public parts of the certificate are stored in /var/lib/waagent/. You can move the certificates to wherever you need to for your application or service.

Certificates can be used for authentication between clients and servers, or between application components and services. A common example is for a web server to use an SSL certificate, which is what you'll do in the end-of-chapter lab.

15.5 Lab: Configuring a secure web server

In the last exercise, you injected a self-signed certificate from Azure Key Vault into a Windows VM. For this lab, install and configure the IIS web server to use the certificate, following this guidance:

1 Open PowerShell on your Windows VM, and install the IIS web server:

```
Add-WindowsFeature Web-Server -IncludeManagementTools
```

www.dbooks.org

2. Open Internet Information Server (IIS) Manager. You can do this from the Tools menu in Server Manager.
3. For Default Web Site, choose Edit Bindings.
4. Add an HTTPS binding on All Unassigned IP addresses on port 443.
5. Select the self-signed certificate you created and injected from Key Vault, typically named something like CLIGetDefaultPolicy.
6. Open a web browser on the VM, and enter `https://localhost`. You generated a self-signed certificate in Key Vault, so the web browser doesn't trust it.
7. Accept the warning to continue, and verify that the HTTPS binding works.
8. Back in the Azure Cloud Shell or portal, create an NSG rule for the VM on TCP port 443. Enter `https://yourpublicipaddress` in a web browser on your local computer. This is the experience your users would receive, with a warning about an untrusted self-signed certificate. For most use cases, remember to use a trusted internal or third-party CA to generate trusted certificates and store them in a key vault.

16 Azure Security Center and updates

Wouldn't it be great if Azure were smart enough to monitor all of your core application resources and alert you about any security concerns? Or what if your business has security policies already defined? (If you don't have any security policies, please stop right now and make a note to create some!) In the latter case, how do you ensure that your Azure deployments remain compliant? If you've ever gone through an IT security audit, you know how fun it can be to look over a list of misconfigurations applied to your environment, especially the basic security lapses that you know to avoid!

Azure Security Center provides a central location for security alerts and recommendations to be grouped for your review. You can define your own security policies and then let Azure monitor the state of your resources for compliance.

In this chapter, we'll discuss how Security Center can alert you to problems and provide steps to correct them, how you can use just-in-time VM access to control and audit remote connections, and how Update Management keeps your VMs up to date with the latest security patches automatically.

16.1 Azure Security Center

Throughout this book, we've discussed security-related topics such as how to create and configure network security groups (NSGs) to restrict access to VMs, and how to permit only encrypted traffic into Azure Storage accounts. For your own deployments beyond the exercises in this book, how do you know where to start, and how can you check that you applied all the security best practices? That's where Azure Security Center can help—by checking your environment for areas you may have missed.

www.dbooks.org

Azure Security Center scans your resources, recommends fixes, and helps remediate security concerns, as shown in figure 16.1. When you have only a couple of test VMs and a single virtual network in your Azure subscription, it may not seem that hard to keep track of what security restrictions you need to put in place. But as you scale up to tens, hundreds, or even thousands of VMs, manually keeping track of what security configurations need to be applied to each VM becomes unmanageable.

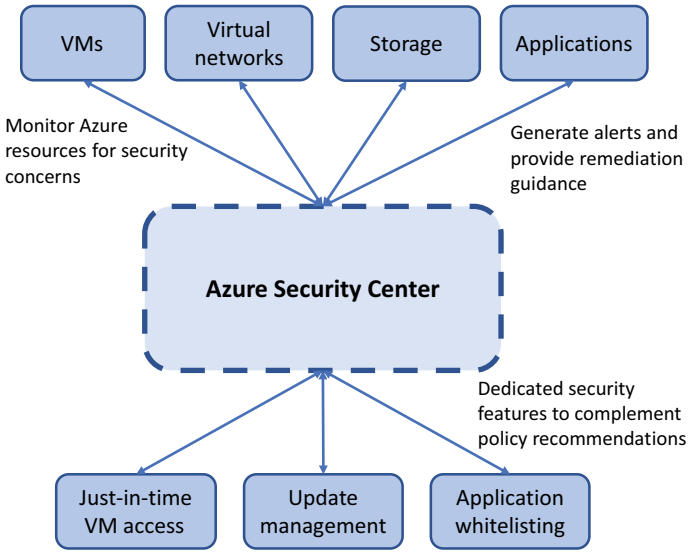


Figure 16.1 Azure Security Center monitors your Azure resources and uses defined security policies to alert you to potential threats and vulnerabilities. Recommendations and steps to remediate issues are provided. You can also use just-in-time VM access, monitor and apply security updates, and control whitelisted applications that can run on VMs

Security Center can also alert you to general best practices, such as if a VM doesn't have diagnostics enabled. Remember in chapter 12 when we looked at how to monitor and troubleshoot VMs? You need to install and configure the diagnostics agent *before* you have a problem. If you suspect a security breach, you may not be able to access the VM and review logs. But if you had configured the diagnostics extension to stream logs to Azure Storage, you could review what had occurred and (if all goes well) track down the source and extent of the problem.

Try it now

To get started with Azure Security Center, complete the following steps:

1. Open the Azure portal, and choose the Cloud Shell icon from the top menu.
2. Create a resource group; provide a name, such as `azuremolchapter16`; and provide a location, such as `eastus`:

```
az group create --name azuremolchapter16 --location eastus
```

3 Create a basic Linux VM so that Security Center has something to monitor and provide recommendations for:

```
az vm create \
  --resource-group azuremolchapter16 \
  --name azuremol \
  --image ubuntults \
  --admin-username azuremol \
  --generate-ssh-keys
```

4 When the VM is deployed, close Cloud Shell.

5 In the Azure portal, select Security Center in the list of services at left. The first time the dashboard opens, it takes a few seconds to prepare all the available components; see figure 16.2.

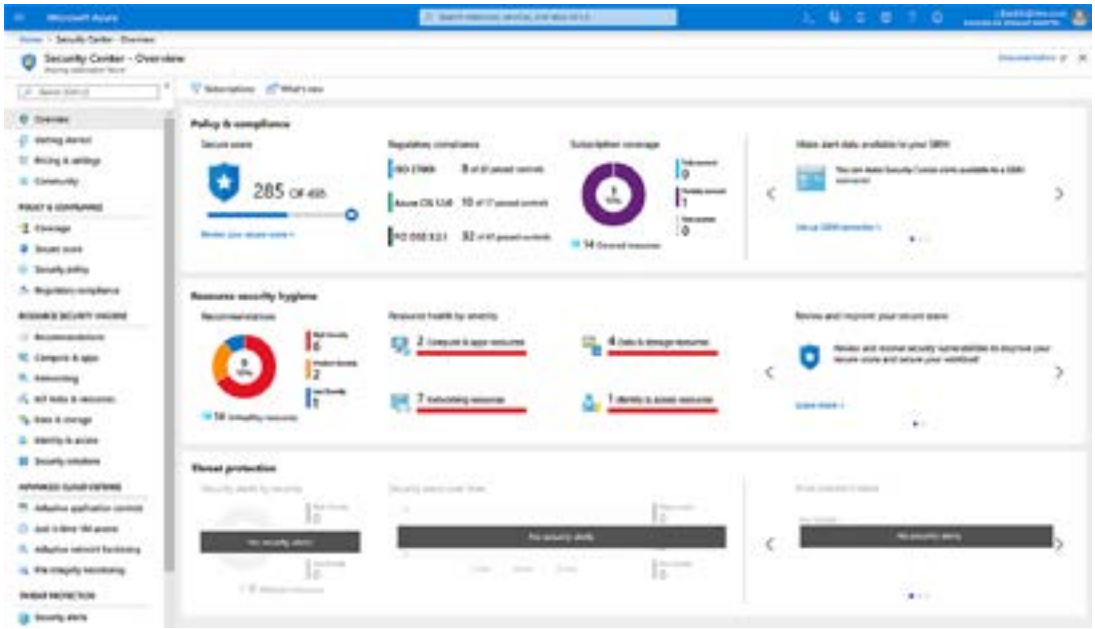


Figure 16.2 The Azure Security Center Overview window provides a list of recommendations, alerts, and events. You can select a core resource type such as Compute or Networking to view a list of security items specific to those resources.

Security Center looks at how resources such as VMs, NSG rules, and storage are deployed. Built-in security baselines are used to identify problems and provide recommendations. The virtual network deployed with your VM generates warnings, for example, as shown in figure 16.3. You can, and should, implement your own security policies that tell Azure how you want to restrict access or what needs to be done to comply with business mandates. Then, as you create or update resources, Azure

www.dbooks.org

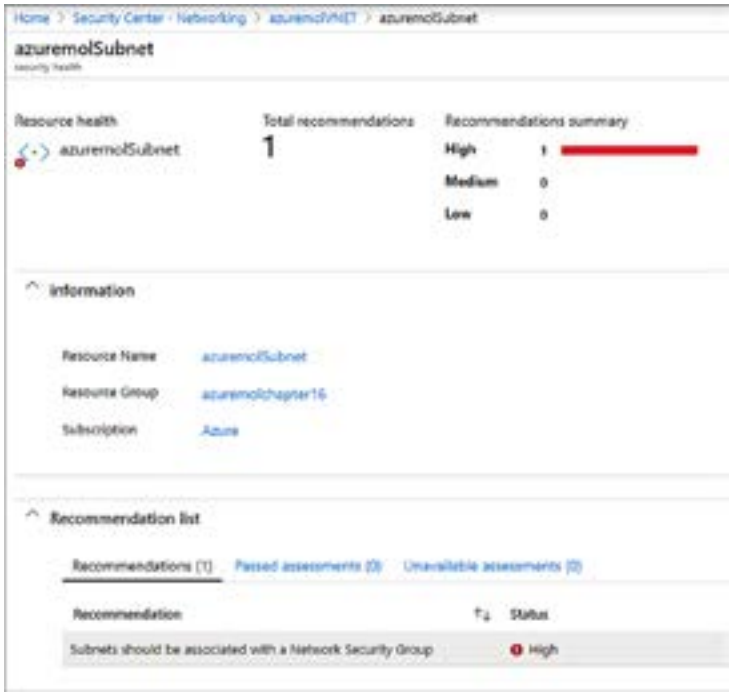


Figure 16.3 The virtual network for your VM already triggers security warnings. In this example, it warns that a network security group should be associated with the subnet.

continually monitors for deviations from these policies and alerts you about what steps need to be taken to remediate the security issues. You'll use the default Azure security policies in this chapter, but think of any particular security configurations that you may want to apply to your VMs and how they could be defined in your own custom policies.

6 Choose Compute & apps on the left menu in the Security Center window; then choose VMs and Computers.

7 Select the VM you created in step 3. Even though you just created this VM and used default values from the Azure CLI, some security warnings are shown.

Explore some of these recommendations. As you select each recommendation, some just give you more information; others guide you through remediation. These aren't hard-and-fast rules; they're recommendations and best practices. In your own environment, some of these may not make sense. But they're a good starting point to know what things you should be doing to secure resources as you create them in Azure.

16.2 Just-in-time access

In section 16.1, you learned how Security Center suggests that you limit the scope of inbound remote connectivity. You could provide an IP range to limit traffic, but ideally, you open inbound connectivity only when it's needed. That way, the VM is completely closed for remote connections and is accessible only for a short time when

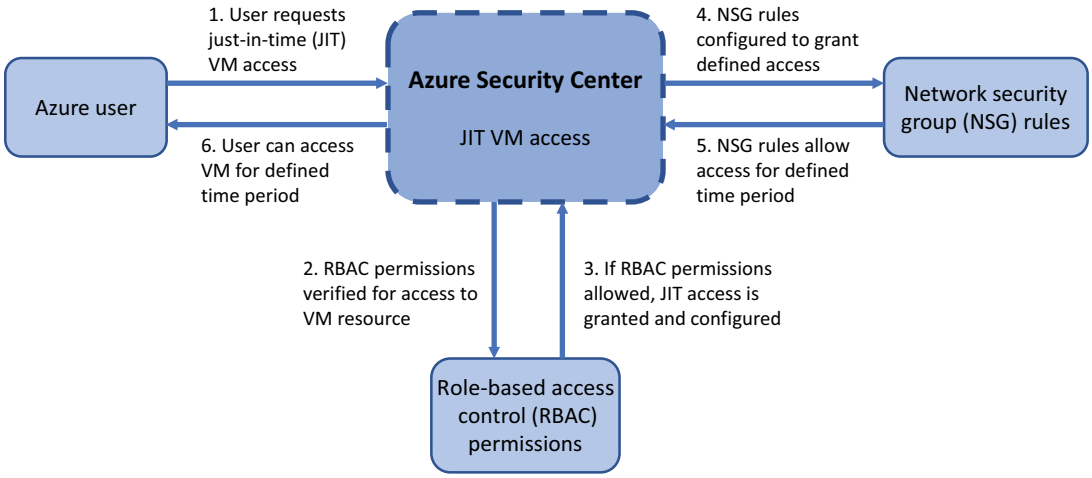


Figure 6.4 With JIT VM access, NSG rules are configured to deny remote connections to a VM. RBAC permissions are used to verify permissions when a user requests access to a VM. These requests are audited, and if the request is granted, the NSG rules are updated to allow traffic from a given IP range for a defined period. The user can access the VM only during this time. When the time has expired, the NSG rules automatically revert to a deny state.

needed. And yes, you should still limit that brief window of connectivity to a specific IP range! That's where just-in-time (JIT) VM access is useful, as shown in figure 16.4.

With JIT access, Security Center dynamically adjusts access restrictions on a VM. When enabled, NSG rules are created that deny all remote connection traffic. Then a user can request access to a VM only when needed. In combination with role-based access control (discussed in chapter 6), Security Center determines whether a user has rights to access a VM when they request a connection. If the user does have permissions, Security Center updates the relevant NSG rules to allow incoming traffic. These rules are applied only in a specific time window. When that time is up, the rules are reverted, and the VM becomes closed to remote connections again. If you have an active connection to a VM, you aren't automatically disconnected when the time expires. You can finish your maintenance or troubleshooting work and disconnect when ready, but you won't be able to start a new connection unless you request JIT access again.

Drinking from a fire hydrant

We haven't really looked at Azure Firewall, but it's a virtual network resource that's a little more similar to an on-premises physical firewall than to NSGs by themselves. If you need more flexibility and control of traffic, Azure Firewall is a great option, though a cost is associated with it.

Without getting too deep into Azure Firewall, I want to note that Azure Security Center can also integrate with Azure Firewall to open and close the required rules. If you

www.dbooks.org

use Azure Firewall to protect VM traffic on virtual networks, not just NSGs, you can still use the automated rules management of JIT VM access.

To learn more about Azure Firewall, see the docs at https://docs.microsoft.com/azure/firewall/overview.

When would you use JIT in your fictional pizza store? Think about any VMs that would run your web application, order system, or business logic applications. Would you want those to be connected to the internet and available for people to access all the time? I hope not! There are valid reasons for remote access with SSH or RDP, but always try to minimize how long that access is available. Even if you have NSG rules that restrict access to certain IP ranges, JIT adds another layer of protection in terms of what Azure users can access and then creates an easier audit trail on which Security Center can provide reports.

Try it now

To enable JIT VM access, complete the following steps:

1. Open the Azure portal, and choose Security Center from the menu at left.
2. Under Advanced Cloud Defense, select Just in Time VM access.
3. If prompted, choose the option Try Just in Time VM access or Upgrade to Standard Tier of Security Center. This free trial lasts 60 days and shouldn't extend automatically. It overlaps with your free Azure account and won't cost you any money to use. Select the option Apply Standard Plan, and wait a few moments for it to be enabled. When it's enabled, you may need to close and reopen the Azure portal before you can complete the following steps.
4. Select Just in Time VM Access in the Security Center window again. When your standard tier account is enabled, you can view a list of VMs to use.
5. Select your VM, and then choose Request Access, as shown in figure 16.5.



Figure 16.5 Select a VM from the Recommended options, and then choose to Enable JIT on 1 VMs. State currently shows that this VM is Open for all remote access, which flags the severity of the security concern as High.

By default, JIT defines rules that can open ports for SSH (port 22), RDP (port 3389), and PowerShell remoting (ports 5985 and 5986) for a period of three hours.

6 For this exercise, choose to enable SSH from your own IP. As a best practice for production use, enter a justification to keep track of why access is being requested. Leave the defaults, and choose Open Ports, as shown in figure 16.6.



Figure 16.6 When you enable JIT, you can change the default rules to be allowed, the allowed source IPs, and a maximum request time in hours. These JIT rules allow granular control of what's permitted to allow only the bare minimum of connectivity.

7 With JIT enabled, browse to your resource group, and select your VM.

8 Choose Networking to view the assigned virtual network configuration for the VM. The list of assigned NSG rules is displayed, as in figure 16.7.

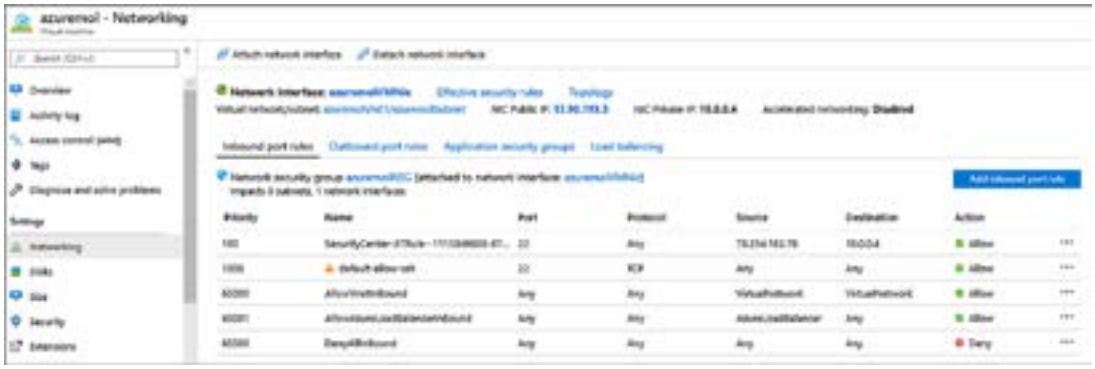


Figure 16.7 The JIT rules are created with the lowest priority. These priorities make sure that the JIT rules take precedence over any later rules applied at the subnet level.

www.dbooks.org

The JIT rules are shown at the top of the list because they have the lowest priority. Traffic is allowed to the IP address of the VM, but only from your own IP address. This is what JIT configured. What may seem odd here is that a default-allow-ssh rule still exists and permits all traffic. Think back to chapter 5, when we discussed NSGs. Can you tell what's happening here?

JIT applies only to the VM. In the JIT rule, Destination shows the IP address of the VM. In the example shown in figure 16.7, that's 10.0.0.4. Traffic is allowed. But the actual NSG rule is applied to the entire subnet. The default-allow-ssh rule applies at the subnet level and allows traffic from Any source and to Any destination.

NSG rules are processed in order of priority, from low to high. As discussed in chapter 5, a Deny action always takes effect, regardless of any additional rules. Even if you changed that default-allow-ssh rule to deny traffic, the JIT rule would still allow access to the specific VM and from the defined source IP address.

Take care with this layering of NSG rules. Ideally, you'd remove the default-allow-ssh rule and then allow access only as needed with JIT. In this approach, SSH is denied by the final DenyAllInbound rule. When you need to connect to a VM, use JIT to request access, which automatically creates a rule to allow SSH scoped to your IP address for a defined period.

The NSG rule is deleted automatically after the specified time period has elapsed. By default, JIT rules are applied for three hours. After that time, the VM returns to a more secure state, and you need to request access to the VM again.

This JIT process controls who can request, and be granted, access to the VM. But just because a person can successfully request access to a VM doesn't mean they have permissions to log on to that VM. All that happens in Azure is that the defined NSG rules are updated. Security Center and JIT can't add, remove, or update access credentials on the VM.

All JIT requests are also logged. In Security Center, select the Just in Time VM Access option, and choose your rule. On the right, select the … menu option, and then choose Activity Log. This activity log helps you audit who requested access to a VM in the event of a problem.

JIT VM access is one way that Security Center and Azure help keep your VMs secure. Controlling access to the VMs is a big part of security. But what about the applications, libraries, and services running on the VMs? That's where you need to ensure that all the latest security updates are applied to your VMs in a timely manner.

16.3 Azure Update Management

One area that Azure Security Center can report on is the status of any OS updates required by the VM. In your pizza store, you should try to install the latest security and application patches. You don't want to run any systems that have a known vulnerability or attack area, so a way to automate the updates of those systems and track compliance improves your security. When you work with applications that involve customer data and payment information, don't run systems without the latest patches installed.

And remember to plan for a test environment that lets you safely apply security patches and validate that they don't cause problems before you apply them to production systems!

An Update Management feature is built into Azure VMs and can scan, report, and remediate OS updates. What's great about this solution is that it works across both Windows and Linux, and even within Linux, across different distributions such as Ubuntu, Red Hat, and SUSE. Figure 16.8 shows how Update Management monitors and can install required updates.

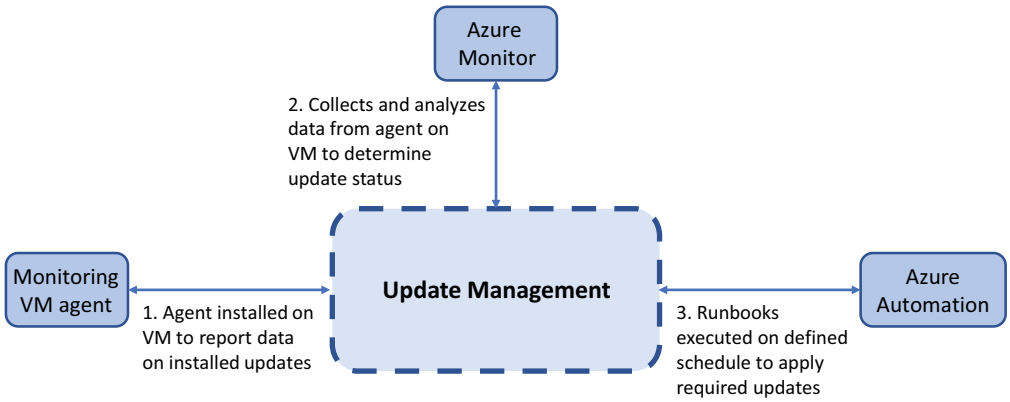


Figure 16.8 Update Management installs a VM agent that collects information on the installed updates on each VM. This data is analyzed by Azure Monitor and reported back to the Azure platform. The list of required updates can be scheduled for automatic install through Azure Automation runbooks.

It takes a few minutes for the VM to prepare itself and report back on its update status, so let's set up your VM and then see what goes on behind the scenes.

Try it now

To configure your VM for Update Management, complete the following steps:

1. Open the Azure portal, and choose Resource Groups from the menu at left.
2. Select your resource group, such as `azuremolchapter16`, and then select your VM, such as `azuremol`.
3. Under Operations, select Update Management.
4. Accept the default option for Location and the option to create a Log Analytics workspace and Automation Account. We'll examine these components in the remainder of this section.
5. To turn on update management for the VM, select Enable.

www.dbooks.org

You return to the Update Management Overview window, but it takes a few minutes to configure the VM and report back on its status. Continue reading, and let the process continue.

Let's look a little more at what happens to make this Update Management solution work.

16.3.1 Combined Azure management services

If you've worked with any on-premises Microsoft technologies, you may have come across the System Center suite. System Center consists of multiple components such as Configuration Manager, Operations Manager, Orchestrator, and Data Protection Manager. There are a couple of other parts, but those core components provide a way to do the following:

- Define configurations and desired state
- Install applications and updates
- Report on health and security
- Automate deployments of large services and applications
- Back up and replicate data

As businesses have moved to cloud computing over the past few years, those more traditional on-premises System Center components are replaced by Azure services that can work in a hybrid environment. We looked at two components in earlier chapters, even if you didn't realize it:

- *Azure Backup* provides a way to back up VMs or individual files, define retention policies, and restore data.
- *Azure Site Recovery* allows you to replicate VMs to different geographic regions in the event of a natural disaster or prolonged outage.

Both Azure Backup and Site Recovery helped you protect your data in chapter 13. Now you'll use some additional services with Update Management:

- *Log Analytics workspaces* collect information from various sources or agents, and allow you to define policies and queries to alert you to conditions that may occur. These queries and alerts can help you track the update status of a VM or notify you of configuration or security issues.
- *Azure Monitor* details and reports information based on processing carried out in Log Analytics workspaces. Azure Monitor provides a centralized way to view alerts, query log data, and generate notifications across all your Azure resources.
- *Azure Automation* allows you to build runbooks that execute commands or entire scripts. Runbooks can be large, complex deployments and can call multiple othwoks. We'll look at Azure Automation in depth in chapter 18.

The integration of these components is shown in figure 16.9.

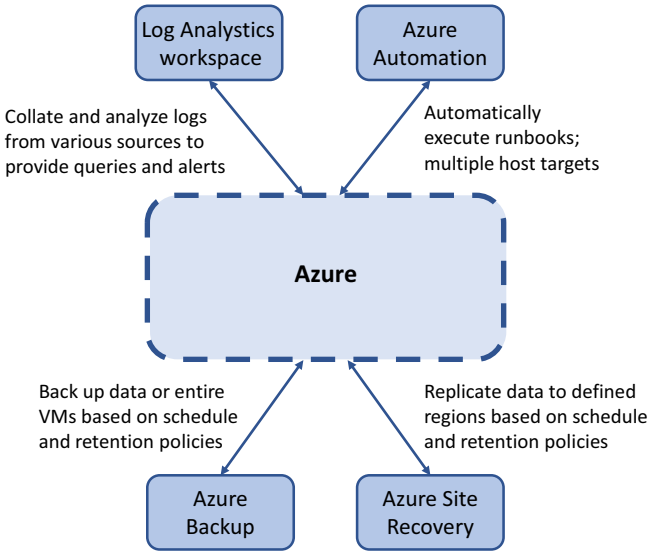


Figure 16.9 Multiple Azure services work together to provide management and configuration features across your entire application environment. The services that use these components aren't limited to Azure VMs or resources and can work across other cloud providers or on-premises systems when they're appropriately configured.

Both Log Analytics workspaces and Azure Automation are powerful components and could easily fill entire chapters of a book by themselves. With only a handful of VMs to manage, you may find it easy to overlook the need for a centralized log repository for querying and alerting, or a way to automate configurations and deployments across VMs. If you haven't already been making a list of Azure components to follow up on when you finish this book, start one, and add both of these components to that list!

One thing to understand is that in Azure, multiple services and components can interact with and complement one another. In the same way that Azure VMs and Azure virtual networks are individual services, both services also complement, or even rely on, each other. Azure Backup and the Azure diagnostics extension are great individual components, but they really shine if Log Analytics workspaces and Azure Monitor are used to monitor their status and collate any generated events or warnings. I hope that you've begun to identify some of these related components and see how Azure services build on one another. Now that we're into these final few chapters and looking at security and monitoring options, the goal is to ensure that the applications you run in Azure are healthy and stable.

www.dbooks.org

This little thing called "Identity"

Thinking about services that complement each other, one large (and I mean *large*!) part of Azure that we've touched on only lightly is Azure Active Directory (Azure AD). Identity is central to everything in Azure, and Azure AD provides some of the security features we examined in chapter 6 with the Azure Resource Manager deployment model. The ability to use RBAC to limit what actions can be performed on a resource by certain users or groups is tied into a central identity solution. Even being able to sign in to the Azure portal or Azure CLI is driven by Azure AD.

This book doesn't cover Azure AD, because what it provides is broad and quite different from Azure IaaS and PaaS services such as VMs, scale sets, and web apps. There may be some overlap in the topics' audience, but most developers would have a different goal for what they wanted to learn about Azure AD, compared with an application manager or an IT pro who deploys the infrastructure.

Depending on your Azure account, you may be limited in what you can do with Azure AD. When you sign up for a free Azure trial account, a default Azure AD instance is created for you. You're the primary account in that directory, and you have full administrator rights. If you log in to Azure with an account from your business or educational institution, there's a good chance that you have few to no administrative rights. So even if we could agree on a couple of topics to focus on, you might not be able to carry out any of the exercises directly. And I really don't recommend that you go digging around in an actual Azure AD environment to learn how things work!

But Azure AD is another of those central services in Azure that binds together many other services and components. Cloud computing doesn't magically make things easier or break down operational silos; you still need the skills to work with different teams and stakeholders. I hope that throughout these chapters, you've picked up the core skills for these Azure services, which will help you understand how to build large, redundant applications and converse at a better level and with more awareness of what other teams may face.

16.3.2 Reviewing and applying updates

It can take some time for the VM agent to perform the first scan and report back on the status of applied updates. The list of installed components must also be cross-referenced with the list of available updates for a given OS and version. If your VM hasn't finished and reported back on its status, keep reading, and check back in a few minutes. When it's ready, the overview looks like figure 16.10. Be patient; it can take 10 to 15 minutes for the agent readiness to show as Ready and let you schedule updates for installation.

A list of required updates is great, but what about a way to install them? That's where Azure Automation steps in! When you enabled Update Management, several Azure Automation runbooks were created that automatically handle the process to apply the required updates.

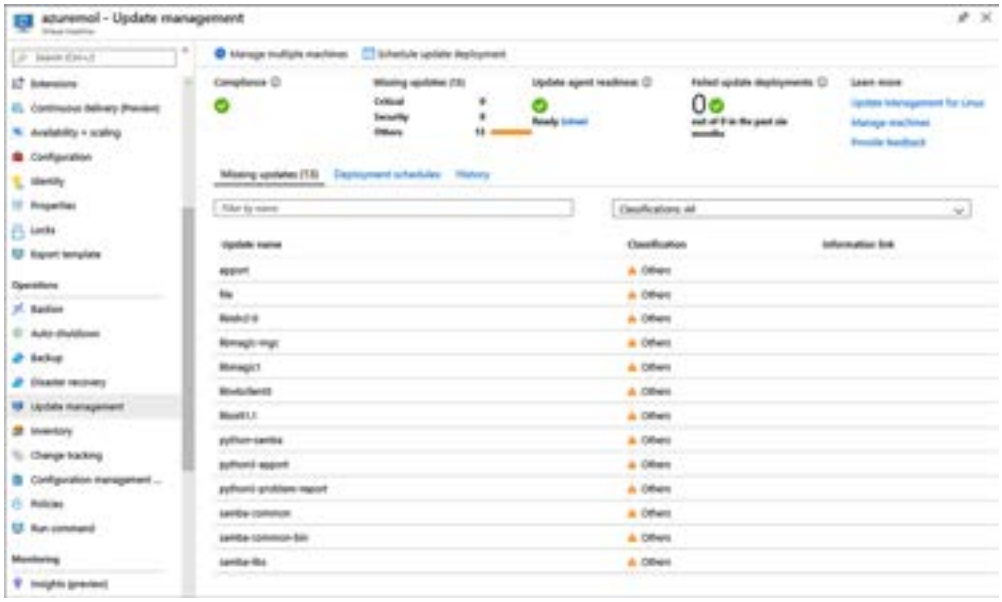


Figure 16.10 When the VM agent has scanned for compliance, a list of available updates is provided. Depending on the OS and version, Update Management may be able to work with the Log Analytics workspace and Azure Monitor to classify the updates based on severity or to provide links to the relevant update hotfix pages.

Try it now

If you're lucky (or unlucky), your VM may report that no updates are required. VM images are frequently updated in Azure, and if you deploy a VM soon after the latest image was built, all the required updates are already installed. If so, read through these steps so that you understand what's required when your VMs do need updating!

To apply the required updates for your VM, complete the following steps:

1. In the Update Management section of your VM, select Schedule Update Deployment.
2. Enter a name for the update deployment, such as `azuremolupdates`, and then review the Update Classifications. You can control which sets of updates are applied. For now, leave all the default options set.
3. Updates to Exclude lets you specify specific updates that you don't want to install. If you know your application requires a specific version of a package or library, you can make sure an updated package isn't installed that breaks things. Review the available options, but there isn't anything to change in this exercise.

www.dbooks.org

4 Select Schedule Settings, and then choose a time for the updates to be applied from the calendar and time options. The start time must be at least five minutes ahead of the current time to give the Azure platform a few moments to process and schedule your runbook in Azure Automation.

5 When you're ready, select OK.

6 If certain applications and services need to pause or shut down before updates are applied and start up again when the updates are finished, choose Pre-scripts + Post-scripts. Separate automation tasks can be configured to carry out actions on the VMs before and after the updates are applied.

7 Maintenance Window (minutes) defines how long the update process can run before the VM needs to be back in operation. This window prevents long-running update processes that may cause a VM to be unavailable for hours at a time. You may want to make the maintenance window shorter or longer, depending on any service-level agreements for the applications that run on those VMs, or the number and size of the updates required. Accept the default value, and then select Create.

8 Back in the Update Management window, select Deployment Schedules. The updates are listed as scheduled to install at the date and time you select, as shown in figure 10.86.11.

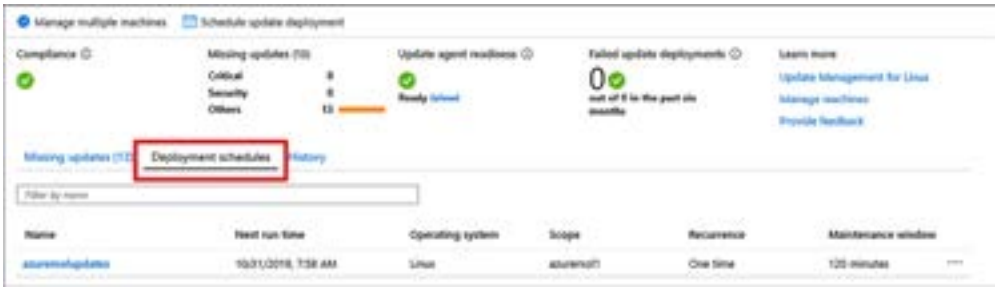


Figure 16.11 The list of scheduled deployment tasks is shown. If desired, you can delete a given task; otherwise, the updates are applied automatically at the defined time.

9 At the top of the Update Management window, select Manage Multiple Machines. The window switches to the Azure Automation account that was created when Update Management was enabled for the VM. Don't worry too much for now about what the runbooks do. There's nothing for you to customize, and we'll examine Azure Automation in chapter 18.

Note that you can choose Add Azure VM or Add Non-Azure machine, as shown in figure 16.12. This ability highlights a single approach to managing updates across your entire application environment, not just for Azure VMs.

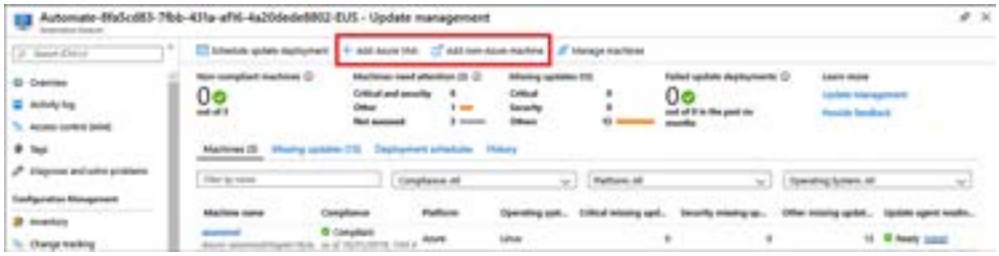


Figure 16.12 In the Azure Automation account, you can manage multiple computers and view the status or apply updates. Both Azure VMs and non-Azure computers can be monitored and controlled by the same Azure Automation account. Behind the scenes, Azure can integrate with other providers to install agents on computers in a hybrid environment. This integration allows a single dashboard and management platform to handle your update needs.

10 Go back to the Update Management window for your VM, and select the History tab. When your update deployment starts, its status is displayed. Remember that you scheduled the job to run a few minutes in the future, so it doesn't show up right away.

11 Select the schedule to see the status and output, as shown in figure 16.13.

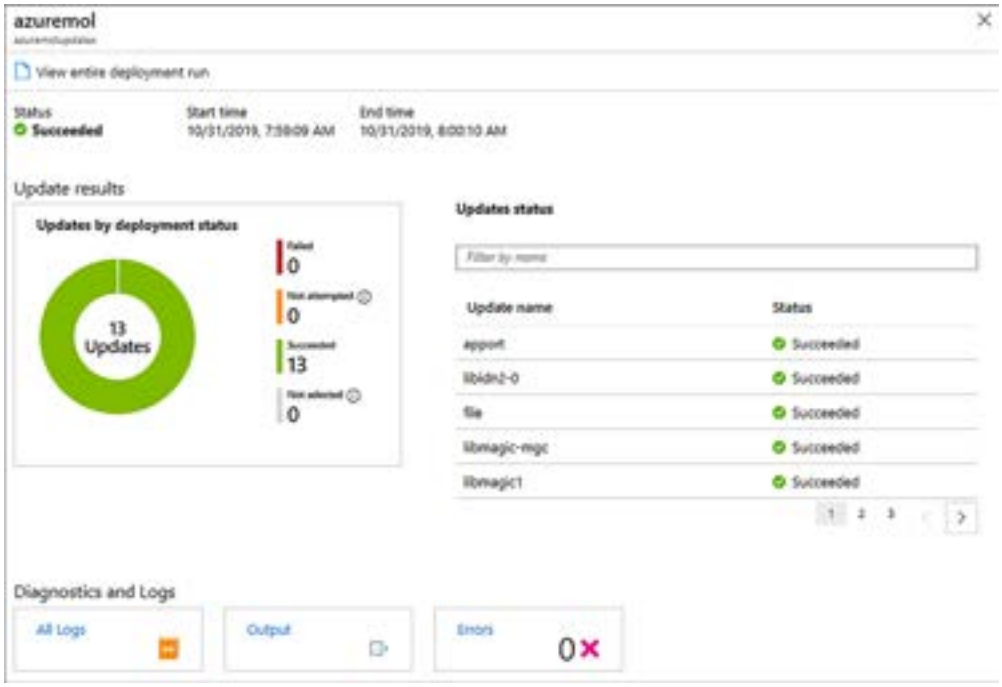


Figure 16.13 You can monitor the status of running Azure Automation jobs in the portal. To review or troubleshoot tasks, you can click a job to view any output and generated logs.

www.dbooks.org

12 When the update deployment has finished, browse back to your resource group, select your VM, and then choose Update Management. It may take a few minutes for the agent to update itself and report back through a Log Analytics workspace that the updates have been applied; then the dashboard should show that the VM is up to date, and no additional updates are required.

This chapter has been a whirlwind tour of Security Center and associated components such as JIT VM access and Update Management. The goal is for you to start to think beyond just how to deploy and run a VM or web app, and instead plan for the wider application management that goes with it. Cloud computing doesn't change the need for security policies; there's arguably a greater need for resources to be secured. Let Azure features such as Security Center guide you through what needs to be done, and use built-in tools such as Update Management and Azure Automation to keep things secure at all times.

16.4 Lab: Enabling JIT and updates for a Windows VM

This chapter covered a few components that may have taken some time to enable themselves and report back on their expected status. This lab is optional; it's designed to show that there's nothing OS-specific about any of these features. If you don't have time, or if you feel that you understand how to apply these features to a Windows VM, feel free to skip this lab. Otherwise, try the following tasks to gain some additional practice with Security Center and Update Management. Practice makes perfect, right?

1. Create a Windows Server VM of your choice in the same resource group you used for the previous exercises, such as `azuremolchapter16`.
2. View the NSG rules for the VM/subnet, and delete any default rules that permit RDP on TCP port 3389.
3. Use your local Remote Desktop Connection client to verify that RDP connections are blocked.
4. Request JIT access, review the NSG rules again, and confirm that now you can RDP to your VM.
5. Enable Update Management on your Windows VM. This time, you should be able to use the existing Log Analytics workspace and Azure Automation accounts.
6. Let the monitoring agent report back on required updates, and then schedule the updates to be applied through Azure Automation.

www.dbooks.org

Part 4

The cool stuff

Now for the really cool stuff! In these final few chapters, you'll learn about some up-and-coming technologies that you can use in Azure, such as artificial intelligence, machine learning, containers, Kubernetes, and the Internet of Things. You may not be using these services right now, but with the current trends in computing, you probably will be soon. These services are some of the most exciting technologies to work with. Although the book moves pretty quickly to cover these topics over your lunch break, this part is a great way to wrap things up and show you the possibilities of what you can build in Azure.

www.dbooks.org

17 Machine learning and artificial intelligence

Let's hope that we won't end up in a world where films like *The Terminator* and *The Matrix* come true. In those movies, the rise of artificial intelligence (AI) almost causes the downfall of humanity as machines fight to take control of their surroundings. One cause for concern in computing right now is how the development of AI is mostly done by large, private companies, with little or no regulation and central oversight. That's not at all to say that AI is a bad thing! Digital assistants on smartphones can help with many day-to-day tasks. Machine learning (ML) in navigation apps can monitor the user's daily drive to suggest alternate routes based on road or weather conditions. Home-heating controls can adjust automatically based on the outside temperature, time of day, and time of year (such as summer or winter).

As you begin this final part of the book, you'll learn about the Azure services for machine learning and artificial intelligence. In one chapter. On your lunch break. Let's set some realistic expectations: you're not going to become an expert in ML or AI in the next 45 minutes! If you eat your sandwich quickly, you may learn enough about the many services that Azure offers to understand how to integrate some ML and AI services into your applications. Many of the Azure ML and AI services expect at least some prior experience in data algorithms, programming languages, batch processing, or language understanding, so don't expect to become an expert in the next hour!

In this chapter, let's go on a whirlwind tour of some of the Azure cognitive services that provide ML and AI features. You'll learn how to use these services to perform basic machine learning on data models; then you'll use a little of the Azure Web Apps service and the Microsoft Bot Framework to apply some of the AI services that can run a pizza-store bot for customers to order pizza.

17.1 Overview and relationship of AI and ML

Hold on tight, because we're about to go from 0 to 600 mph in just a few pages! AI and ML often overlap as you build applications in Azure. Let's explore what each is, and then worry about how they work together.

17.1.1 Artificial intelligence

AI allows computers to complete tasks with some amount of flexibility and awareness, and to adjust their decisions based on external factors or without the need for human interaction. The goal usually isn't to build a completely autonomous system that can evolve and develop thoughts for itself, but to use a set of data models and algorithms to help guide the decision-making process.

Common AI on personal computers and smartphones includes Siri, Cortana, and Google Assistant. As shown in figure 17.1, these AI resources allow you to communicate, often via voice commands, to ask for directions, set reminders, search the web, and more.

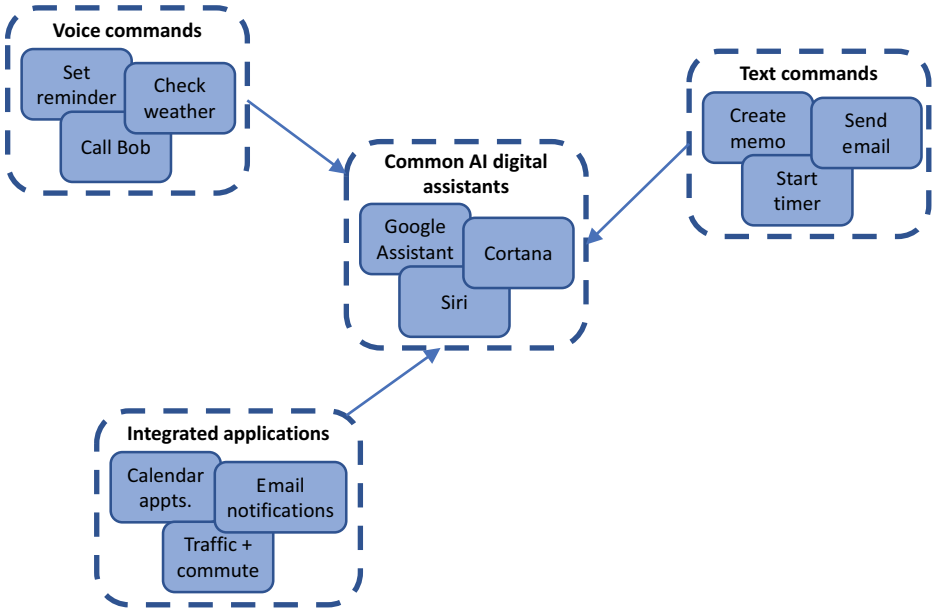


Figure 17.1 A common use of AI in everyday life is digital assistants such as Cortana, Siri, and Google Assistant. You can use voice or text commands to interact with them, and they can monitor your daily calendar and commute conditions to warn you about traffic problems.

Digital assistants like these typically don't involve a large amount of what you may consider *intelligence*. They listen and respond to input you provide. But those inputs can vary and may not always be specific commands. Think about how a digital assistant lets you set a reminder. You could use one of the following phrases:

www.dbooks.org

- "Remind me at 5 to pick up milk."
- "Tell me to pick up milk on the way home."
- "I need to get milk when I'm at the store."

If you developed a traditional application, you'd need to write code that could handle all the possible variations of how a user might provide instructions. You could build regular expressions to help catch some of the variations, but what happens when the user comes up with a phrase that you didn't program? Or what if they interact via text and have a typo in their request that you didn't anticipate? These types of interactions are a great fit for AI. As shown in figure 17.2, the application is programmed for several common phrases and is then able to make an educated guess based on what it "thinks" the user is asking for.

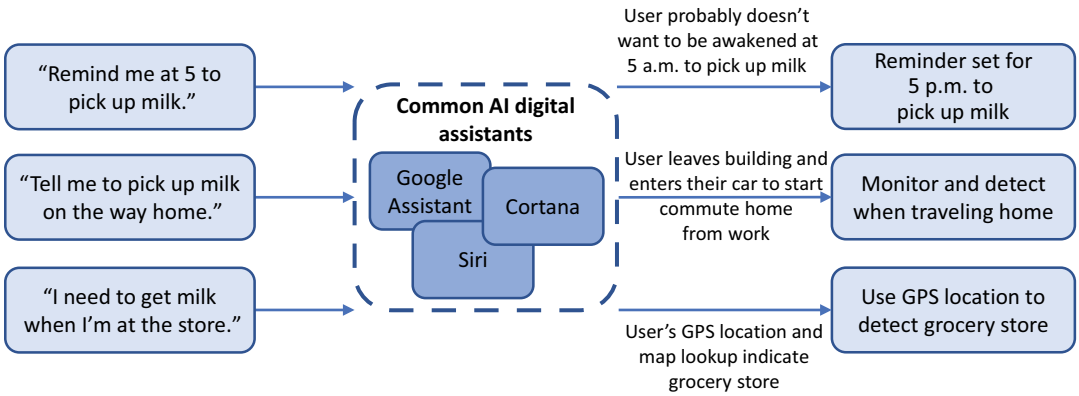


Figure 17.2 AI can take input from the user and make decisions that best suit the anticipated action. The AI isn't preprogrammed with all of these possible responses and decision trees. Instead, it uses data models and algorithms to apply context to the user input and interpret the meaning and appropriate outcome.

It's not true intelligence (yet), even in complex forms of AI; instead, it's an educated guess based on a data model that the AI has been trained with. This data model may include many variations and phrases, and may be able to learn new meanings over time. How does it learn, and where do these data models come from? That's where ML becomes important.

17.1.2 Machine learning

A great buzzword in computing over the past few years is *big data*. The concept is that computer systems, especially in the cloud, are a great resource for processing large amounts of data. *Really* large amounts of data. These processing jobs may run for a few minutes or hours, depending on the amount of the data and the calculations required, and allow you to prepare and analyze large volumes of data to determine specific patterns and correlations. These learnings form data models that other applications or



Figure 17.3 Large amounts of raw data are processed and made ready for use. Different preparation techniques and data sanitization may be applied, depending on the raw inputs. Then ML algorithms are applied to the prepared data to build an appropriate data model that reflects the best correlation among all the data points. Different data models may be produced and refined over time. Applications can use the data models on their own data inputs to help guide their decision making and understand patterns.

AI can use to make decisions. As shown in figure 17.3, ML involves a few steps and includes both inputs and outputs.

Here's how the most basic form of ML works:

1. To begin the process, large amounts of raw data are provided as input.
2. This data is processed and prepared in a usable format to focus on the specific data points required for analysis.
3. ML algorithms are applied to the data. This is where the real number crunching occurs. The algorithms are designed to detect and compute similarities or differences across the large number of data points.
4. Based on the analysis of the algorithms, a data model is produced that defines patterns within the data. These data models may be refined over time if parts of the model prove to be incorrect or incomplete when additional real-world data is applied.
5. Applications use the data models to process their own datasets. These datasets are typically much smaller than the raw data provided to the ML algorithms. If the data model is valid, even with a small data input from the application, the correct outcome or correlation can be determined.

ML often involves complex algorithms that are designed to process all the data points provided. Hadoop and Apache Spark are two application stacks commonly used to process big data. Azure HDInsight is a managed service that allows you to analyze the large datasets processed by these application stacks. To get a little deeper into the analysis and algorithms, data scientists commonly use the R programming language to develop the models required. Don't worry too much about what Hadoop or R is. The key point is that Azure can run the common ML tools that are widely accepted within the industry.

17.1.3 Bringing AI and ML together

A common application on a smartphone is the navigation app, as shown in figure 17.4. Your provider, such as Google, can track the route you take to work each day, what time you usually leave home, and how long it takes you to get there.

This Google Maps example shows AI and ML working together. AI is applied to know when to generate a notification based on the data received after processing the

www.dbooks.org

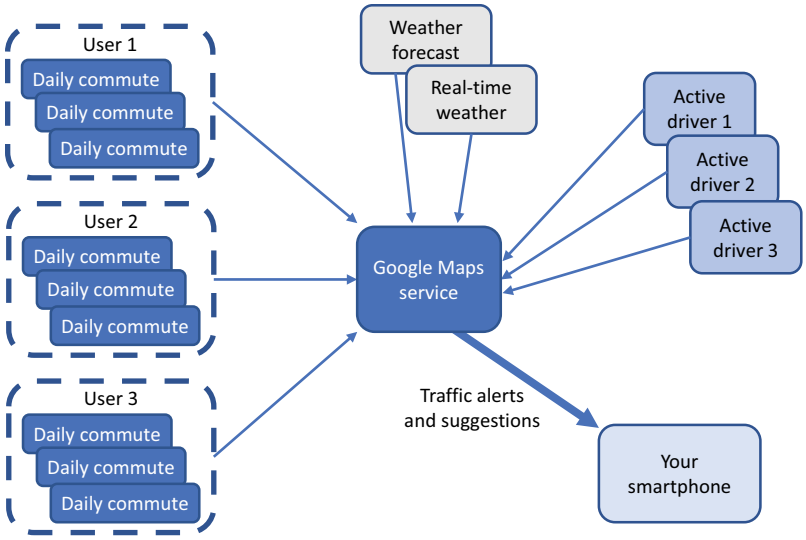


Figure 17.4 Each day, the Google Maps service receives multiple data points from users that record details of their commute. This data can be prepared and processed, along with the weather forecast and real-time weather during those commutes. ML algorithms can be applied to these large datasets and a data model produced. As a smaller sample of active drivers feed their current travel conditions or weather data into the Google Maps service, the data model can be applied to predict the commute and generate a traffic alert to their smartphones that suggests an alternative route home.

ML data model. Another example of AI and ML working together is the idea of setting a reminder to buy milk. If the AI was trained with ML data models, the assistant would know that you probably buy milk at the grocery store, so it wouldn't remind you if you went to the hardware store. The ML data model would also be able to help the AI understand that there's a greater probability that you want to be reminded of something at 5 p.m., not 5 a.m., so it shouldn't wake you at 5 a.m. to remind you to buy milk. If your smartphone tracks you getting in your car at 5 p.m. and starting to drive away from work, ML will generate a data model that predicts you're driving home, so that hour is a good time for the AI to remind you about buying milk.

These basic but powerful examples show how ML is used to improve AI. You train AI by providing a set of data points that are processed by ML to improve accuracy or decision making.

17.1.4 Azure ML tools for data scientists

I want to quickly cover a couple of ways that some real-world number crunching and ML work can be done. To make this chapter accessible to all, the exercises use the Microsoft Bot Framework for AI, and ML with Language Understanding Intelligent Service (LUIS). To get your hands dirty with ML, we need to focus a little more on data processing and algorithms.

In Azure, a couple of cool components help you dig into data on a massive scale. First, there's Azure Machine Learning itself, a web-based service that lets you visually build experiments by adding datasets and analysis models. These experiments can use data sources such as Hadoop and SQL, and additional programming support is provided for languages such as R and Python. You can drag and drop data sources, data–preparation techniques, and ML algorithms. You can adjust those algorithms and then review and tweak the data models produced.

Azure Machine Learning provides a low barrier for entry to the large-scale compute resources available in Azure. A primary benefit of performing ML data crunching in Azure is that you can access a large amount of compute power and use it for only the time required to complete your calculations. In traditional environments, those expensive compute resources would sit idle for large periods of time between data-processing jobs.

One other cool resource that helps you perform serious ML and number crunching in Azure is data science virtual machines (DSVMs). These VMs are available for both Linux and Windows. They come with many common applications preinstalled, including Jupyter Notebooks, Anaconda Python, and R Server or SQL Server (figure 17.5).



Figure 17.5 DSVMs are available for Windows and Linux. This Window Server 2016 DSVM comes with several preinstalled data science applications, such as R Server and Jupyter Notebooks. DSVMs let you get up and running quickly with processing big data and building ML algorithms.

www.dbooks.org

There's no need to install all the tools and dependencies on your local computer; you can create a DSVM with as much CPU and memory resources as you need to process your data quickly and then delete the VM when your processing job is complete and you have the data models you need.

17.2 Azure Cognitive Services

Okay, so what about AI services to make your apps smarter? In Azure, a set of related services makes up the Cognitive Services suite. The services cover a few common areas of AI that let you quickly integrate these intelligent resources into your applications, divided into the following general areas:

- Vision
- Speech
- Language
- Decision
- Search

More than two dozen services are part of the Cognitive Services family. Some of these services are

- *Vision*, which includes
  - *Computer Vision* for image analysis, captioning, and tagging.
  - *Face* for analyzing and detecting faces in images.
- *Speech*, which includes
  - *Speech Services* for analyzing and converting speech to text, and vice versa.
  - *Speaker Recognition* for identifying and verifying the speaker.
- *Language*, which includes
  - *Language Understanding (LUIS)* for understanding and processing interaction with users. We'll explore LUIS in the lab at the end of this chapter.
  - *Translator Text* for analyzing and correcting spelling mistakes or performing translations.
- *Decision*, which includes
  - *Content Moderator* for reviewing and moderating photos, video, and text.
  - *Personalizer* for analyzing patterns and providing recommendations to customers.
- *Search*, which includes
  - *Bing Custom Search* for implementing search on your custom data and within applications.
  - *Bing Autosuggest* for providing automatic suggestions as users enter search phrases and queries.

As you can see, many Azure services combine AI and ML features. This chapter focuses on language, specifically LUIS. This service is commonly used to build an intelligent bot that can help customers on your website. Then you can build an application that

uses AI services in Azure that can interpret phrases and questions, and provide the appropriate response to guide a user through an order process or support request.

17.3 Building an intelligent bot to help with pizza orders

A *bot* is an application that's programmed to respond to tasks and input from a user. If this sounds much like any normal application, well, it pretty much is! The difference is how the bot application determines the response.

A basic, common bot is often nothing more than an application that provides some form of automation. When a user sends a message, sets a tag on an email message, or submits a search term, the bot carries out preprogrammed tasks that perform a specific action. There's no real AI or ML here; the bot application is just responding to user input.

With the right framework, a bot can be extended and given a little more freedom and intelligence. At the start of our overview of AI, I discussed how a typical application must be preprogrammed with all the anticipated user inputs and what the corresponding output would be. But there's no flexibility if the user provides a different input phrase or a spelling mistake, for example.

Microsoft produces the Bot Framework, which allows an Azure bot to easily integrate the Bot Builder SDKs and connect to Azure Cognitive Services. With minimal code experience, you can build intelligent bots that use the power of Azure to deliver a great customer experience. Just don't try to build Skynet unless you know how *The Terminator* ends!

17.3.1 Creating an Azure web app bot

Let's deploy a bot and integrate some AI and ML services. The bot runs in an Azure web app and uses Microsoft Bot Framework to connect to LUIS and let a customer order pizza. Figure 17.6 outlines what these exercises will build and what services are used.

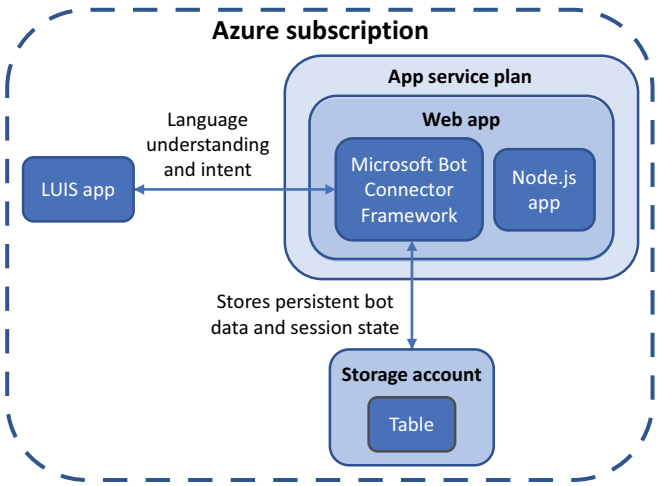


Figure 17.6 In the upcoming exercises, you'll create a web app bot that integrates multiple Azure AI and ML services to interact with a customer and help them order pizza.

www.dbooks.org

Try it now

To create an Azure web app bot, complete the following steps:

1. Open the Azure portal, and select Create a Resource in the top-left corner.
2. Search for and select Web App Bot, and then select Create.
3. Enter a name for your bot, such as `azuremol`; then create a new resource group and provide a name, such as `azuremolchapter17`.
4. Select the most appropriate region for you, and choose the F0 pricing tier. Your bot won't process a lot of messages, so the free (F0) tier is fine.
5. Select a bot template, and choose the Node.js SDK language.
6. Create a basic bot, as we'll provide our own sample application code in a later exercise. This step creates a LUIS app you can use to perform language training and ML.
7. Choose the most appropriate region for your LUIS app, and create a new LUIS account.
8. Provide a name for the LUIS account, such as `azuremol`. This LUIS account handles the user sentiment for our bot.
9. Choose App Service Plan, and create a new plan. Provide a name, such as `azuremol`, and again, select the most appropriate region for you.
10. Turn off App Insights, because your bot won't use it. As in earlier chapters on web apps, for production use you may want to harness the power of App Insights to gain visibility into the performance of your application by streaming data and analytics straight from the code.
11. Accept the option to autocreate the Microsoft app ID and password, accept the agreement, and then choose Create.

It takes a few minutes to create the web app bot and associated components. A lot happens behind the scenes:

- An Azure App Service plan is created.
- A web app is deployed, along with a sample Node.js web application.
- A LUIS app is created, and the connection keys are configured with your web app.
- A bot is created with the Microsoft Bot Connector, and the connection keys are configured from your web app.

17.3.2 Language and understanding intent with LUIS

One of the Azure Cognitive Service areas that we looked at earlier is language. This makes sense, because some form of language is often used to interact with an AI. You can use LUIS to process a message or phrase from the user and determine their intent. That intent helps your app provide an appropriate response. Let's extend your bot with LUIS.

Try it now

To build a LUIS app and use ML to train it, complete the following steps

1. Open a web browser to www.luis.ai, and sign in with the same Microsoft credentials as your Azure subscription.
2. Select My Apps and then choose your app, such as `azuremol`. Your LUIS app name likely has some additional numerical characters appended to it from the bot name you specified in the Azure portal.

   Some prebuilt intents were created, but you want to overwrite the LUIS app with a more pizza-store-focused sample.
3. Download the azuremol.json file from GitHub at https://github.com/fouldsy/azure-mol-samples-2nd-ed/blob/master/17/luisapp/azuremol.json to your local computer. To make life easier, select the Raw button in GitHub to see only the contents of the file.
4. Back in your LUIS app, choose to Manage the app, and then select Versions.
5. Choose to import a version, browse to and select the azuremol.json file you downloaded, enter a version name of `1.0`, and then select Done.
6. Go back to Build in the top menu to see the imported intents from the sample app. Choose one or two of the intents, such as `greetings or orderFood`, and look at some of the example phrases a customer could use to communicate with the bot.
7. Before you can see the app in action, you must train it. Select Train, and wait a few seconds for the process to complete. Figure 17.7 shows the ML processes at work to train your LUIS app.

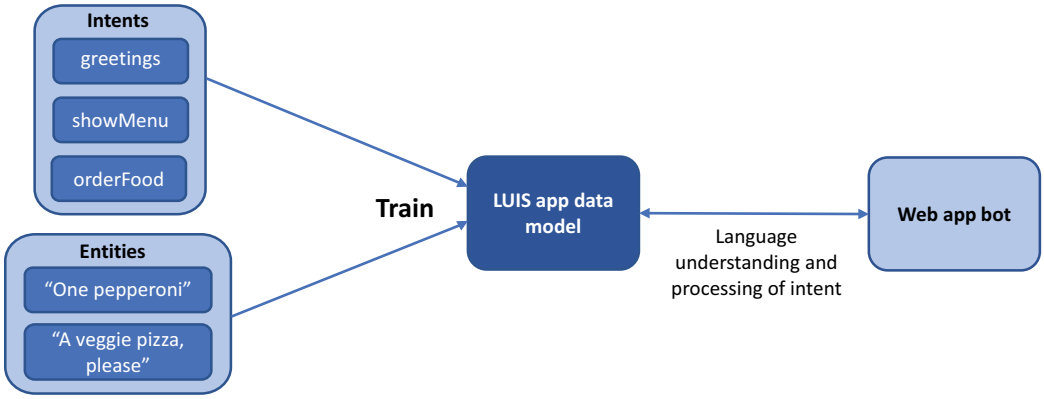


Figure 17.7 When you train the LUIS app, the intents and entities are input and processed to create a data model. Then your web app bot uses this data model to process language understanding and intent. The number of intents and entities input for processing is small, so the data model isn't perfect. In the real world, many more intents and entities would be provided, and you'd repeatedly train, test, and refine the data model to build progressively larger datasets to build an accurate model for processing language and intent.

www.dbooks.org

In a more complex real-world application, it may take longer to complete this training process as all your intents and entities are processed by the ML algorithms to build the required data model for your app to respond appropriately to customer communication.

8 With the LUIS app trained, select Test, and enter a couple of greetings, such as *hi* and *hello*. Below each of your messages is the top-scoring intent, along with the probability that the message, or utterance, you entered matches the intent. These basic greetings should match the `greetings` intent.

9 Try to enter a different greeting, such as *(good) afternoon* or *(good) evening*. The single-word greeting based on the time of day may return an incorrect top-scoring intent, such as `orderStatus`. Try some other phrases until something doesn't line up with the expected intent, which indicates that the LUIS app doesn't fully understand what you mean. Select one of your incorrect messages, such as *morning*, and choose Inspect.

10 On the Inspect menu, choose to edit the incorrect top-scoring intent. From the drop-down menu, choose `greetings`, or whatever the most appropriate intent is for your incorrect phrase.

11 You've made a change to your app, so choose to Train the LUIS app again. Figure 17.8 shows how to provide additional inputs for the ML algorithms to process the data model and refine the language understanding and intent.

12 In the test-messages window, reenter the incorrect message, such as *morning*. This time, the top-scoring intent should correctly be identified as `greetings`.

13 To make the updated LUIS app available to your web app bot, choose the Publish option from the top menu. Accept all the defaults, and choose to publish to a production slot. It takes a few seconds to complete the publish process.

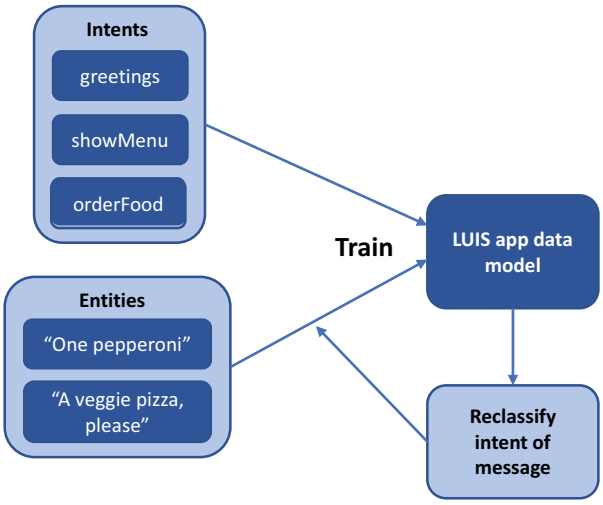


Figure 17.8 As you reclassify the intent of messages and retrain the LUIS app, the data model is refined as additional data inputs are provided to the ML algorithms. When you enter similar greetings in the future, the data model will (ideally) be improved and will respond more appropriately.

Remember that your bot runs on a web app, so it has production and staging slots, as you learned way back in chapter 3. In the real world, you should publish to a staging slot, verify that everything works as expected, and then publish to the production slot. The same PaaS features that allowed you to test and move web code between development and production lifecycles also benefit the lifecycle of your web app bot powered by LUIS.

In this basic example, ML was able to take your data input of *(good) morning* as a greeting and to understand that similar greetings, such as *(good) evening*, are also greetings. ML works best when a large set of data can be input to the data model, so it's important to thoroughly test and help train your app. The AI—in this case, the LUIS app—is only as good as the size and quality of the data provided to the ML algorithms.

17.3.3 Building and running a web app bot with LUIS

You now have a basic web app bot in Azure and a LUIS app that handles the language processing and returns the customer intent. To integrate the two, you need to modify the code for your bot to use LUIS. SDKs are available for the C# and Node.js programming languages. I find that Node.js makes it a little quicker and easier to understand what happens in the code, if this is new to you. If you're familiar with C#, you're welcome to explore the C# SDK when you're finished with this chapter. For now, let's use a basic Node.js app from the GitHub sample repo to see your bot in action with LUIS.

Try it now

To update your web app bot with your trained LUIS bot, complete the following steps:

1. In the Azure portal, choose Resource Groups from the menu at left, and choose your resource group, such as `azuremolchapter17`; then select your web app bot, such as `azuremol`.

   Let's use a sample bot from our GitHub samples repo. The sample bot is written in Node.js, but as with previous sample apps, don't worry if that's not your thing.

2. To deploy the sample bot, open Cloud Shell. If necessary, clone the GitHub samples repo in your Cloud Shell as follows:

   ```
   git clone https://github.com/foouldsy/azure-mol-samples-2nd-ed.git
   ```

3. Change to the directory for chapter 17:

   ```
   cd azure-mol-samples-2nd-ed/17/webappbot
   ```

4. Initialize the Git repo, and add the bot files:

   ```
   git init && git add . && git commit -m "Pizza"
   ```

5. To upload the sample bot, create a connection to your web app. The following command gets the web app repository and configures your local samples Git repo to connect to it. In previous chapters, I made you dig around for this

www.dbooks.org

address, but by now, I hope you've started to explore what else the Azure CLI can do and realized that much of this information can be obtained quickly.

```
git remote add webappbot \
  $(az webapp deployment source config-local-git \
  --resource-group azuremolchapter17 \
  --name azuremol \
  --output tsv)
```

6 Push the sample Node.js bot to your web app with the following command:

```
git push webappbot master
```

7 When prompted, enter the password for the Git user you created and have used in previous chapters (the account created in chapter 3).

If you didn't write your Git password on a sticky note

If you've forgotten the password, you can reset it. First, get the username of your local Git deployment account:

```
az webapp deployment user show --query publishingUserName
```

To reset the password, enter the name of your account from the previous command, and then answer the prompts to set a new password. The following example resets the password for the user account named azuremol:

```
az webapp deployment user set --user-name azuremol
```

Let's look at figure 17.9 to see what you've deployed. Now the LUIS app is trained with ML algorithms, and your data model is ready for the Node.js app to let customers interact and order pizza.

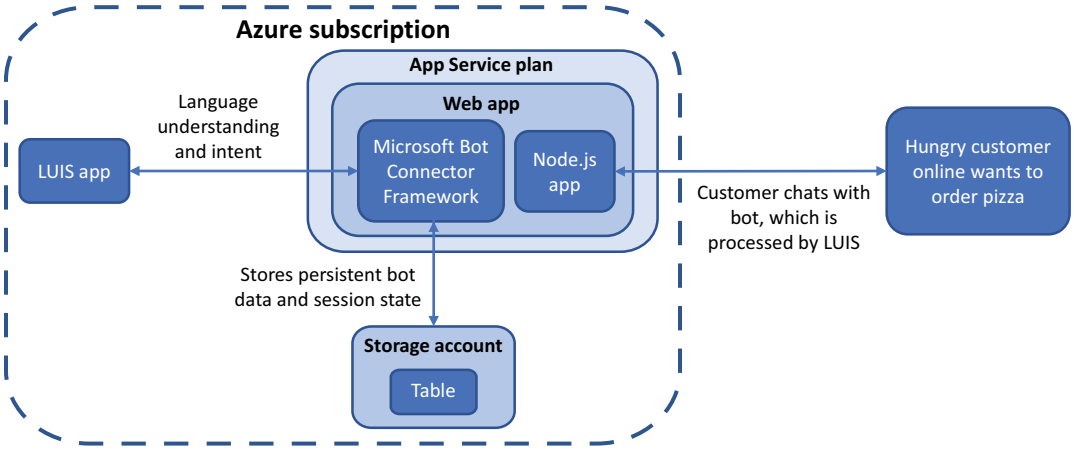


Figure 17.9 Now a customer can access your bot online and ask to view the menu or order pizza. LUIS provides the language understanding, which allows the bot to process orders and send them to Azure Storage for additional processing.

Back in the Azure portal for your web app bot, select Test in Web Chat. It takes a few seconds the first time you connect to the bot, but then you should be able to interact, view the list of pizzas on the menu, and create an order, as shown in figure 17.10. Try it yourself!

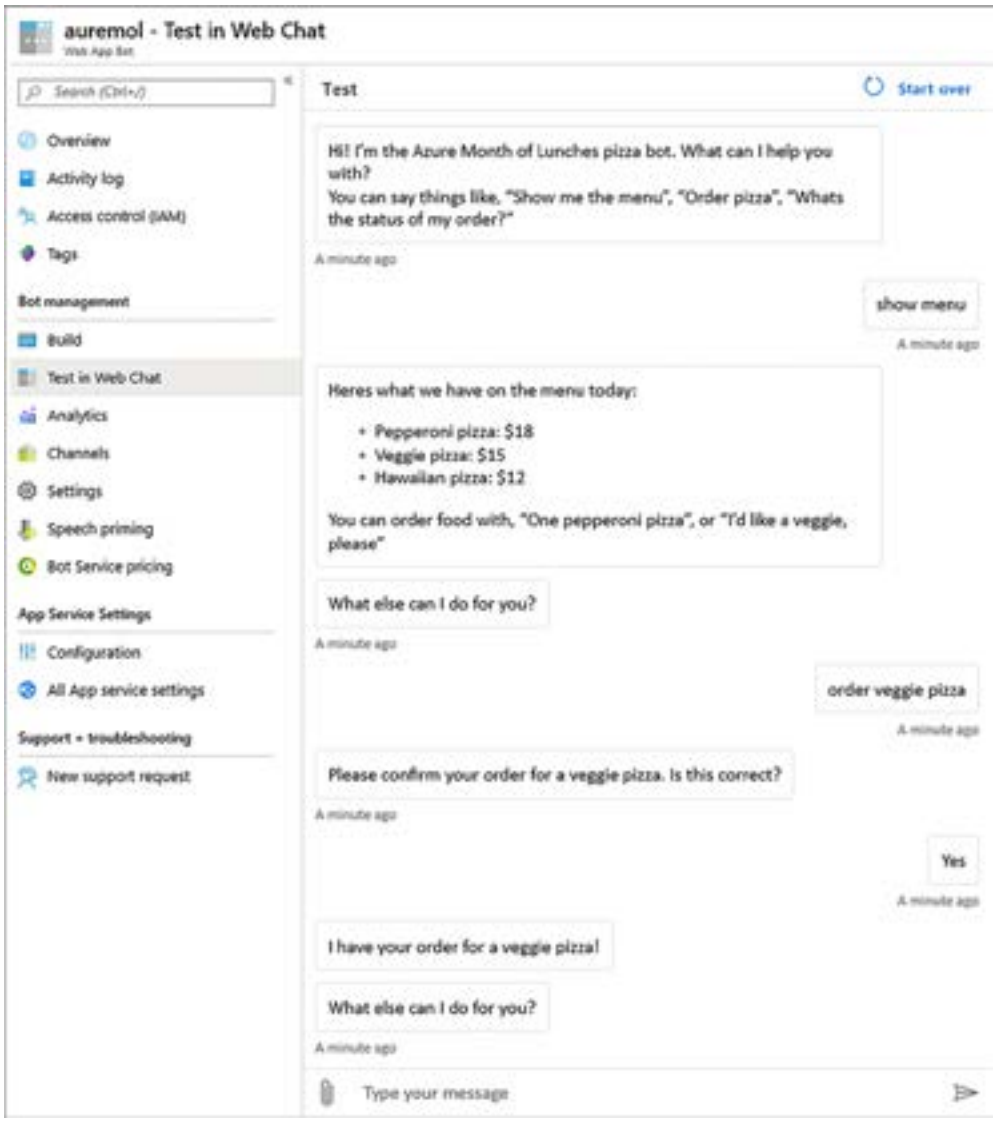


Figure 17.10 With your web app bot running, start a conversation, and try to order a pizza. In this example dialogue, you can view the menu, order a pizza, and check the order status. The app is basic and isn't really creating orders or updating the status beyond what pizza was ordered, but I hope that the exercise shows how you can quickly deploy a bot in Azure.

www.dbooks.org

I hope that these basic exercises have given you an idea of what Azure can offer for AI and ML. The web app bot with LUIS can be expanded to include additional Azure Cognitive Services, such as Spell Check and Translator. These services let you interpret words and phrases if the user spells them incorrectly or let your bot converse in multiple languages. Or you could use Face and Personalizer to detect which customer was making an order based on facial recognition from their camera and automatically suggest pizzas that they may like.

The ML was part of the LUIS app, but many more ML resources and tools are available in Azure. The ability to process large datasets and compute ML data models on high-performance Azure compute resources lowers the entry for you to build applications backed by some serious datasets. The applications are more accurate and efficient, and there's no hardware to buy or special tools to install because the DSVMs include all the components required. Not all applications are a good fit for AI and ML, but as customers start to expect more intelligent features from your business, these Azure services can often help differentiate you.

Batch workload processing

Two other areas of Azure that may be of interest in terms of big data and compute for ML are the Azure Batch and HPC services. Azure Batch lets you perform large, repetitive compute tasks without the need to manage clusters of schedulers for the work. Batch runs tasks on VMs with its own management and scheduler to help you, just as scale sets include autoscale and load balancing for VMs. Although Batch isn't directly related to ML, if you need other large compute-processing tasks, Batch is a great fit.

There are also high-performance computing (HPC) components in Azure for large VM sizes or access to graphical processing unit (GPU) VMs. Specific tools and suites such as DataSynapse and Microsoft HPC Pack can also be used to run applications that demand a large amount of compute power.

Areas such as ML, Azure Batch, and HPC are great examples of how to use cloud computing providers like Azure to run large compute tasks. You pay only for the compute resources you use, so you don't need to purchase and maintain expensive equipment that sees minimal use.

17.4 Lab: Adding channels for bot communication

In the earlier examples, you communicated with your bot through a test window in the Azure portal. Channels allow you to expand how you can interact with your bot. You can allow your bot to communicate with Skype or Facebook Messenger, or with apps like Microsoft Teams and Slack. The Azure Bot Service simplifies the steps needed to integrate a bot with those external services:

1. In the Azure portal, select your web app bot, and then choose Channels.
2. Pick a channel you like, such as Skype.

Other channels often require you to create a developer connection, such as to Facebook or Slack. Skype lets you copy and paste some HTML code to make it work.

3 Provide any required information, such as Bot Application ID. You can find this ID under Settings for Bot Management.

4 If needed, use the online code editor to create a basic HTML page, such as default.htm, in the wwwroot directory, and paste any embedded code for your channel. You can open your web app from the Azure portal and then select its URL to open the default.htm page that includes your channel code, such as http://azuremol.azurewebsites.net/default.htm.

www.dbooks.org

18 Azure Automation

Where possible, you shouldn't log in to a server manually and make changes. Software doesn't need to be installed by clicking buttons in a GUI, and updates don't need to be made to configuration files in a text editor. These manual actions introduce an opportunity for errors to occur, which can result in misconfigurations and application failures. If you want to replicate the configuration of a server, can you remember all the steps that were required to get the existing server up and running? What if you need to do it again in six months?

In chapter 16, we touched on a way to automatically check for and apply updates to servers. This magic happened with the use of Azure Automation. In this chapter, we'll examine how you can create, run, and edit runbooks, and use PowerShell Desired State Configuration to install applications and configure servers automatically.

18.1 What is Azure Automation?

An Azure Automation account brings together many elements, as shown in figure 18.1. A core feature is creating and running scripts on demand or on a defined schedule. You can create scripts in PowerShell or Python and let the Azure platform handle the scheduling and execution of those runbooks. You can share credentials and connection objects, and automatically apply and report on desired configurations of servers. Update Management, which we examined in chapter 16, keeps your servers secure and up to date with the latest host patches and updates throughout the lifecycle of your application environment.

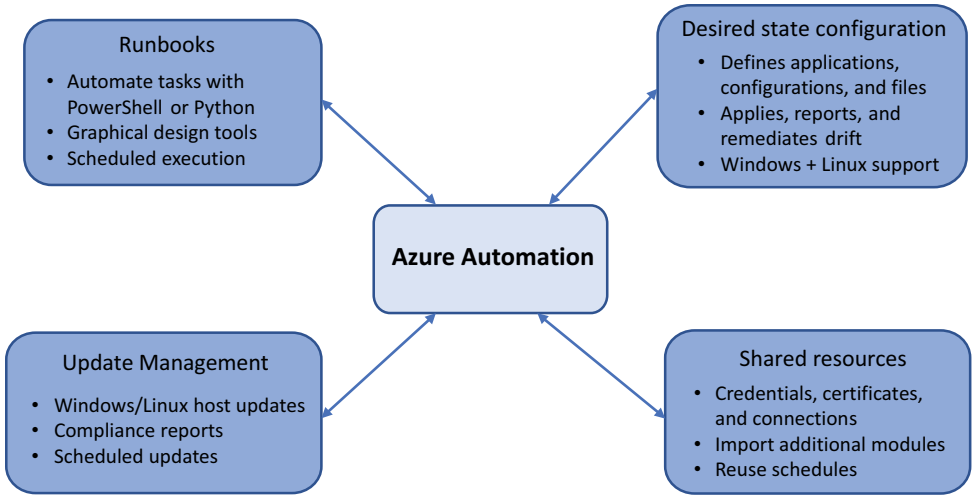


Figure 18.1 Azure Automation provides many related features. A shared set of resources—such as credentials, certificates, schedules, and connection objects—can be used to run PowerShell or Python scripts automatically on target servers. You can define the desired state of a server, and Azure Automation installs and configures the server appropriately. Host updates and security patches can be applied automatically. All these features work across both Windows and Linux servers, in Azure, other cloud providers, and on-premises.

To simplify management across multiple runbooks or desired state configurations in an Automation account, you can share the following resources:

- *Schedules* let you define a set of times and recurrences that can be applied to each runbook or Update Management task. If you want to change a regular occurrence later, you can change one of the shared schedules rather than each individual runbook or Update Management task that uses it.
- *Modules* extend the core functionality by storing additional PowerShell modules. The base Windows PowerShell and Azure modules are already available, but additional modules, such as for Linux management, can be added and used in runbooks.
- *Credentials* for the different accounts that have permissions to execute various runbooks are stored as assets, not defined in each runbook. This approach lets you update and reset credentials as needed, and each runbook that uses them is updated automatically. Thus credentials aren't stored in plain text in runbooks, which increases the security of the runbooks.
- *Connections* define authentication properties to Azure AD service principals. This is a special type of user account that allows runbooks to access your Azure resources. These connections typically use digital certificates, not usernames and passwords, to provide an additional layer of security.
- *Certificates* are often integrated with connection assets to provide a secure way to verify the identity of a service principal. As with basic credentials, you can update

www.dbooks.org

these certificates regularly in a central location, and each runbook that uses them can access the new certificates automatically. You can create and store your own certificates for use with runbooks or desired state configuration definitions.

- *Variables* provide a central place for runtime values such as names, location strings, and integers to be stored. When your runbooks are executed, these variables are injected. This approach limits the amount of hardcoded resources inside each runbook.

Work smarter, not harder

In chapter 16, we touched on how Azure management services work together to monitor and report on servers in Azure, on-premises, or in other cloud providers. You install and configure the required agents on remote servers, and then provide a way for them to connect back to the Azure infrastructure.

Azure Automation can also work across platforms and infrastructure. The hybrid runbook worker can execute Automation runbooks on servers outside Azure, for example. You continue to use the shared Automation assets that define credentials, connections, and certificates, only this time, those assets can be used to define the authentication components for the different platforms. You can also use desired state configurations on non-Azure VMs, for both Windows and Linux.

In all cases, a gateway component is installed in the remote environment to act as a proxy for the Automation commands as they're sent to the designated targets. This gateway proxy approach provides a single connection point for Automation into the remote environments and minimizes any security concerns, because there's no direct access to otherwise remote servers.

Runbooks and desired state configuration definitions may need to be edited slightly to run against on-premises physical servers compared with Azure VMs. As with Azure Backup, Site Recovery, or Update Management, the advantage of Azure Automation is that it provides a single management plane and set of tools to deliver automation across all of your infrastructures and servers.

18.1.1 Creating an Azure Automation account

Let's jump in by creating an Azure Automation account and looking at the default runbooks that are included. The demo runbooks provide a great framework to build your own runbooks, and there's also a graphical editor that you can use to drag and drop building blocks to generate automation scripts.

Try it now

To create an Azure Automation account and sample runbooks, complete the following steps:

1. In the Azure portal, select Create a Resource in the top-left corner.
2. Search for and select Automation, and then select Create.

The Automation and Control option also creates an Operations Management Suite (OMS) workspace and configures the Automation Hybrid Worker to manage resources outside Azure. OMS is somewhat on the way out, replaced by the core Azure services we looked at in previous chapters. For now, choose to create only the automation resource.

3. Enter a name, such as azuremol, and then create a new resource group, such as `azuremolchapter18`.
4. Select the most appropriate Azure region closest to you, and accept the option Create Azure Run As Account.

   The Create Run As Account option creates additional accounts in Azure AD. Security certificates are also created to allow the accounts to authenticate in an automated fashion without the need for user prompts or password saves. You could create and specify additional regular account credentials, defined as an Automation asset, to provide more granular control of which accounts are used to run certain runbooks.

When combined with RBAC, which we looked at in chapter 6, specific Run As accounts can be created for runbooks, which provide a limited set of permissions needed to accomplish the tasks each runbook, or set of runbooks, requires. From a security perspective, this approach allows you to audit and control how and when these accounts are used. Avoid the temptation to create a single Run As account that provides admin-like permissions, because this approach provides little protection against abuse.

18.1.2 Azure Automation assets and runbooks

The Azure Automation account you created in section 18.1.1 includes some sample runbooks. Both PowerShell and Python samples are available. Connection assets and certificates are also added to the Automation account for the Run As accounts that were created. Let's explore those shared connection assets.

Try it now

To see the configured assets and sample runbooks, complete the following steps:

1. In the Azure portal, select Resource Groups at left; choose your group, such as `azuremolchapter18`; and select your Azure Automation account, such as `azuremol`.
2. Under Shared Resources in the menu on the left, select Connections.
3. Select AzureRunAsConnection, as shown in figure 18.2.
4. Choose Certificates from the main menu on the Automation account under Shared Resources, and then choose the AzureRunAsCertificate. As shown in figure 18.3, the digital thumbprint matches RunAsConnection from the preceding step.

www.dbooks.org

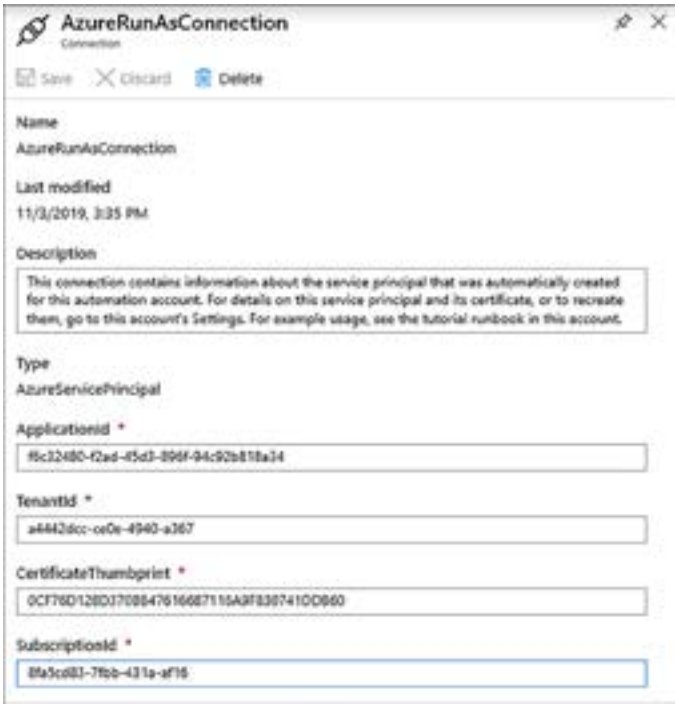


Figure 18.2 Information on the Run As account includes an ApplicationId and TenantId—specific properties for Azure AD that help identify the credentials for this account. A CertificateThumbprint matches a digital certificate that we'll look at in the next step.



Figure 18.3 The thumbprint of the RunAsCertificate matches that shown in RunAsConnection. In your runbooks, you define which connection asset to use. The appropriate certificate is used to log in to the Azure account.

5 Now that you understand the assets for connections and certificates, let's look at one of the sample runbooks. Choose Runbooks from the menu at left in the Automation account. A few sample runbooks are available.

6 Choose the PowerShell runbook called AzureAutomationTutorialScript.

7 Across the top of the sample runbook are options to Start, View, and Edit the runbook. These options should be self-explanatory!

You also have an option for scheduling, which lets you create or select a shared resource that defines a schedule to execute the runbook at a given time, and an option for webhooks, which lets you create a webhook URL to execute the runbook from some other script or action. Choose View.

Azure Automation and source control with GitHub

Runbooks can be integrated with a source control system, such as GitHub. One of the great benefits of a source control system for your runbooks is that it provides a way for change management to be documented and to revert to earlier versions of the runbooks in the event of a problem.

Each time you save an Azure Automation runbook, a new version is committed to source control. You don't need to leave the runbook editor, because the Azure platform and the source control system are configured to work back and forth. If you run into a problem with the new runbook, you can pull a previous version from source control that allows jobs to continue to run without delay, and then troubleshoot why the updated version has an issue.

Using source control also provides a record of what changes occurred and when. If you need to audit your runbooks or understand how they developed over time, source control systems provide a great way to see the differences with each revision.

18.2 Azure Automation sample runbook

Let's examine how the sample PowerShell runbook, AzureAutomationTutorialScript, connects to Azure and gathers information about your resources. You can follow along with the Python sample runbook if you prefer; the layout is similar. PowerShell and Python are the only languages currently supported in Azure Automation runbooks. The following listing sets up the connection credentials in the runbook.

Listing 18.1 Setting up connection credentials

```
$connectionName = "AzureRunAsConnection"
try
{
  # Get the connection "AzureRunAsConnection "
  $servicePrincipalConnection=Get-AutomationConnection -Name
➥$connectionName
```

Creates an object for $connectionName

Makes the connection request

Creates a service principal object

www.dbooks.org

```
    "Logging in to Azure..."
    Add-AzureRmAccount `
        -ServicePrincipal `
        -TenantId $servicePrincipalConnection.TenantId `
        -ApplicationId $servicePrincipalConnection.ApplicationId `
        -CertificateThumbprint
➥$servicePrincipalConnection.CertificateThumbprint
}
```

Logs in to Azure

The code begins by creating an object for `$connectionName`. In the "Try it now" exercise, you saw that a default connection asset for AzureRunAsConnection was created. As you create your own runbooks, you may want to create additional Run As accounts and connection assets to separate the runbooks and the credentials that they use. The connection parts and exception handling that we'll look at next should be common across all runbooks. As needed, you can change the Run As connection asset to use.

Next, a `try` statement is used to make the connection request. A service principal object named `$servicePrincipalConnection` is created, based on `$connectionName`. The runbook then logs in to Azure with `Add-AzureRmAccount` and uses the `$servicePrincipalConnection` object to obtain the `TenantId`, `ApplicationId`, and `Certificate-Thumbprint`. We discussed these parameters as part of the connection asset earlier. The certificate asset that matches the thumbprint of `$servicePrincipalConnection` is then used to complete the login to Azure.

The next listing shows that if the connection fails, the runbook catches the error and stops execution.

Listing 18.2 Catching an error and stopping the runbook execution

```
catch {
     if (!$servicePrincipalConnection)
     {
          $ErrorMessage = "Connection $connectionName not found."
          throw $ErrorMessage
     } else{
          Write-Error -Message $_.Exception
          throw $_.Exception
     }
}
```

The `catch` statement handles any errors as part of the login attempt. If a service principal connection couldn't be found, an error is output. This error usually means that the connection asset you specified can't be found. Double-check the name and spelling of your connection.

Otherwise, the connection object was found, and the service principal was used to log in, but that authentication process was unsuccessful. This failure could come from a certificate that is no longer valid or a Run As account that is no longer being enabled. This functionality shows how you can revoke an account in Azure AD and ensure that any runbooks that use the credentials can no longer run.

Now the runbook gets a list of all Azure resources.

Listing 18.3 Getting a list of Azure resources

```
$ResourceGroups = Get-AzureRmResourceGroup
foreach ($ResourceGroup in $ResourceGroups)
{
    Write-Output ("Showing resources in resource group "
➥+ $ResourceGroup.ResourceGroupName)
     $Resources = Find-AzureRmResource -ResourceGroupNameContains
➥$ResourceGroup.ResourceGroupName |
➥Select ResourceName, ResourceType
     ForEach ($Resource in $Resources)
     {
         Write-Output ($Resource.ResourceName + " of type "
➥+  $Resource.ResourceType)
     }
     Write-Output ("")
}
```

The final part of the runbook is where your runbook code would go. An object is created for `$ResourceGroups` that gets a list of all available Azure resource groups. Then a `foreach` loop goes through the resource groups, finds a list of resources, and writes out a list of the resource names and types.

This basic example shows how you can interact with Azure when the runbook has authenticated against the subscription. If you implement RBAC on the Run As account, only the resource groups that the account has permissions to see are returned. This approach to RBAC highlights why it's a good security principle to create and use scoped Run As accounts to limit runbooks' access to resources in your Azure environment. Always try to provide the least privileges necessary.

If all this PowerShell or Python is new to you, don't worry. Both are great, basic scripting languages that also can be used to develop complex, powerful applications. As a developer, you should find either language relatively easy to pick up and use. If you're an IT pro, automating tasks helps free time for you to perform all the other jobs that are stacked up, and both PowerShell or Python are good places to start. Manning Publications has some other great books to help you, too!

18.2.1 Running and viewing output from a sample runbook

Now that you've seen what the sample runbook script contains and how the connection and certificate assets are used, let's execute the runbook and look at the output.

Try it now

To see the runbook in action, complete the following steps:

1. Close the window that shows the content of the runbook, and return to the overview of AzureAutomationScriptTutorial.

www.dbooks.org

2 Select Start at the top of the runbook window.

3 Confirm that you wish to start the runbook, and wait a few seconds for the runbook to begin to run.

4 Select Output, as shown in figure 18.4, and watch the console window as the runbook logs in to Azure, gets a list of resource groups, and loops through and outputs the list of resources in each group.

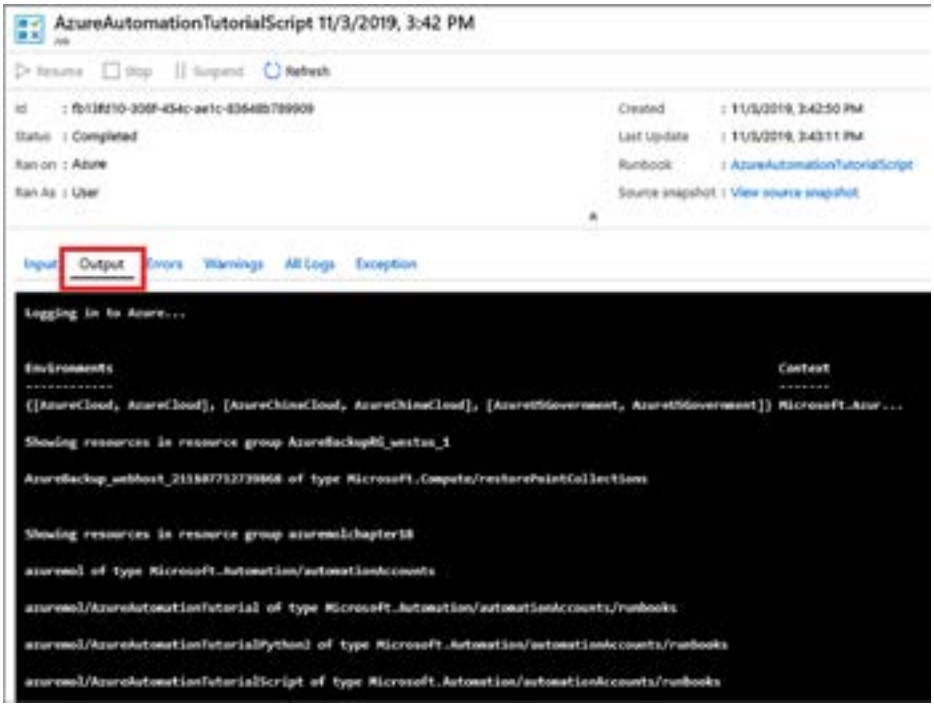


Figure 18.4 You can view the output of the runbook, along with any logs that are generated or errors and warnings. This basic example completes in a few seconds, but more-complex runbooks may take longer. You can monitor the status of those longer runbooks and stop or pause their execution as needed.

Automation runbooks don't need to exist in isolation. One runbook can execute another runbook. This ability lets you build complex, multistep automation and minimize the duplication of code. As you design and build runbooks, try to break them down into small, discrete blocks of code. Common functions that you may reuse, such as logging in to Azure and generating a list of resources or a list of VMs, should be created as small runbooks that can be included in larger runbooks. As new PowerShell cmdlets are released or parameters are changed, you can quickly update a single shared runbook that includes those cmdlets rather than needing to update multiple different runbooks. At first, it may not seem that smaller, reusable runbooks are worth

a little bit of extra work, but as your environment and Automation use grows, you'll thank me! A lot of what you've done in this book has been in smaller deployments, but start to think about how to deploy and manage applications at scale.

18.3 PowerShell Desired State Configuration (DSC)

Chapter 12 introduced the concept of VM extensions. An *extension* is a small software component that's installed in a VM to perform a given task. The VM diagnostics extension was installed on a VM to allow performance metrics and diagnostic logs to be reported back to the Azure platform from inside the VM. That's great, but we also talked a little about how you can install software automatically.

One way to install software and configure a server is to use PowerShell Desired State Configuration (DSC). With DSC, you define how you wish a server to be configured—the desired state. You can define packages to be installed, features to be configured, or files to be created, for example. What's great about DSC is that it goes beyond the first install-and-configure action. Over time, servers often undergo maintenance or troubleshooting events in which configurations and packages are manually changed. Then the server would deviate from the desired state that you initially defined. Figure 18.5 shows how Azure Automation can act as a central server that stores the DSC definitions, allowing target servers to receive their configurations and report back on their compliance.

The Local Configuration Manager (LCM) on each target server controls the process for connecting to the Azure Automation pull server, receiving and parsing the DSC definition, and applying and reporting on compliance. The LCM engine can operate without a pull server, where you locally call the process to read and apply a

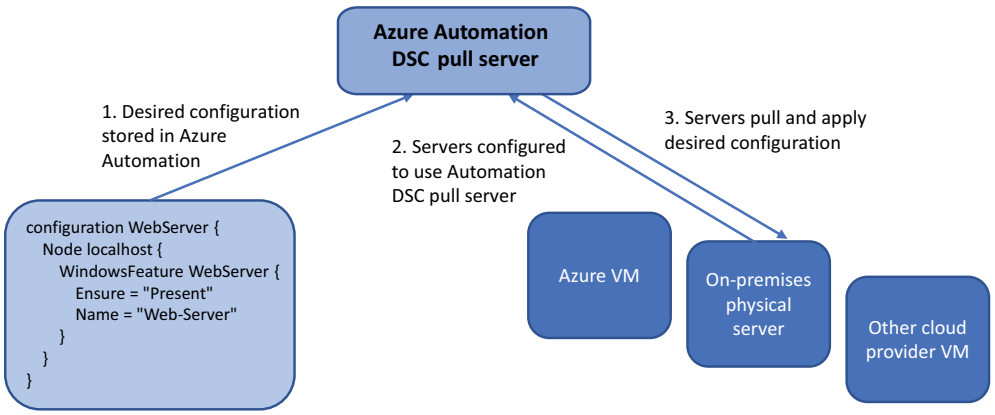


Figure 18.5 The desired state configuration for a server is created and stored in Azure Automation. The Automation account acts as a pull server, which allows connected servers to pull the required configuration from a central location. Different configuration modes can be set for the remediation behavior of the server if their configuration deviates from the desired state.

www.dbooks.org

DSC definition. In this mode, where you manually push the configuration to the LCM engine, you miss out on a lot of the central controls and reports that are often needed when you manage many servers.

There's also flexibility in how the target servers process the DSC definitions received from the Azure Automation pull server. You can configure DSC to operate in one of three configuration modes:

- *Apply only*—Your desired state is pushed and applied to the target server, and that's it. This is like the behavior of the Azure Custom Script Extension in that any configurations or installations are applied when first deployed, but there are no processes in place to stop those configurations from manually changing over the lifecycle of the server.
- *Apply and monitor*—After the server has the desired state applied, DSC continues to monitor for any changes that cause the server to deviate from that initial configuration. A central report can be used to view servers that are no longer compliant with their desired state. This configuration is a good compromise between the need to keep a server compliant with the desired state and providing an element of human interaction to decide on remediation options.
- *Apply and autocorrect*—The most automated and self-contained configuration applies the desired state, monitors for any deviations, and automatically remediates the server should any changes occur to ensure that it remains compliant. There's a danger that legitimate manual changes will be overwritten and instead returned to the configured desired state, but this configuration mode makes sure that the settings you assign always take priority.

PowerShell DSC can be used on VMs that run in other cloud providers, as well as on-premises VMs and physical servers. Thanks to .NET Core, PowerShell DSC can also be used on Linux servers, so it's not a Windows-only solution. This cross-provider, multi-OS support makes PowerShell a powerful choice for configuring and managing servers at scale.

You can build and maintain your own DSC pull server, but the built-in features of Azure Automation provide some additional benefits:

- Credentials are centrally managed, and certificates are automatically generated.
- Communication between the DSC pull server and target servers is encrypted.
- Built-in reports are provided for DSC compliance, and there's integration with Log Analytics to generate more detailed reports and alerts.

This section is very much a crash course in PowerShell DSC; it's a powerful component by itself and has been widely available for a few years now. When combined with Azure Automation, DSC is a great choice for automating the installation and configuration of software. Think back to the earlier chapters on virtual machine scale sets, for example. You can apply a DSC configuration to the scale set with Azure Automation, and then, as each VM is created in the scale set, it will be configured automatically with the required components and application files.

18.3.1 Defining and using PowerShell DSC and an Azure Automation pull server

I hope that whirlwind tour of PowerShell DSC has given you an idea of what's possible! Let's use PowerShell DSC to automate the example of installing a basic web server on a VM.

Try it now

To see PowerShell DSC in action, complete the following steps:

1. Create a Windows Server 2019 Datacenter VM, and open TCP port 80 for HTTP traffic. No more hand-holding now that we're in chapter 18! You can create the VM in the Cloud Shell or Azure portal—your choice. Use the resource group you created in the previous exercises, such as `azuremolchapter18`. You can continue with the next few steps as the VM deploys.
2. On your local computer, create a file named webserver.ps1; enter the following code; and save and close the file when you're done:

```
configuration WebServer {
    Node localhost {
        WindowsFeature WebServer {
            Ensure = "Present"
            Name = "Web-Server"
        }
    }
}
```

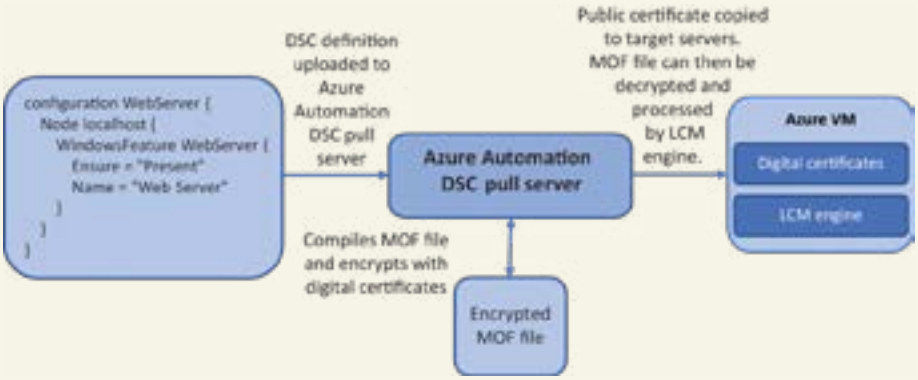
3. In the Azure portal, select your resource group, and then choose your Automation account.
4. At left, choose State Configuration (DSC); select the Configurations tab; and at the top of the window, choose to Add a configuration.
5. Browse to and select your webserver.ps1 file. The configuration name must match the filename, so accept the default name webserver, and then choose OK.

   It takes a few moments to upload and create the configuration.
6. When it's ready, select the configuration from the list, and then choose Compile.

Behind the scenes of DSC

Let's pause to talk about what happens when you compile the configuration, as shown in the figure in this sidebar. To distribute the DSC definitions, your PowerShell files are converted to a Managed Object Format (MOF) file. This file type is used for more than just PowerShell DSC and allows configuration changes on Windows components in a central, well-understood way. Any DSC definition, not just in Azure Automation, must be compiled before it can be applied to a target server. The LCM engine accepts and processes only MOF files.

www.dbooks.org



The Azure Automation DSC pull server automatically compiles the DSC definition you provide into a Managed Object Format (MOF) file. Digital certificates managed by Automation are used to encrypt the MOF file. DSC target servers receive the required public digital certificates and allow the LCM engine to decrypt and process the MOF file. Then the desired state can be applied to the server.

Because the MOF file defines the complete state of your servers, you should protect its content. If an attacker knows all the application components installed and the location of various configuration files and custom code, the chance that your servers will be compromised increases. Recent versions of PowerShell encrypt the entire MOF file. Azure Automation automatically generates the required digital certificates and keys when a target server is configured for DSC, which allows you to use encrypted MOF files seamlessly. Automation also encrypts the traffic between the DSC pull server and target nodes, not just the MOF file.

The compile process in Azure Automation both converts the DSC definition you provide to an MOF file and encrypts the MOF file with the digital certificates and keys. The process of compiling your DSC definition takes a few seconds but greatly secures your environment—just another example of Azure securing your resources by default!

7 To apply the configuration to your VM, select the nodes tab in the State configuration (DSC) windows; select Add; and choose the VM you created in previous steps.

8 Choose Connect.

9 From the Node Configuration Name drop-down menu, choose webserver.localhost.

10 Set the configuration mode to ApplyAndMonitor, and select OK.

It can take a minute or two to enable the VM to use the Azure PowerShell DSC pull server and apply the initial desired state.

11 When the Azure portal reports that the configuration is applied, select your resource group; then select the VM you created in the previous steps.

12 Did you open TCP port 80 for the VM when you created the VM? If not, create a network security group rule to allow the traffic and then open the public IP of the VM in a web browser. The DSC process installs the IIS web server, and the default web page loads, as shown in figure 18.6.

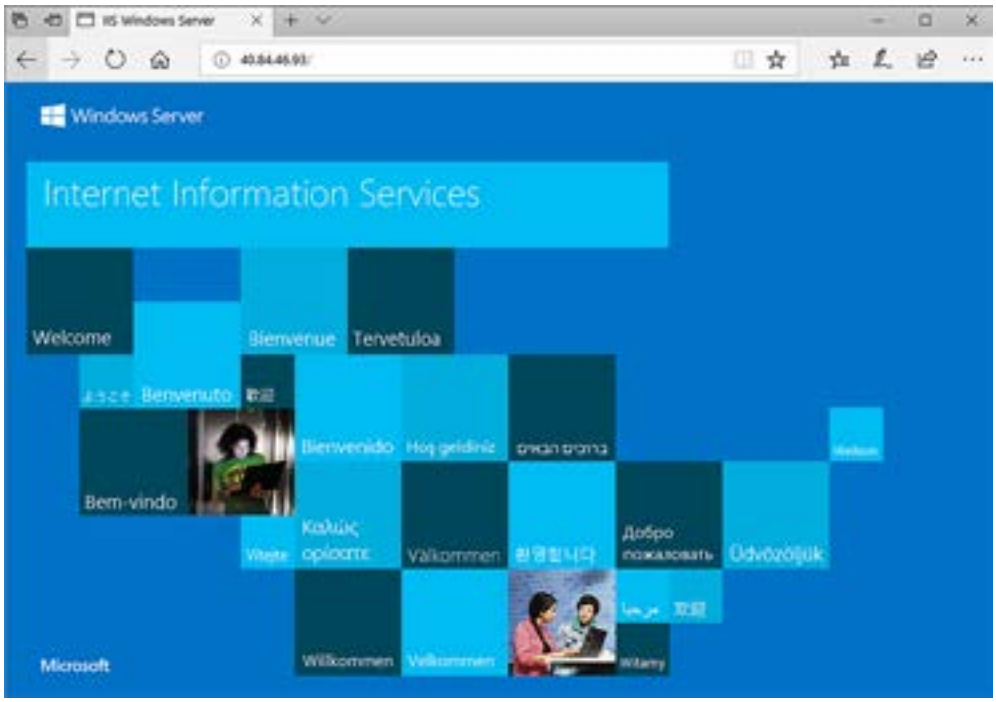


Figure 18.6 After the VM has been connected to Azure Automation DSC, the desired state is applied, and the IIS web server is installed.

This basic example of PowerShell DSC only installs the web server feature. You can use PowerShell DSC to configure the IIS web server or copy your application code to the VM and run the site. Complex DSC definitions can be used to get the VM ready to serve traffic to your pizza-store customers with no manual interaction. Again, think back to how you should design your applications to scale automatically; the VM can't wait for someone to log in and install and configure everything manually!

18.4 Lab: Using DSC with Linux

Just to prove that PowerShell DSC works on Linux servers, let's create an Ubuntu VM, install the necessary prerequisites, and then install a basic NGINX web server with DSC. In production, you could use a custom VM image that already had the management components installed, and then apply PowerShell DSC definitions as normal:

www.dbooks.org

1. PowerShell DSC for Linux has a few limitations in the Linux distros that it supports without additional configuration, so to keep this end-of-chapter lab exercise as simple as possible, create a CentOS 7.7 or later VM, and open port 80.
2. In your Azure Automation account, choose Modules from the menu on the left.
3. Select Browse Gallery, and then search for, select, and import the nx module for managing Linux DSC resources.
4. On your local computer, create a file name httpd.ps1, and type the following code:

```
configuration httpd {
    Import-DSCResource -Module nx
    Node localhost {
        nxPackage httpd {
            Name = "httpd"
            Ensure = "Present"
            PackageManager = "yum"
        }
        nxService httpd {
            Name = "httpd"
            State = "running"
            Enabled = $true
            Controller = "systemd"
        }
    }
}
```

5. Add a DSC configuration to the Azure Automation account, upload the httpd.ps1 file, and compile the configuration.
6. Add a DSC node to your Azure Automation account, select your CentOS VM, and then choose your httpd.localhost Node Configuration name.

   Again, it takes a minute or two for the VM to apply the desired configuration. You can view the list of connected VMs and their compliance status in the DSC nodes window. The VM reports Compliant when the LCM has accepted and applied the MOF file, but the commands to install and configure the required httpd packages inside the VM may take another minute or two.
7. Select your CentOS VM in the Azure portal, get its public IP address, and enter the IP address of your VM in a web browser to see the web server installed by DSC. If the website doesn't load, wait a minute or two for the install process to finish, and then refresh the page.

If you want to truly experience the brave new world of Microsoft and Linux, you can install PowerShell on your Linux VM. Complete the quick-setup steps at http://mng.bz/VgyP to understand how cross-platform PowerShell scripts can be!

19 Azure containers

Containers, Docker, and Kubernetes have gained a huge following in a few short years. In the same way that server virtualization started to change how IT departments ran their data centers in the mid-2000s, modern container tools and orchestrators are now shaking up how we build and run applications. There's nothing that inherently connects the growth of containers with cloud computing, but when combined, they provide a great way to develop applications with a cloud-native approach. Entire books have been written on Docker and Kubernetes, but let's go on a whirlwind introduction to see how you can quickly run containers in Azure. A powerful suite of Azure services is dedicated to containers, aligning more with the PaaS approach. You can focus on how to build and run your applications rather than how to manage the container infrastructure, orchestration, and cluster components.

In this chapter, we'll examine what containers are, how Docker got involved, and what Kubernetes can do for you. To see how to run either a single container instance or multiple container instances quickly in a cluster, we'll explore Azure Container Instances (ACI) and Azure Kubernetes Service (AKS).

19.1 What are containers?

There's been a huge wave of interest in adoption of containers over the past few years, and I'd be impressed if you haven't at least heard of one company that has led this charge: Docker. But what is a container, exactly, and what does Docker have to do with it?

First, let's discuss a traditional virtualization host that runs VMs. Figure 19.1 is like the diagram in chapter 1, in which each VM has its own virtual hardware and guest OS.

www.dbooks.org

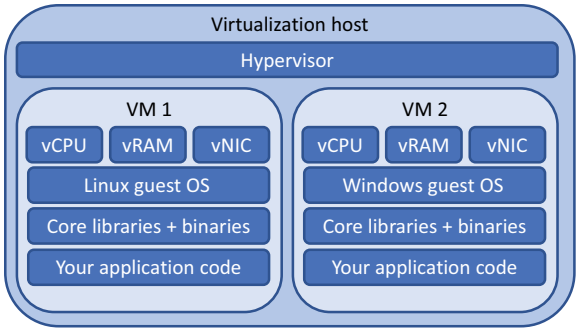


Figure 19.1 With a traditional VM infrastructure, the hypervisor on each virtualization host provides a layer of isolation by providing each VM its own set of virtual hardware devices, such as a virtual CPU, virtual RAM, and virtual NICs. The VM installs a guest OS, such as Ubuntu Linux or Windows Server, that can use this virtual hardware. Finally, you install your application and any required libraries. This level of isolation makes VMs very secure but adds a layer of overhead in terms of compute resources, storage, and startup times.

A container removes the virtual hardware and guest OS. All that's included in a container are the core applications and libraries required to run your app, as shown in figure 19.2.

Many VMs can run on a single hypervisor, each VM with its own virtual guest OS, virtual hardware, and application stack. The hypervisor manages requests from the virtual hardware of each VM, schedules the allocation and sharing of those physical hardware resources, and enforces the security and isolation of each VM. The work of the hypervisor is shown in figure 19.3.

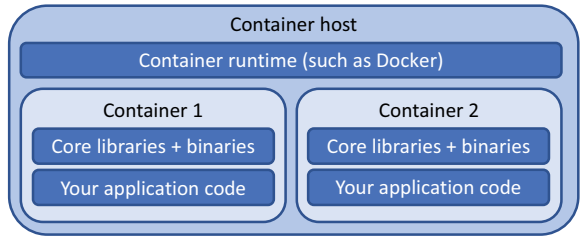


Figure 19.2 A container contains only the core libraries, binaries, and application code required to run an app. The container is lightweight and portable because it removes the guest OS and virtual hardware layer, which also reduces the on-disk size of the container and startup times.

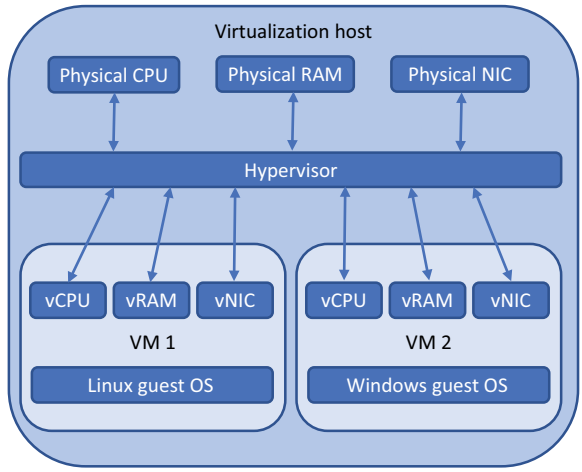


Figure 19.3 In a traditional VM host, the hypervisor provides the scheduling of requests from the virtual hardware in each VM onto the underlying physical hardware and infrastructure. The hypervisor typically has no awareness of what specific instructions the guest OS is scheduling on the physical CPU time—only that CPU time is required.

Multiple containers can also run on a single host. The container host receives the various system calls from each container and then schedules the allocation and distribution of those requests across a shared base kernel, OS, and hardware resources. Containers provide a logical isolation of application processes. The work of the container runtime is shown in figure 19.4.

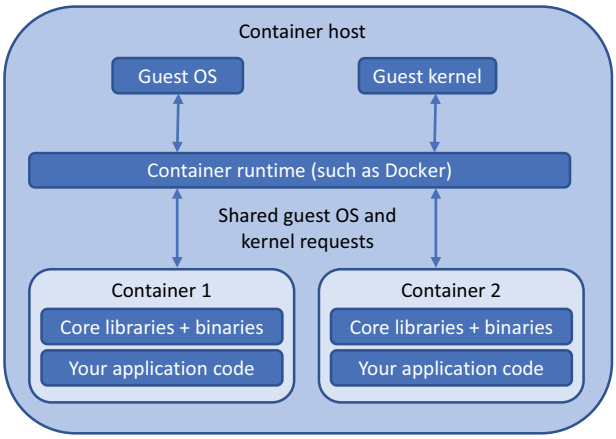


Figure 19.4 Containers have a common guest OS and kernel. The container runtime handles the requests from the containers to the shared kernel. Each container runs in an isolated user space, and some additional security features protect containers from each other.

Containers are typically much more lightweight than VMs. Containers can start faster than VMs, often in a matter of seconds rather than minutes. The size of a container image is typically only tens or hundreds of MBs compared with many tens of GBs for VMs. Security boundaries and controls are still in place, but it's important to remember that each container technically shares the kernel with other containers on the same host.

Try it now

It takes a few minutes to create an AKS cluster for use in the upcoming exercises, so complete the following steps and then continue reading the chapter:

1. Open the Azure portal, and choose the Cloud Shell icon from the top menu.
2. Create a resource group. Provide a name, such as `azuremolchapter19`, and a location, such as `eastus`. Region availability of AKS may vary, so pick a major region such as `eastus` or `westeurope`. (For an up-to-date list of region availability, see https://azure.microsoft.com/regions/services.)

```
az group create --name azuremolchapter19 --location eastus
```

www.dbooks.org

3 To create a Kubernetes cluster, specify `--node-count` as `2`, and use scale sets and Availability Zones (which you learned about in previous chapters):

```
az aks create \
  --resource-group azuremolchapter19 \
  --name azuremol \
  --node-count 2 \
  --vm-set-type VirtualMachineScaleSets \
  --zones 1 2 3 \
  --no-wait
```

The final `--no-wait` parameter returns control to Cloud Shell while the rest of your cluster is created. Keep reading while the cluster is deployed.

Docker joined the container party with a set of tools and standard formats that defined how to build and run a container. Docker builds on top of existing Linux and Windows kernel-level features to provide a portable, consistent container experience across platforms. A developer can build a Docker container on their laptop that runs macOS; validate and test their app; and then run exactly the same Docker container, without modification, in a more traditional Linux or Windows-based server cluster on-premises or in Azure. All the required application binaries, libraries, and configuration files are bundled as part of the container, so the underlying host OS doesn't become a design factor or constraint.

The importance of Docker shouldn't be missed here. The terms *container* and *Docker* are often used interchangeably, although that's not technically accurate. Docker is a set of tools that helps developers build and run containers in a consistent, reliable, and portable manner. The ease of using these tools led to rapid adoption and brought the underlying container technology that had been around in one shape or another for more than a decade into the mainstream. Developers embraced containers and the Docker platform, and IT departments have had to play catch-up ever since.

Docker participates in the Open Container Initiative. The format and specifications that Docker defined for how a container should be packaged and run were some of the founding principles for this project. Docker's work has continued and been built upon by others. Large contributors in the container space include IBM and Red Hat, which contributed some of the core designs and code that powers the current container platforms. The Open Container Initiative and design format for container packaging and runtimes are important because they let each vendor layer its own tools on top of the common formats, allowing you to move the underlying container between platforms and have the same core experience.

19.2 The microservices approach to applications

If containers offer a concept of isolation similar to VMs, can you run the same kind of workloads you do in a VM? Well, yes and no. Just because you can do something doesn't necessarily mean that you should! Containers can be used to run whatever workloads you're comfortable with, and there are benefits in terms of portability and orchestration features that we'll examine in section 19.4. To maximize the benefits of

containers and set yourself up for success, take the opportunity to adopt a slightly different mental model when you start working with containers. Figure 19.5 compares the traditional application model with a microservices approach.

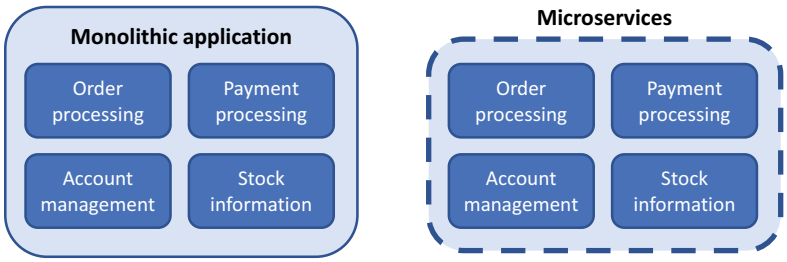


Figure 19.5 In a traditional monolithic application, the entire application runs as a single application. The application may have various components, but it runs from a single install, and it's patched and updated as a single instance. With microservices, each component is broken down into its own application service and unit of execution. Each component can be updated, patched, and scaled independently of the others.

A standard VM includes a full guest OS install, such as Ubuntu or Windows Server. This base OS install includes hundreds of components, libraries, and tools. Then you install more libraries and applications, such as for the NGINX web server or Microsoft SQL Server. Finally, you deploy your application code. This VM typically runs a large part, if not all, of the application. It's one big application install and running instance. To improve performance, you may add more memory or CPU to the VM (vertical scaling, discussed in previous chapters) or increase the number of instances that run your application (horizontal scaling, as with scale sets). Creating multiple application instances works only if your application is cluster-aware, and it often involves some form of shared storage to enable a consistent state across the application instances. This traditional form of deployment is called a *monolithic* application.

A different approach to how you design, develop, and run applications is to break things down into smaller, bite-size components. This is a *microservices* approach to application development and deployment. Each microservice is responsible for a small part of the wider application environment. Microservices can grow, scale, and be updated independently of the rest of the application environment.

Although this model may offer challenges while development and IT teams learn to adopt a different way to build and deploy applications, containers are a great fit for the microservice approach. Developers are empowered to deploy smaller, more incremental updates at a quicker pace than in the monolithic approach to application development. Microservices and containers are also a great fit for continuous integration and continuous delivery (CI/CD) workflows, which make it easier to build, test, stage, and deploy updates. Your customers receive new features or bug fixes faster than they would otherwise, and ideally, your business grows as a result.

www.dbooks.org

Microservices with Azure Service Fabric

This chapter focuses on Docker containers and orchestration with Kubernetes, but a similar Azure service moves application development toward a microservices model. Azure Service Fabric has been around for several years and was historically a Windows-centric approach to building applications in which each component was broken down into its own microservice. Service Fabric keeps track of where each microservice component runs in a cluster, allows the services to discover and communicate with each other, and handles redundancy and scaling.

Many large Azure services use Service Fabric under the hood, including Cosmos DB. That should give you a sense of how capable and powerful Service Fabric can be! Service Fabric itself runs on top of virtual machine scale sets. You know a thing or two about scale sets by now, right?

The Service Fabric platform has matured, and now it can handle both Windows and Linux as the guest OS, so you can build your app with any programming language you're comfortable with. Here's another example of choice in Azure: you have the flexibility to choose how you want to manage and orchestrate your container applications. Both Service Fabric and AKS have excellent benefits and use cases.

As a good starting point, if you currently develop, or would like to develop, microservices outside containers, Service Fabric is a great choice. Applications designed around the actor model are also a great fit, as Service Fabric was originally built with this programming model in mind. Service Fabric provides a unified approach to handling both more traditional microservices applications and container-based applications. If you choose to adopt containers for other workloads, you can use the same Service Fabric management tools and interface to manage all your application environments.

For a more container-focused application approach from the get-go, AKS may be a better choice, with the growth and adoption of Kubernetes providing a first-class container experience. You can run with Linux and Windows containers in AKS.

19.3 Azure Container Instances

Now that you understand a little more about what containers are and how you can use them, let's dive in and create a basic instance of the pizza store. This example is the same one used in earlier chapters, in which you created a basic VM that ran your website or deployed the app to web apps. In both of those cases, you had to create the VM or web app, connect to it, and then deploy a basic web page to it. Can the power of containers make your life that much easier? Absolutely!

A neat service called Azure Container Instances (ACI) lets you create and run containers in a matter of seconds. There are no up-front network resources to create and configure, and you pay for each container instance by the second. If you've never used containers and don't want to install anything locally on your computer, ACI is a great way to try the technology.

To see how you can run your pizza store quickly, let's create a container instance. It takes only one command to run a container instance, but figure 19.6 shows how you bring together many components to make this happen behind the scenes. We'll look at the components of a Dockerfile and Docker Hub after you have the container instance up and running.

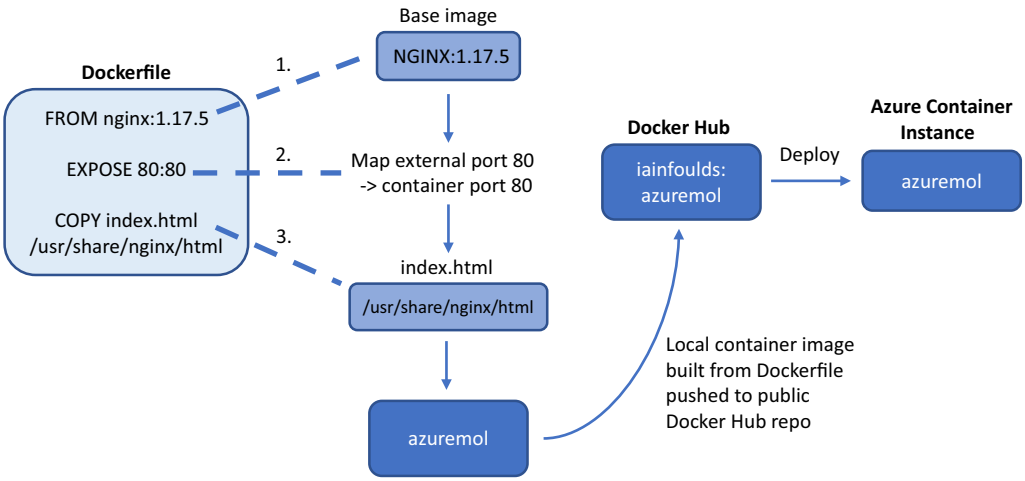


Figure 19.6 A Dockerfile was used to build a complete container image: azuremol. This image was pushed to an online public registry called Docker Hub. Now you can create a container instance using this prebuilt public image from Docker Hub, which provides a ready-to-run application image.

Try it now

To create an Azure container instance that runs a basic website, complete the following steps:

1. Open the Azure portal, and choose the Cloud Shell icon from the top menu.
2. Create a container instance, and specify that you want to have a public IP address and to open port 80:

```
az container create \
  --resource-group azuremolchapter19 \
  --name azuremol \
  --image iainfoulds/azuremol \
  --ip-address public \
  --ports 80
```

This exercise uses a sample image that I've created for you, which we'll examine a little more when the container is up and running.

www.dbooks.org

3 To see what was created, look at the output of the command to create the container.

In the Events section, you can watch as the image is pulled (downloaded) from Docker Hub, a container is created, and the container is started.

Some CPU and memory reservations are also assigned, which can be adjusted if needed. A public IP address is shown, along with some information about the container such as the provisioning state, OS type, and restart policy.

4 To open the basic website that runs in the container, you can query for just the assigned public IP address:

```
az container show \
  --resource-group azuremolchapter19 \
  --name azuremol \
  --query ipAddress.ip \
  --output tsv
```

5 Open the public IP address of your container instance in a web browser. The basic pizza store should be displayed, as shown in figure 19.7.

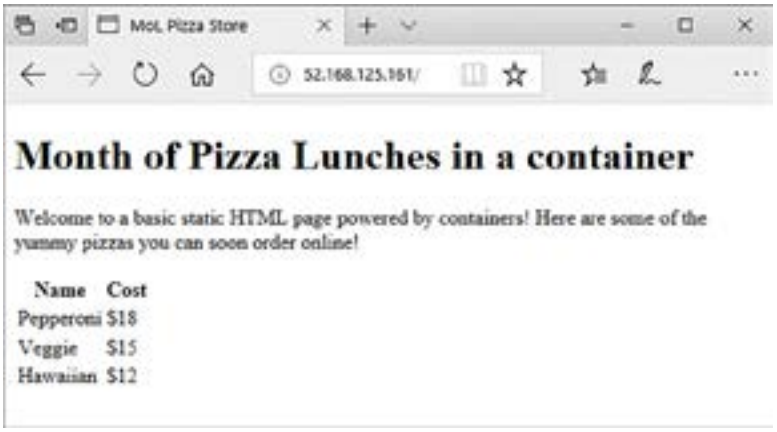


Figure 19.7 When you create a container instance, the pizza-store website runs without any additional configuration. All the configuration and content are included within the container image. This quick exercise highlights the portability and power of containers; when the container image has been prepared, your app is up and running as soon as a new container instance is deployed.

Let's examine the container image. I don't want to get too far into the weeds of Docker and how to build container images, but it's important to understand where this image came from and how it runs the website without any additional configuration.

The image is built from a configuration definition called a *Dockerfile*. In a Dockerfile, you define what the base platform is, any configuration you wish to apply, and any

commands to run or files to copy. Dockerfiles can be, and often are, more complex than the following example, which was used to build the azuremol sample container:

```
FROM nginx:1.17.5

EXPOSE 80:80

COPY index.html /usr/share/nginx/html
```

When this Dockerfile was used to build a Docker container image, NGINX was used as the source image, and the sample web page was copied to it. Then this container was pushed to Docker Hub, an online public repository that Docker provides to share and deploy containers. To deploy the container instance, you provided iainfoulds/azuremol as the container image to use. Azure looked in Docker Hub and found a repository named iainfoulds and, within it, an image named azuremol.

Let's examine each line of the Dockerfile:

- `FROM nginx:1.17.5`—In previous chapters, you created a basic VM, connected to it with SSH, and then manually installed the NGINX web server. In the example Dockerfile, all of that is accomplished in one line. This line says to base the container on an existing container image that's preinstalled with NGINX. The `1.17.5` is the version of the public NGINX container image to use; that's the latest at the time of writing. It's good practice to include a specific version number. If you don't include a version number, the latest version is always used. This sounds good in theory, but microservices applications can scale to a pretty large number of active containers, so to make sure that you have a consistent environment, you want to control the exact version number of each component in use.
- `EXPOSE 80:80`—To allow access to your VM in previous chapters, you created an NSG rule that allowed port 80. In the Dockerfile, this line tells the container to open port 80 and map it to the internal port 80. When you created your container instance with `az container create`, you also specified that the Azure platform should permit traffic with `--ports 80`. That's all the virtual networking you have to think about!
- `COPY index.html /usr/share/nginx/html`—The final part is to get your application into the container. In previous chapters, you used Git to obtain the sample pizza-store web page and then push it to your web app. With the Dockerfile, you `COPY` the index.html file to the local /usr/share/nginx/html directory in the container. That's it!

For your own scenarios, you can define a Dockerfile that uses a different base image, such as Node.js or Python. Then you install any additional supporting libraries or packages required; pull your application code from source control, such as GitHub; and deploy your application. This Dockerfile would be used to build container images that are stored in a private container registry, not a public Docker Hub repo like that in the example.

www.dbooks.org

Azure Container Registry

You may think that Docker Hub sounds great. Does Azure have such a wonderful thing? It does! Because you need to create a Dockerfile and build a container image, unfortunately, it's not a two-minute exercise, and there's a lot to cover in this chapter. You can easily integrate Azure Container Registry (ACR) and AKS, so the two services work well together. You can build your own images from a Dockerfile in Cloud Shell, though, and I encourage you to explore this if you have time. Azure Container Registry (ACR) is the route I'd choose to store my container images, for a couple of reasons:

- It's a private registry for your container images, so you don't need to worry about potential unwanted access to your application files and configuration. You can apply the same RBAC mechanisms we discussed in chapter 6. RBAC helps you limit and audit who has access to your images.
- Storing your container images in a registry in Azure means that your images are right there in the same data centers as the infrastructure used to run your container instances or clusters (which we'll look at in section 19.4.1). Although container images should be relatively small (often only tens of MB in size), they can add up if you keep downloading those images from a remote registry.

ACR also provides built-in replication and redundancy options you can use to place your containers close to where you deploy and run them for users to access. This region locality is similar to how you used Cosmos DB global replication in chapter 10 to make those milliseconds count and provide your customers with the quickest possible access time to your applications.

If all this sounds exciting, check out the ACR quick-start page to get up and running with your own private repository in a few minutes: http://mng.bz/04rj.

19.4 Azure Kubernetes Service

Running a single container instance is great, but that doesn't give you much redundancy or ability to scale. Remember how we spent entire chapters earlier in the book talking about how to run multiple instances of your application, load balance, and scale them automatically? Wouldn't it be great to do the same with containers? That's where you need a container orchestrator.

As the name implies, a *container orchestrator* manages your container instances, monitors their health, and can scale as needed. Orchestrators can, and often do, handle a lot more, but at a high level, a primary focus is handling all the moving parts involved in running a highly available, scalable, container-based application. There are a few container orchestrators, such as Docker Swarm and Distributed Cloud Operating System (DC/OS), but one has risen above the rest to become the go-to orchestrator of choice: Kubernetes.

Kubernetes started as a Google-led and -sponsored open source project that grew out of the company's internal container orchestration tooling. Widely accepted by the open source community, Kubernetes is one of the largest and fastest-growing open

source projects on GitHub. Many large technology companies, including Red Hat, IBM, and Microsoft, contribute to the core Kubernetes project.

In this section, let's take the same sample web app from the previous exercise with ACI to run a redundant, scalable deployment in Kubernetes. You'll end up with a few components, as shown in figure 19.8.

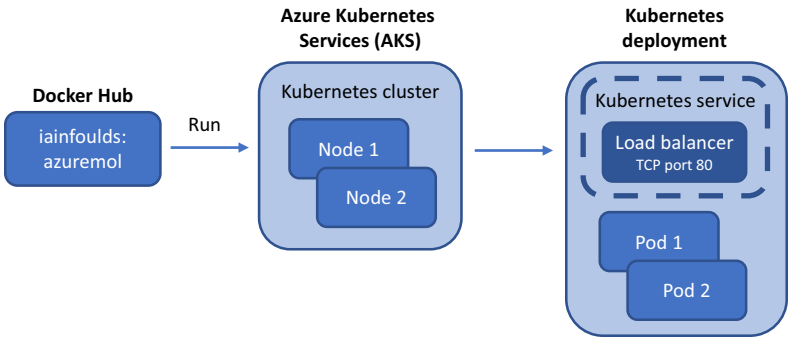


Figure 19.8 Your sample container from Docker Hub runs on a two-node Kubernetes cluster that you create in AKS. The Kubernetes deployment contains two logical pods, one on each cluster node, with a container instance running inside each pod. Then you expose a public load balancer to allow your web app to be viewed online.

19.4.1 Creating a cluster with Azure Kubernetes Services

In chapter 9, we looked at how virtual machine scale sets reduce the complexity of deploying and configuring the underlying infrastructure. You say how many VM instances you want in a scale set, and the rest of the network, storage, and configuration is deployed for you. AKS works in much the same way to offer a resilient, scalable Kubernetes cluster, with management handled by the Azure platform. Scale sets can be used for the underlying VMs that run in the AKS cluster, and those VMs can be distributed across Availability Zones. Azure load balancers, also zone-redundant, are used. Basically, AKS brings together several of the infrastructure components and best practices you've learned about so far in this book!

Try it now

To view the information on your AKS cluster, complete the following steps:

1. Open the Azure portal, and choose the Cloud Shell icon from the top menu.
2. Earlier in the chapter, you created a Kubernetes cluster. The process took a few minutes, but I hope it's ready now! Look at the status of the cluster as follows:

```
az aks show \
  --resource-group azuremolchapter19 \
  --name azuremol
```

www.dbooks.org

The `provisioningState` near the end should report `Succeeded`.

3 If your cluster is ready, obtain a credentials file that allows you to use the Kubernetes command-line tools to authenticate and manage resources:

```
az aks get-credentials \
  --resource-group azuremolchapter19 \
  --name azuremol
```

That's all it takes to get Kubernetes up and running in Azure! You may be wondering, "Can't I just build my own cluster with VMs or scale sets, and manually install the same Docker and Kubernetes components?" You absolutely can. The parallel is the IaaS and PaaS approach of VMs versus web apps. The web app approach offers many benefits: you worry only about high-level configuration options, and then you upload your application code. A managed Kubernetes cluster, as offered by AKS, reduces the level of complexity and management; your focus becomes your applications and your customers' experience.

In the same way that you may choose VMs over web apps, you may choose to deploy your own Kubernetes cluster rather than use AKS. That's fine; both approaches end up using the same Azure services components. VMs, scale sets, load balancers, and NSGs are all topics you've learned about in previous chapters, and all are still present with AKS clusters, although they're abstracted away. From a planning and troubleshooting perspective, you should have the skills to understand what's happening under the hood to make the managed Kubernetes offering work. Your comfort level, and how much time you want to spend managing the infrastructure, will help guide your decision-making process as you build a new application around containers in Azure.

19.4.2 Running a basic website in Kubernetes

You created a Kubernetes cluster in section 19.4.1, but there's no application running. Let's change that! You need to create the Kubernetes deployment that you saw earlier in figure 19.8; see figure 19.9.

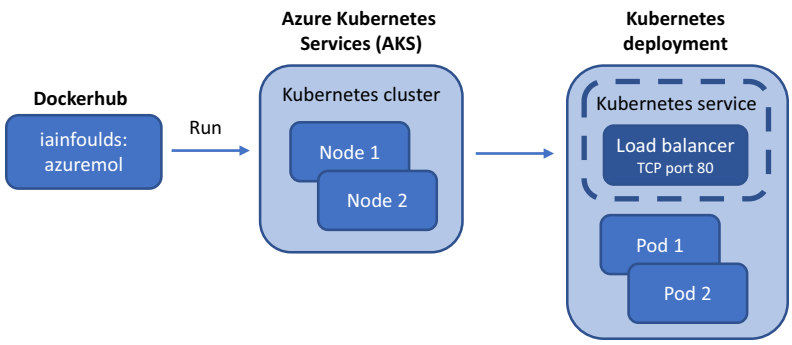


Figure 19.9 With the Kubernetes cluster created in AKS, you can create a Kubernetes deployment and run your app. Your container runs across both nodes, with one logical pod on each node; you need to create a Kubernetes service that exposes a public load balancer to route traffic to your app.

Try it now

To deploy an application to your Kubernetes cluster, complete the following steps:

1. You interact with a Kubernetes cluster by using a command-line utility called `kubectl`. Use the same iainfoulds/azuremol container image from Docker Hub that you ran as a container instance:

```
kubectl run azuremol \
  --image=docker.io/iainfoulds/azuremol:latest \
  --port=80
```

   It may take a minute or so to download the container image from Docker Hub and start the application in Kubernetes. The application runs in a *pod*: a logical construct in Kubernetes that houses each container.

2. Pods can contain additional helper components, but for now, monitor the status of your container by looking at the pod:

```
kubectl get pods --watch
```

Even when the status of the pod reports as `Running`, you won't be able to access your application. The container instance you created earlier could route traffic over a public IP address directly to that one instance, but what do you think is needed for a Kubernetes cluster to route traffic to containers? If you guessed a load balancer, congratulations! Right now, you have only one pod: a single container instance. You'll scale out the number of pods in the end-of-chapter lab, and for that to work, you need a way to route traffic to multiple instances. So let's tell Kubernetes to use a load balancer.

Here's where the integration between Kubernetes and Azure becomes cool. When you tell Kubernetes that you want to create a load balancer for your containers, under the hood, Kubernetes reaches back into the Azure platform and creates an Azure load balancer. This Azure load balancer is like the one you learned about in chapter 8. There are frontend and backend IP pools and load-balancing rules, and you can configure health probes. As your Kubernetes deployment scales up or down, the load balancer is automatically updated as needed.

Try it now

To expose your application to the internet, complete the following steps:

1. Tell Kubernetes that you want to use a load balancer, and add a rule to distribute traffic on port 80:

```
kubectl expose deployment/azuremol \
  --type="LoadBalancer" \
  --port 80
```

www.dbooks.org

2 As before, watch the status of your service deployment:

```
kubectl get service azuremol --watch
```

When the external public IP address is assigned, the Azure load balancer has finished deploying, and the Kubernetes cluster and nodes are connected.

3 Open the public IP address of your service in a web browser to see your web application running.

Application deployments in Kubernetes are often much more involved than this basic example. You typically define a service manifest, similar to a Resource Manager template, that defines all the characteristics of your application. These properties can include the number of instances of your application to run, any storage to attach, load-balancing methods and network ports to use, and so on. In the real world, you don't even do this manually; a CI/CD system like Azure DevOps or Jenkins automates the deployments of applications and services directly inside the AKS cluster. What's great about AKS is that you don't have to worry about Kubernetes installation and configuration. As with other PaaS services, such as web apps and Cosmos DB, you bring your applications, and let the Azure platform handle the underlying infrastructure and redundancy.

Keeping it clean and tidy

Remember to clean up and delete your resource groups so that you don't end up consuming lots of your free Azure credits. As you start to explore containers, it becomes even more important to pay attention to what Azure resources you leave turned on. A single web app doesn't cost much, but a five-node AKS cluster and a few container instances with georeplicated Azure Container Registry images sure can!

ACI instances are charged for by the second, and the cost adds up quickly if they're left running for days or weeks. An AKS cluster runs a VM for each node, so if you scale up and run many VMs in your cluster, you're paying for one VM for each node.

There's no charge for the number of containers that each of those AKS nodes runs, but as with any VM, an AKS node gets expensive when left running. What's great about Kubernetes is that you can export your service configurations (the definition for your pods, load balancers, autoscaling, and so on) to deploy them elsewhere. As you build and test your applications, you don't need to leave an AKS cluster running; you can deploy a cluster as needed and deploy your service from a previous configuration.

AKS clusters can scale up and down, as you'll see in the end-of-chapter lab exercise. You can also configure autoscaling that does this scaling for you depending on the load. It's the same kind of autoscaling we looked at in chapter 9 for scale sets and web apps. Are you starting to see everything coming together in Azure?

This chapter has been a warp-speed introduction to containers and Kubernetes, so don't worry if you feel a little overwhelmed right now! Manning has several great books, such as *Learn Docker in a Month of Lunches*, by Elton Stoneman (https://livebook.manning.com/book/learn-docker-in-a-month-of-lunches), and *Kubernetes in*

(continued)

Action, by Marko Luksa (https://livebook.manning.com/book/kubernetes-in-action), that can help you dive further into Docker, microservices application development, and Kubernetes. Check them out if this chapter sounds exciting and you want to explore further!

The examples in this chapter used Linux VMs for the AKS cluster nodes and then ran Linux containers for NGINX. Containers get a little tricky in that you can run Linux containers only on Linux nodes, for example. As you learned at the start of the chapter, containers share the guest OS and kernel. So you can't run Windows containers on a Linux node. In general, you can't run Linux containers on a Windows node either. Some cool technical trickery is involved, but in general, the container and underlying node OS should match.

What's great in AKS is that you can run both Linux and Windows nodes, so you can run both Linux and Windows containers! You do need to pay a little attention to how these different containers are scheduled on the different node OSes, but this approach greatly expands what applications and services you can run in AKS.

19.5 Lab: Scaling your Kubernetes deployments

The basic example in this chapter created a two-node Kubernetes cluster and a single pod that runs your website. In this lab, explore how you can scale the cluster and number of container instances. This example is a basic one, but the more nodes you have, the more container instances you can run, which is especially useful the more applications you need to run in your cluster.

1. You can see how many nodes are in your Kubernetes cluster with `kubectl get nodes`. Scale up your cluster to three nodes:

   ```
   az aks scale \
     --resource-group azuremolchapter19 \
     --name azuremol \
     --node-count 3
   ```

   It takes a minute or two to scale up and add the new node.

2. Use `kubectl` again to see the status of your nodes. When you scale up a node, Kubernetes doesn't create any additional container instances for your applications automatically, so you don't immediately get any benefit from the additional compute resources the new node provides.

3. Look at your current deployment with `kubectl get deployment azuremol`. Only one instance was created earlier. This sample application isn't making the most of the new node you added to the cluster in step 1. Scale up to five instances, or *replicas*:

   ```
   kubectl scale deployment azuremol --replicas 5
   ```

www.dbooks.org

4. Use `kubectl` again to examine the deployment. Look at the pods—the running container instances—with `kubectl get pods`. Within a matter of seconds, all those additional replicas were started and connected to the load balancer.
5. Use `kubectl get pods -o wide` to see what nodes the pods run on. Look at the last number in the node name, which indicates what node in the scale set is used. The pods should be distributed across all nodes in your cluster. As other applications would scale up the number of containers in a similar way, you can start to maximize the use of the compute resources across all nodes in the cluster.

20 Azure and the Internet of Things

For me, one of the most exciting areas of technology in the past few years is the Internet of Things (IoT). I don't quite believe that a dishwasher or fridge needs to be connected to the internet just yet, and there are valid privacy concerns about a TV or audio device that's permanently connected to the internet and always listening for the sound of your voice to issue a command. There are a lot of a practical applications for IoT devices, however. You could have manufacturing equipment report on its health status, generate maintenance alerts, and allow operators to understand its efficiency across multiple factories around the world. A trucking company could stream telemetry from its vehicles about loads being carried and average driving times, and be able to reroute drivers as needed more intelligently. Shipping companies could track each container and help their customers manage their supply chain better by knowing where their resources are.

In Azure, you can integrate IoT devices with a range of services. Azure Web Apps can provide a frontend for your data to be visualized, Storage can be used to log data streamed from devices, and serverless features such as Azure Logic Apps (discussed in the next and final chapter) can process the data received.

In this chapter, we'll examine what IoT is and how to use Azure IoT Hub to centrally manage and collect data from devices. Then you'll then see how to use an Azure web app to view real-time data from an IoT device.

20.1 What is the Internet of Things?

Interest in IoT has grown considerably the past few years, but it's a vague term that can be applied to many scenarios. At a basic level, IoT is an approach in which many interconnected devices—typically, small, low-cost electronic devices—connect to

www.dbooks.org

central systems and applications. The connected devices usually report information that they collect from attached sensors or inputs. Then this information can be processed by a central system—perhaps with AI or ML, as discussed in chapter 17—and carry out appropriate actions. Figure 20.1 shows a high-level approach to IoT.

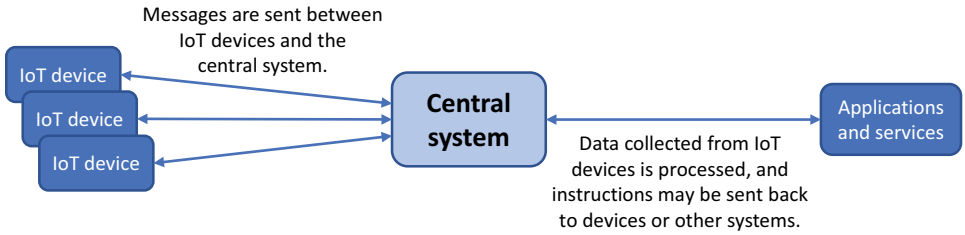


Figure 20.1 Messages are sent between many connected IoT devices and a central system. Your applications and services can process the data received and send device instructions to perform additional actions in response to their collected data.

Examples of IoT in action include the following:

- *Parking garage* —A small sensor above each parking bay detects whether a vehicle is parked there. A light above each bay can illuminate green if the parking bay is empty or red if it's occupied. Drivers entering the parking garage can see real-time information boards on each floor that let them know how many open parking spots there are. The red and green lights above each bay help drivers quickly determine the location of open spots as they drive along each aisle.
- *Factory*—Machinery on a factory floor can report information on operating output, consumable levels, and maintenance needs. Then a central system can schedule a maintenance technician to repair equipment proactively or resupply consumables, which reduces any downtime in the production line. When combined with AI and ML, maintenance schedules can be predicted, and the correct amount of supplies or raw materials can be delivered just before they're needed in production.
- *Transportation*—Public transportation buses or trains can include GPS sensors that report on location and speed. Ticketing information can be collected to report on how many people are being transported. Passenger information boards at a train station or bus terminal can provide real-time information about when each vehicle will arrive. When this technology is combined with AI and ML, waiting passengers can receive suggestions for alternative routes based on traffic conditions, delays, or heavy passenger volume.

IoT often works alongside other applications and services. The factory and transportation scenarios could use AI and ML to better inform production decisions or make

suggestions to passengers. Web applications can use information received from IoT devices to provide access from mobile devices or generate alerts and notifications. Data received from IoT devices could be logged to a database system such as Azure Cosmos DB and then processed by business-intelligence applications and generate reports.

More future-looking ideas for IoT include things like your refrigerator's sensing food levels and generating a shopping list or even ordering food from a local grocery store. Your car could report data to the dealership, which could have any required parts or consumables ready when you take the vehicle in for service. Or what if, when your alarm clock goes off to wake you up in the morning, your coffeemaker turns on and gets ready for breakfast?

One big area of concern with IoT is device security. With so many devices outside your primary network infrastructure and often connected to the public internet, being able to provision, maintain, and update those devices is a challenge. Many IoT devices are low-power, simple electronics that may not have the storage or processing capabilities to update themselves with security and application updates the way traditional desktops or laptops do. It's not enough to deploy a bunch of IoT devices, especially consumer-level devices, without a plan to secure them adequately and provide updates and maintenance.

These security concerns shouldn't stop you from building applications and services that use IoT devices. IoT brings a new set of challenges to traditional device maintenance, but there are solutions that allow you to provision and maintain devices centrally, as well as provide secure device communication.

By now, I'm sure you may have guessed that Azure has such an IoT solution! It offers a suite of IoT services. Let's see how you can start to explore IoT with Azure.

Accelerating your Azure IoT deployments

This chapter focuses on Azure IoT Hub, a service that lets you provision and connect IoT devices to build your own solutions. You can define how those IoT devices connect, what users or applications can access their data, and secure connectivity. How to build and deploy the application infrastructure to connect everything together is up to you.

Azure IoT solution accelerators are prebuilt key scenarios, such as remote monitoring of devices or a connected factory. Accelerators deploy common Azure services such as IoT Hub, Web Apps, Cosmos DB, and Storage, and run a sample application that integrates all these different services.

You still need to customize the application for your own environment, IoT devices in use, and the data to be collected and monitored, but IoT solution accelerators give you a great framework to get started. Whereas IoT Hub creates a way for you to connect IoT devices to Azure and then leaves you to deploy any additional services that you need, IoT solution accelerators deploy prebuilt solutions that use the most common Azure services you'd use.

If you get hooked on IoT after this chapter and want to learn more, the Azure IoT solution accelerators are a great way to see the possibilities of what Azure can offer. As

www.dbooks.org

we've discussed throughout this book, Azure is way more than just one or two independent services. You can deploy many services together to provide the best application experience possible for your customers.

20.2 Centrally managing devices with Azure IoT Hub

Azure IoT Hub lets you centrally manage, update, and stream data from IoT devices. With this service, you can perform actions such as configuring application routes for data received from devices, provisioning and managing certificates to secure communication, and monitoring health with Azure diagnostics and metrics. You can connect your IoT devices to other Azure services and applications to let them send and receive data as part of a wider solution. As with all things in Azure, access can be controlled with RBAC, and diagnostic data can be centrally collected for troubleshooting and monitoring or alerts. Figure 20.2 outlines how IoT Hub acts as the central place for IoT devices to connect to the wider Azure services and applications.

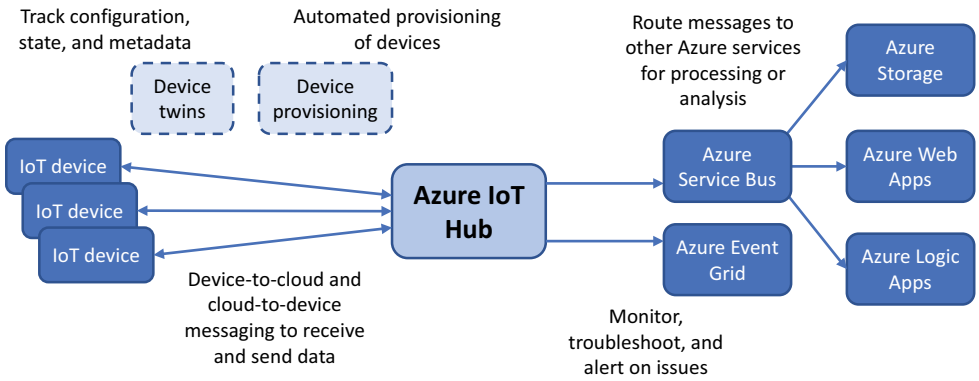


Figure 20.2 With an IoT hub, you can centrally provision and manage many IoT devices at scale. Two-way communication exists between devices and Azure to read and write data. You can process data received from devices and route it to other Azure services, such as Web Apps and Storage. To monitor and troubleshoot issues, you can route information to Azure Event Grid, which we'll look at in chapter 21, and then link to other monitoring solutions.

You control access to an IoT hub with shared access policies. These policies are like user accounts and permissions. Default policies exist that allow devices and services to connect to the IoT hub, or to read and write information from the device registry that tracks connected IoT devices and security keys. Each policy can be assigned one or more of the following permissions:

- Registry read
- Registry write

- Service connect
- Device connect

Shared access keys are used by applications and services to connect to an IoT hub. As with Storage (discussed in chapter 4), shared access keys allow you to define connection strings to identify the host, access policy, and access key. A connection string combines the access key, access policy type, and the IoT hub hostname. Here's a sample IoT hub connection string:

```
HostName=azuremol.azure-devices.net;SharedAccessKeyName=registryRead;
➥SharedAccessKey=6be2mXBVN9B+UkoPUMuwVDtR+7NZVBq+C7A1xCmQGAb=
```

Primary and secondary keys exist and can be rotated and updated for security purposes, just like updating passwords regularly. Solutions such as Azure Key Vault (discussed in chapter 15) are great ways to track and store these keys for applications to obtain when needed. This approach to key management means you can rotate access keys frequently without the need to also update all of your application code.

Digital certificates can be stored in an IoT hub and automatically provisioned to IoT devices. Remember that IoT devices are often outside your core infrastructure and may connect directly over the internet without any form of secure network connection, like a VPN. Make sure that all the data between your devices and the IoT hub is encrypted using SSL/TLS connections. Azure Key Vault can generate and store SSL certificates that are then added to the IoT hub. Or you can use an existing certificate authority to request and issue certificates. The important thing is to make sure all communication between your IoT devices and Azure is encrypted. Otherwise, you're likely to receive an error.

IoT hub routes let you send data from IoT devices to other Azure services. You can define criteria, such as the message content containing a certain keyword or value, and then route the messages to be stored in Azure Storage or processed by a web app. In one of the following exercises, you'll simulate a basic temperature sensor connected to an IoT device. You could define a route in the IoT hub to watch the incoming data and, if the recorded temperature exceeded 30°C, route the data to a logic app to send an email alert. We'll discuss the wonderful world of serverless computing and logic apps in chapter 21.

Living on the Edge

In this chapter, we focus on Azure IoT Hub. Another service, Azure IoT Edge, lets you run services such as Azure Functions and Stream Analytics in your local environment. Rather than having all of your IoT devices streaming data processed centrally in Azure, you can process the data within each location.

Azure IoT Edge runs applications and services in containers (discussed in chapter 19). The use of containers allows IoT Edge to be portable and consistent in the way it works across different devices and environments. Prebuilt Azure services can be deployed, or you can write your own applications and distribute them to edge locations.

www.dbooks.org

The major benefit of IoT Edge is that you offload some of the data processing and network data transfers. If you can process data locally in IoT Edge, you can batch large chunks of data and transmit them back to Azure. Then central applications can aggregate information from other edge locations to be processed by services such as AI and ML.

Another great scenario for Azure IoT Edge is remote locations, often found in the oil and gas or transportation industries, where internet connectivity may not be reliable enough for all the IoT device data to be streamed back to Azure for central processing. IoT Edge allows those remote locations to continue to operate with some amount of autonomy, even when there's no internet connection.

As you plan an application infrastructure that involves IoT devices, examine how you handle network outages and poor internet connections. If your environment relies on the internet, plan for redundant internet connections and equipment to route the data. Or look at IoT Edge to process data locally when it can't be done centrally in Azure.

Try it now

To get started with IoT and create an IoT hub, complete the following steps:

1. Open the Azure portal; launch Cloud Shell; and create a resource group, such as `azuremolchapter20`:

```
az group create --name azuremolchapter20 --location eastus
```

2. You've done a lot of work with the Azure CLI in this book because Cloud Shell and CLI commands allow you to create and manage resources quickly. As mentioned in earlier chapters, the Azure CLI can use additional modules, called *extensions*. These extensions add more functionality and often update outside the regular release cycle of the main Azure CLI. Azure IoT is rapidly expanding and adding new features, so the main commands to interact with IoT Hub come from an Azure CLI extension.

   To get the full functionality you need for these exercises, install the Azure CLI IoT extension:

```
az extension add --name azure-cli-iot-ext
```

3. Create an IoT hub, and enter a name, such as `azuremol`. For these exercises, you can use a free-tier IoT hub, f1:

```
az iot hub create \
  --resource-group azuremolchapter20 \
  --name azuremol \
  --sku f1 \
  --partition-count 2
```

NOTE You can create only one free-tier hub per subscription, but these hubs are great for testing communication between devices and integrating with

other Azure services. The free-tier hub is currently limited to 8,000 messages per day and supports a maximum of 500 connected devices. This may sound like a lot, but depending on what you're doing, a single device that sends a message to the IoT hub approximately every 12 seconds would max out that 8,000-message limit!

Your IoT hub is pretty empty right now. There's not much you can do with it without one or more connected IoT devices. A common device used for IoT is the Raspberry Pi, a low-cost minicomputer that can connect to Wi-Fi networks and use common off-the-shelf sensors for temperature, humidity, and pressure. You can also use it to control small motors, lights, and timers. You don't need to rush out and buy a Raspberry Pi to work with an IoT hub, though; you can simulate one in your web browser!

20.3 Creating a simulated Raspberry Pi device

IoT devices are great, but there's a barrier to entry in that you need an actual device to use, right? Nope! There are a few ways that you can simulate an IoT device with software. This software-based approach lets you focus on building your application quickly and then transitioning to real hardware. You still need to pay attention to how your code runs on real IoT hardware, especially low-power devices, because they may not have access to all the required libraries, or even memory resources, that your simulated application does.

Microsoft provides a free Raspberry Pi simulator through GitHub at https://azure-samples.github.io/raspberry-pi-web-simulator. A Raspberry Pi is great for testing, but take care when using cheap off-the-shelf hardware like the Raspberry Pi in production environments. Plan how you would update and manage such devices. Dedicated IoT devices, such as Azure Sphere (https://azure.microsoft.com/services/azure-sphere), provide additional security and management options. For this book and in your own testing and learning, the Raspberry Pi is a good alternative. In this simulator, a common BME280 sensor that collects temperature and humidity readings is simulated in software, along with a simulated LED to show when the device transmits data to the IoT hub. You can't customize this much, but you can see how a basic Node.js application can run on the Raspberry Pi, poll data from a sensor, and send that back to Azure.

> **NOTE** If things like the Raspberry Pi, electronics and temperature sensors, and Node.js seem daunting, don't worry. As in the chapters on AI and ML, containers, and Kubernetes, we're not going to get super-deep into IoT devices and programming. If you feel like you want to plug in a soldering iron and geek out with electronics by the end of this chapter, though, you're more than welcome to!

Before you can use the Raspberry Pi simulator, you need to create a device assignment in Azure IoT Hub. This process creates a unique device ID so that your IoT hub understands which device it's communicating with and how to process the data. In more complex scenarios, you could provision additional settings for the device and push digital certificates. For this exercise, you'll just create a device identity.

www.dbooks.org

Try it now

To create a simulated Raspberry Pi IoT device, complete the following steps:

1. In the Azure Cloud Shell, create a device identity in your IoT hub, such as `azuremol`, and provide a name for the device, such as `raspberrypi`:

```
az iot hub device-identity create \
--hub-name azuremol \
--device-id raspberrypi
```

2. Remember the shared access policies from section 20.2? Each IoT device also has its own access key and connection string, which are used to identify it when it communicates back to your IoT hub. This key feature of Azure IoT secures devices and minimizes the risk of exposure if one device is compromised.

   To use your device with the Raspberry Pi simulator, you need the information for the device connection string. This unique identifier includes the hostname of your IoT hub, the ID of the device, and an access key:

```
az iot hub device-identity show-connection-string \
  --hub-name azuremol \
  --device-id raspberrypi \
  --output tsv
```

3. Copy the contents of your connection string; you'll need it in step 4. The output is similar to the following:

```
HostName=azuremol.azure-devices.net;DeviceId=raspberrypi;
➥SharedAccessKey=oXVvK40qYYI3M4u6ZLxoyR/PUKV7A7RF/JR9WcsRYSI=
```

4. Now comes the fun part! Open the Raspberry Pi simulator in your web browser: https://azure-samples.github.io/raspberry-pi-web-simulator. Look in the code section at right in the simulator. Around line 15, there should be a `connectionString` variable, which already prompts you for *[Your IoT hub device connection string]*. Copy and paste your connection string from step 3, as shown in figure 20.3.
5. Select the Run button just below the code window to start the simulator.

   Every two seconds, the console window displays a message that shows the data sent to the IoT hub. The red LED on the circuit diagram also flashes when this happens to simulate how outputs connected to the Raspberry Pi can be controlled. The output message in the console window is similar to the following:

```
Sending message: {"messageId":1,"deviceId":"Raspberry Pi Web
➥Client","temperature":24.207095037347923,
➥"humidity":69.12946775681091}
```

Where did the temperature and humidity readings come from? This device is a simulated Raspberry Pi, and there's no real BME280 sensor, so the application generates these values in software. If you look at the rest of the code in the

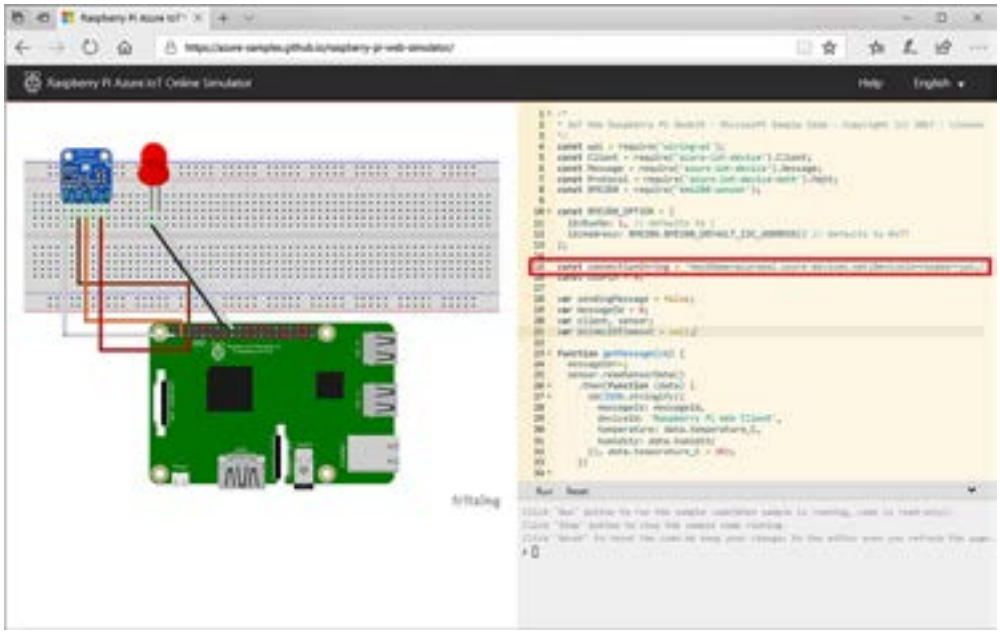


Figure 20.3 Copy and paste the connection string for your Azure IoT device to the Raspberry Pi simulator. The `connectionString` variable is used to connect to transmit the simulated sensor data to Azure.

simulator window, around line 99, the application defines the sensor. Then the simulator replicates how the real sensor would act and generates data returned from the sensor to the application. This example is basic, so think what else you could read in here: revolutions per minute (RPM) of a motor or engine, GPS coordinates of a shipping container or truck, and so on. There's a balance between simulating a device in software and building a functional application with real hardware and sensor data. At some point, you need to purchase or borrow equipment if you want to get into more depth with Azure IoT.

6 To confirm that your simulated device messages are being received by your IoT hub, examine the quota status. Provide the name of your IoT hub, such as azuremol:

```
az iot hub show-quota-metrics --name azuremol
```

The output is similar to the following example, which shows that 5 messages out of the maximum 8,000 total messages per day have been received and that there's 1 connected device from a maximum of 500 total devices. It may take a few minutes for these metrics to populate, so don't worry if you don't see any data right away:

```
[
  {
    "currentValue": 5,
```

www.dbooks.org

```
    "maxValue": 8000,
    "name": "TotalMessages"
  },
  {
    "currentValue": 1,
    "maxValue": 500,
    "name": "TotalDeviceCount"
  }
]
```

You can also look in the Azure portal: choose your resource group, and then select your IoT hub. On the Overview page, the hub usage reports the number of messages received and connected devices. Again, it may take a minute or two for the messages to appear and be recorded against the quota. Any applications would be able to use the messages received immediately, as we'll see in section 20.4.

Trouble in paradise

If you don't receive any messages in your IoT hub, check the output window of your simulated Raspberry Pi device. One of the first things the application does is connect to Azure IoT Hub. A connection error is shown if your connection string is wrong. Make sure that you correctly copy and paste the entire connection string. The connection string starts with `HostName`, and the last character in every access key is always an equal sign (`=`).

If the output window reports an error, copy the error text into your favorite search engine, and search for a matching result. Make sure you didn't change any of the other lines of code, which would cause a problem! The only thing you need to change in the code window is the line for your connection string.

Because the simulated Raspberry Pi device runs in a web browser, you could have a generic website problem. Try to refresh the page, or access the simulator in a different browser (https://azure-samples.github.io/raspberry-pi-web-simulator).

20.4 Streaming Azure IoT hub data into Azure web apps

A device that connects to an IoT hub isn't useful if you can't do anything with the data. This is where you can start to integrate many of the services and features you've learned about in this book. Want to stream to Azure Storage tables or queues? You can do that. Process data from IoT devices in Azure VMs or containers? Go right ahead! Use Azure Cosmos DB to replicate your data, and then access it with globally redundant Azure web apps and Traffic Manager? Sure!

In the example scenario, the IoT hub is the connection mechanism and entry point for your IoT devices into Azure. The hub itself doesn't do anything with the data directly. A default endpoint exists for events, which is a big bucket for any messages received from the IoT device. Your simulated Raspberry Pi device sends messages to the IoT hub, and these messages hit this events endpoint. The flow of messages from devices through the IoT hub to an endpoint is shown in figure 20.4.

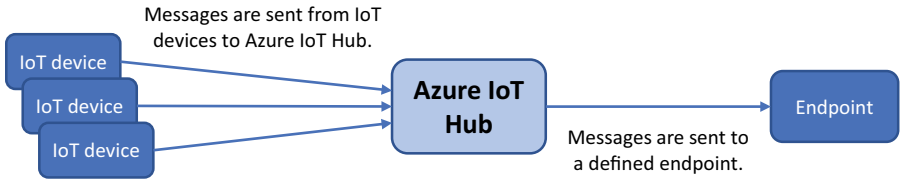


Figure 20.4 An IoT hub receives messages from connected IoT devices and sends the messages to an endpoint. These endpoints can be used by other Azure services to consume data from the IoT devices. A default endpoint for events exists, and services like web apps can read from it.

You can create custom endpoints that route messages directly to Azure services such as Storage and Service Bus. In chapter 4, we looked at Azure Storage queues as a way to pass messages back and forth between applications. A more robust and scalable enterprise messaging platform is Azure Service Bus. Messages can be added to the service bus, such as data received from IoT devices; then other applications can listen for these messages and respond accordingly.

If you don't need the complexity of reading messages from something like a service bus, you can use consumer groups with the default events endpoint. A consumer group allows services such as Azure Web Apps to read data from the endpoint, as shown in figure 20.5. Each service reading from Azure IoT Hub should have its own consumer group. Multiple services, each with its own consumer group, can receive the same messages and process them as needed.

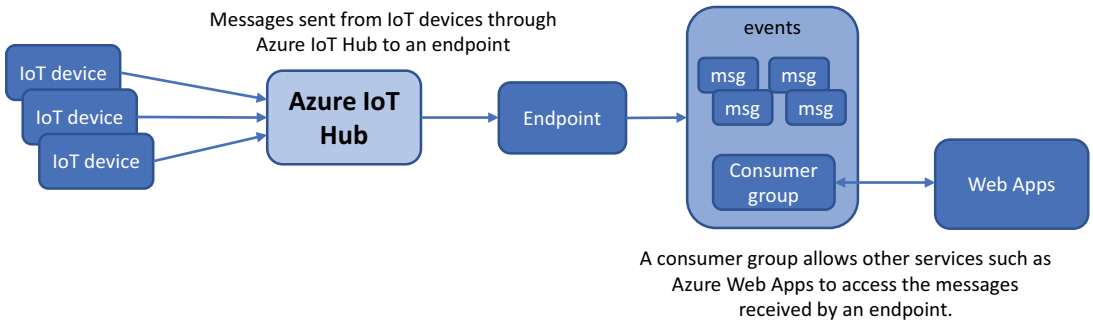


Figure 20.5 Messages are sent from IoT devices to the IoT hub, which directs the messages to an endpoint. In each endpoint, consumer groups can be created. These consumer groups allow other Azure services to access the device messages, which they otherwise wouldn't have access to. With consumer groups, you don't have to use message queues to allow external applications to read IoT device data.

Let's create an Azure web app that uses a consumer group to read message data in real time from your simulated Raspberry Pi device. This basic example shows how you can stream data from IoT devices and access them from web applications.

www.dbooks.org

Try it now

To create an Azure web app that reads data from IoT devices, complete the following steps:

1. Create an Azure App Service plan for your web app in Cloud Shell, and provide a name, such as `azuremol`. For these exercises, the free tier (f1) is good enough and keeps costs down:

```
az appservice plan create \
  --resource-group azuremolchapter20 \
  --name azuremol \
  --sku f1
```

2. Create your web app. Provide a name, such as molwebapp, and enable it for use with Git so that you can deploy the sample application. As with other publicly accessible Azure resources, you need to provide your own globally unique name.

```
az webapp create \
  --resource-group azuremolchapter20 \
  --plan azuremol \
  --name molwebapp \
  --deployment-local-git
```

3. Define the consumer group for your IoT hub, along with some web app application settings. These settings let your web app connect to your IoT hub. Figure 20.6 shows what you build in the next few steps.

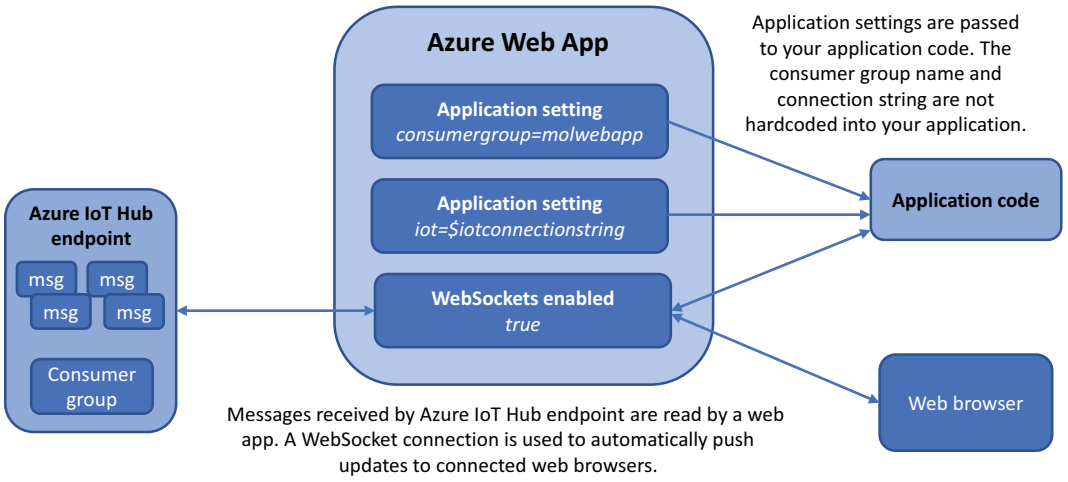


Figure 20.6 To let your web app read the data from your simulated Raspberry Pi IoT device, you create a consumer group in the IoT hub. Then define two application settings for your web app that let you connect to the consumer group. To let your web browser automatically receive the stream of data from the Raspberry Pi as new data is received, you also enable a setting for WebSockets.

4 Create a consumer group that allows your web app to access the event data streamed from your IoT device. Provide your IoT hub, such as `azuremol`, and then enter a name for your consumer group, such as `molwebapp`. Make sure to use your own name throughout the next few steps. Your consumer group is created in the default events endpoint:

```
az iot hub consumer-group create \
  --hub-name azuremol \
  --name molwebapp
```

5 You need to tell your web app what the consumer group is called. Create a web app application setting that's used by the sample application you'll deploy at the end of the exercise. Application settings in web apps allow you to define specific settings, such as the consumer group name and connection string, without those values being hardcoded into your application.

Provide the name of the consumer group you created in step 4, such as `molwebapp`:

```
az webapp config appsettings set \
  --resource-group azuremolchapter20 \
  --name molwebapp \
  --settings consumergroup=molwebapp
```

6 To connect to your IoT hub, your web app needs to know the connection string for the hub. This connection string is different from the one you copied for your simulated Raspberry Pi device in the previous exercise. Remember that there's a connection string for your IoT hub, which uses shared access policies to define access permissions; also, there's a connection string for each IoT device. Your web app needs to read from the IoT hub endpoint consumer group, so you must define a connection string for the IoT hub itself.

7 Get the IoT hub connection string, and assign it to a variable named `iotconnectionstring`, which is used in the step 8:

```
iotconnectionstring=$(az iot hub show-connection-string \
  --hub-name azuremol \
  --output tsv)
```

8 Create another web app application setting, this time for the IoT hub connection string. The variable defined in step 7 is used to let the sample application connect to and read data from the IoT device:

```
az webapp config appsettings set \
  --resource-group azuremolchapter20 \
  --name molwebapp \
  --settings iot=$iotconnectionstring
```

9 Enable WebSockets. A *WebSocket* is a two-way means of communication between a browser and server. The sample application automatically updates the web browser with the data received from the Raspberry Pi device. To perform this

www.dbooks.org

automated update, the application uses WebSockets. Then the server can push data to the browser and cause it to update automatically:

```
az webapp config set \
--resource-group azuremolchapter20 \
--name molwebapp \
--web-sockets-enabled
```

Let's pause here to discuss what you've done so far. You've worked with web apps in many of the previous chapters, but the web app application settings and WebSockets are new. Figure 20.7 recaps how your web app and IoT hub are connected.

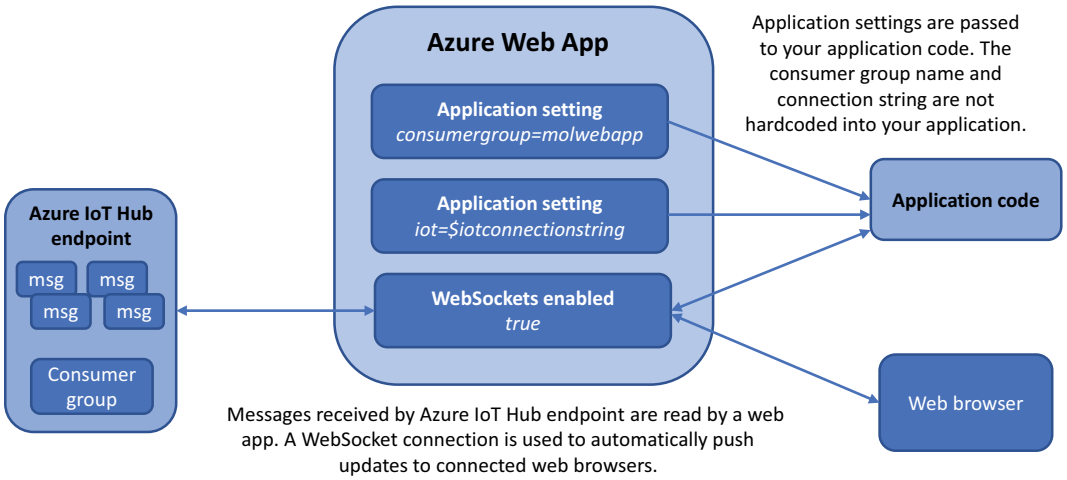


Figure 20.7 As messages are sent from IoT devices, they pass through the IoT hub to an endpoint. Your application code reads in web app application settings that define the IoT hub connection string and consumer group to use. Once the application is connected to the IoT hub, the consumer group allows web apps to read the IoT device messages. Each time a new message is received from an IoT device, your web app uses a WebSocket connection with web browsers that access your site to push updates automatically. This connection allows you to view real-time data streamed from IoT devices, such as temperature and humidity information, from your simulated Raspberry Pi device.

Now let's finish the exercise and deploy the sample application from the GitHub repo to your web app. You can then open the web app in your browser and see the real-time data streamed from your simulated Raspberry Pi!

10 If necessary, clone the GitHub samples repo in your Cloud Shell as follows:

```
git clone https://github.com/fouldsy/azure-mol-samples-2nd-ed.git
```

11 Change to the directory for chapter 20:

```
cd azure-mol-samples-2nd-ed/20
```

12 Initialize the Git repo, and add the basic web page:

```
git init && git add . && git commit -m "Pizza"
```

13 To upload the sample application, create a connection to your web app. The following command gets the web app repository and configures your local samples Git repo to connect to it:

```
git remote add molwebapp \
  $(az webapp deployment source config-local-git \
  --resource-group azuremolchapter20 \
  --name molwebapp \
  --output tsv)
```

In previous chapters, I made you dig around for this address; but by now I hope you've started to explore what else the Azure CLI can do and realized that much of this information can be obtained quickly.

14 Push the HTML sample site to your web app with the following command:

```
git push molwebapp master
```

15 When prompted, enter the password for the Git user you created and have used in previous chapters (the account created in chapter 3).

If you didn't write your Git password on a sticky note

If you've forgotten the password, you can reset it. First, get the username of your local Git deployment account:

```
az webapp deployment user show --query publishingUserName
```

To reset the password, enter the name of your account from the previous command, and then answer the prompts to set a new password. The following example resets the password for the user account named azuremol:

```
az webapp deployment user set --user-name azuremol
```

16 View the hostname for your web app, and then open the address in a web browser:

```
az webapp show \
  --resource-group azuremolchapter20 \
  --name molwebapp \
  --query defaultHostName \
  --output tsv
```

The first time you open the site in your web browser, it may take a few seconds for the web app to connect to your IoT hub, start the WebSocket connection, and wait for the first device message to be received. Every two seconds, the web browser should update automatically with the latest simulated data from the Raspberry Pi device, as shown in figure 20.8.

www.dbooks.org



Figure 20.8 The sample application uses a WebSocket connection between your web browser and web app to update automatically every two seconds with the latest data from your simulated Raspberry Pi device.

If your web app instance doesn't show any data, make sure that the Raspberry Pi simulated device is still running. If necessary, start the simulated device and make sure that it connects to Azure IoT and sends messages. The data should begin to appear in your web app instance.

20.5 Azure IoT component review

I hope the exercises in this chapter have given you an idea of what services are available in Azure for IoT solutions:

- *Azure IoT Hub* provides a great way to provision, connect, and manage many IoT devices and then integrate with other Azure services.
- *Azure IoT solution accelerators* provide prebuilt scenarios that automatically integrate many Azure services to provide a complete application environment.
- *Azure IoT Edge* lets you deploy Azure services in your local environment to process data from IoT devices without the need to stream all the data centrally back to Azure.

To really dig into Azure IoT and IoT devices in general, I recommend that you purchase a basic Raspberry Pi or similar device. These devices are relatively cheap, often come with a few basic sensors or electrical components to test different ideas, and give you a great learning platform as you see what's possible when integrating hardware and software. Just remember the warnings in chapter 17 about AI and ML and building Skynet! Manning also has some excellent books, such as *Building the Web of Things*, by Dominique D. Guinard and Vlad M. Trifa (https://www.manning.com/books/building-the-web-of-things) and *JavaScript on Things*, by Lyza Danger Gardner (https://www.manning.com/books/javascript-on-things), that go into more depth on the Raspberry Pi, IoT best practices, and JavaScript and Node.js programming on IoT devices.

> **Remember that I said "Always delete your resource groups"?**
>
> The best practice throughout this book has been to delete your resource groups at the end of each chapter. This approach ensures that you don't leave services and applications in use that cost money when you don't need them.

(continued)

Azure IoT gives you a great platform to stream data into Azure. You typically need to process that data, not just display it in a web app as you did in the exercises. Chapter 21 examines serverless computing with the Logic Apps and Functions services.

To show how these Azure services work well together, don't delete the resource group and services you deployed in this chapter. You'll use them right away at the start of chapter 21 to see how you can take actions based on the data received from your IoT devices. Just make sure you go back to your simulated Raspberry Pi device and select the Stop button; otherwise, that 8,000-message limit will be used up pretty quickly!

20.6 Lab: Exploring use cases for IoT

This chapter discussed a lot of new stuff, and without a real IoT device, you're limited in what you can do. Chapter 21 builds on Azure IoT Hub and the simulated Raspberry Pi, so I don't want to configure too much more right now. Here are a few things you can do to think further about IoT:

1. What areas can you think of in which IoT devices could benefit your business? If you don't work in a business right now, think about the fictional *Azure Month of Lunches* pizza store.
2. What could you do to improve things for customers with IoT?
3. Would you use Azure IoT Edge? Why or why not?
4. What other Azure services would you likely integrate to run your applications?
5. If you have time left in your lunch break, try one of the Azure IoT solution accelerators at www.azureiotsolutions.com/Accelerators. There's a Device Simulation scenario that creates a VM and simulated sensors, which is like the simulated Raspberry Pi device but much bigger! It takes a few minutes to provision all the required resources, but then look around in the Azure portal to see what was created and how all the parts work together.
6. Can you see how services from earlier chapters, such as Storage and Cosmos DB, are used?
7. What other IoT solution accelerators are available? Do any of them align with ideas you've had for your own applications?

www.dbooks.org

21 Serverless computing

In this final chapter, let's gaze into the future with serverless computing. If you're a developer, the idea of containers (examined in chapter 19) may be appealing because there's less need to configure the underlying infrastructure for your applications. If so, you're going to love the Azure serverless components! And if you're an IT administrator who suddenly wonders what your job will include if there are no servers in the future, don't worry! *Serverless computing* may be more of a marketing term, as many of the server and infrastructure skills you have will continue to apply!

In Azure, two main offerings provide serverless compute features: Azure Logic Apps and Azure Function Apps. In this chapter, we'll explore what each service offers and how they can work together. To make sure your serverless applications can communicate and pass data around, we'll also discuss messaging services such as Azure Event Grid, Service Bus, and Event Hubs.

21.1 What is serverless computing?

To say that serverless computing is without a server is just plain wrong: a server somewhere runs some code for you. The difference from IaaS application workloads like Azure VMs and PaaS workloads in web apps is that serverless applications are usually broken down into smaller discrete units of an application. You don't run a single large application; instead, you run bite-size application components. If this sounds like the containers and microservices that we discussed in chapter 19, don't worry that you're going crazy: serverless computing has a lot of overlap with those topics in terms of how you design your applications. You could arguably create microservices by using the serverless approaches that we'll look at in this chapter.

Figure 21.1 shows how an application is broken into small components that run on a serverless computing provider and provide small units of output.

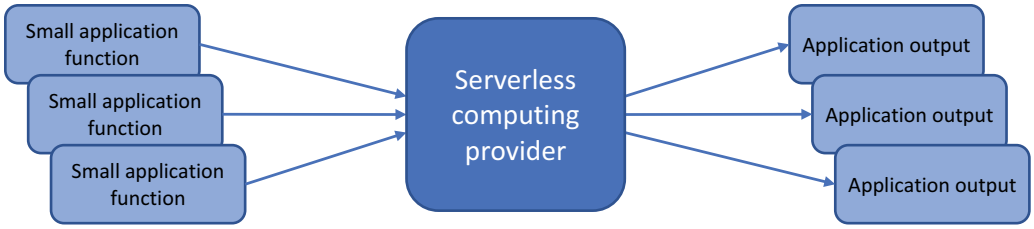


Figure 21.1 In a serverless computing environment, each application is broken into small, discrete units of application components. Each component runs on a serverless computing provider, such as Azure Function Apps, and output is produced that can be consumed by other serverless application components or other Azure services such as Azure IoT or Azure Storage.

In Azure, serverless computing covers two primary services:

- *Azure Logic Apps*—To respond to certain inputs and triggers, logic apps let you visually build workflows that can process and generate additional actions in a point-and-click, no-code-required way. Logic apps can be built by users with no programming or IT infrastructure background. A simple logic app outline is shown in figure 21.2.

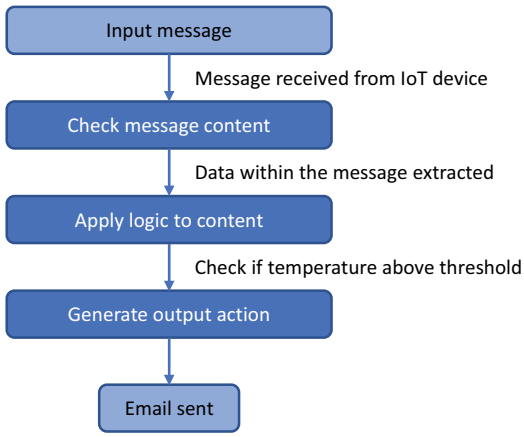


Figure 21.2 In a logic app, an input could be when a tweet is posted, a file is uploaded, or a message is received from an IoT device. The logic app applies rules and filters to the data and determines if the message meets criteria you define. Then output actions, such as generating an email, are completed. All this logic involves no programming or application infrastructure other than an Azure subscription.

There are no security updates to maintain and no design requirements for high availability or the ability to scale. The Azure platform automatically handles these chores. Hundreds of prebuilt connectors exist for logic apps to integrate with services such as Twitter, Office 365, SharePoint, and Outlook. You can respond to public tweets about your company or product, email an alert when a file is uploaded to SharePoint, or send a notification when a message is received from an IoT device.

- *Azure Function Apps*—To run small blocks of code, function apps let you use common programming languages such as C#, Node.js, and Python without any

www.dbooks.org

additional infrastructure management. Your code runs in a secure, isolated environment, and you're billed based on memory consumption per second. Figure 21.3 outlines the basic process for a function app.

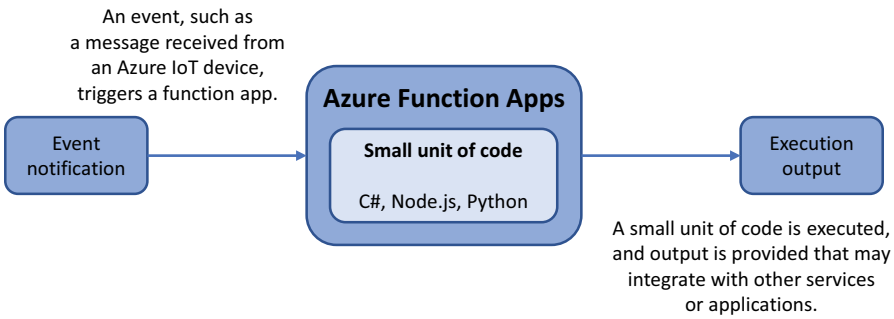


Figure 21.3 As with a logic app, an event notification or trigger usually starts an Azure function. The function app contains a small unit of code that executes a specific task. There's no infrastructure to configure or maintain. Only your small code block is required. When code execution is complete, the output can be integrated with another Azure service or application.

There's no VM to maintain, and no web app is required. You don't have to worry about high availability or scale because the Azure Function Apps service handles these tasks for you. All you provide is your code, and the Azure platform makes sure that whenever you need to run that code, resources are available to process your request.

Logic apps require no code, so they have a wider potential user base. Business application owners or finance and accounting teams, for example, can build their own logic apps without having to write code. Function apps provide more control and flexibility, and let you handle events in a specific way and better integrate with other application components.

Both logic apps and function apps provide a way for you to carry out actions based on triggers without having to maintain any application environment or infrastructure. A server somewhere in Azure runs your logic app or function, but from your perspective as the IT administrator or developer, these technologies are serverless.

21.2 Azure messaging platforms

In chapter 12, we looked at how to monitor and troubleshoot Azure resources, and in chapter 16 we saw how to use Azure Security Center to detect issues and perform update management. Both features rely on streams of data, such as the Azure VM diagnostics extension, to inform the platform about what's happening in the VM. The Azure diagnostics and monitoring platform are great, and other services—such as Web Apps, Azure Container Instances, and Azure IoT Hub—can also stream service diagnostics for central analysis.

With serverless applications, you often need a way to exchange messages and transmit actual application data, not just troubleshoot diagnostics or status updates. That's when you need a messaging platform.

21.2.1 Azure Event Grid

What if you just want to report on certain actions or activities being completed? In automation workflows and serverless computing, the ability to carry out an action in response to an event is useful, as shown in figure 21.4.

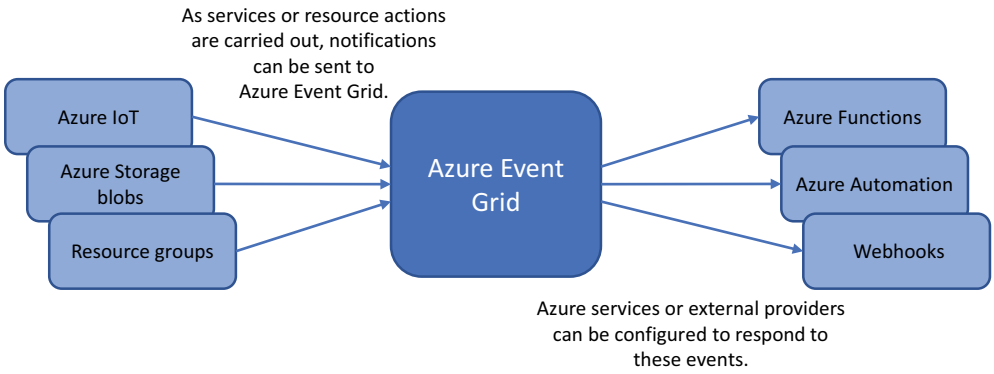


Figure 21.4 Azure services such as Azure IoT and Azure Storage can send notifications to Azure Event Grid. These notifications may happen when a message is received from an IoT device or a file is uploaded to storage. Azure Event Grid allows other services and providers to subscribe to these notifications to perform additional actions in response to events.

Let's examine a couple of scenarios that you may be able to use in your pizza store:

- *Message received in an IoT hub*—An IoT device connected to an IoT hub may report a temperature reading in an oven or a delivery vehicle's location. The IoT hub is configured to forward a notification to Azure Event Grid.

  An Azure function is subscribed to the Event Grid notifications for an IoT hub and runs a small serverless application component to log the information to Azure Cosmos DB and send an email notification. You could also use Logic Apps instead of Azure Function Apps, depending on how complex the application response needs to be.
- *File uploaded to Azure storage*—The marketing department may upload to storage a promotional coupon to save money on a pizza order. When a new file is created, a notification is sent to Event Grid.

  A webhook is subscribed to Event Grid and posts a copy of the image from storage to Twitter. This tweet lets customers know about the deal of the week or money-saving coupon.

www.dbooks.org

These scenarios are for truly hands-off serverless computing scenarios, but Event Grid can also integrate with more traditional resources, such as VMs and web apps. A resource group can be configured to send notifications to Event Grid, for example. There are many ways to create a VM, such as in the portal, with the Azure CLI, or with a Resource Manager template, so you want to make sure the VM is configured correctly for Update Management through Security Center. An Azure Automation runbook could be subscribed to Event Grid for notifications about VM create operations; then it would on-board the VM to the Update Management service and install required security or application updates.

21.2.2 Azure Event Hubs and Service Bus

Event Grid can work with many Azure resources, and it's well suited to serverless computing with logic apps or function apps. But logic apps and function apps can run based on other data inputs, such as event hubs or a service bus. Let's look at the differences among these various messaging services so that you can decide when to use them:

- *Azure Event Hubs* lets you receive a stream of data, such as from IoT devices or application telemetry. Event hubs provide a low-latency messaging platform capable of handling millions of events per second from multiple concurrent providers. Event hubs are a data store rather than a queue of messages, and the client or application checks for events in the hub at whatever frequency you wish. Then the data received in the event hub can be processed by other services, as shown in figure 21.5.
- *Azure Service Bus* allows application components to exchange message data, such as the storage queues that we examined in chapter 4. Storage queues are an earlier, more basic implementation of a messaging platform in Azure. A *service bus*

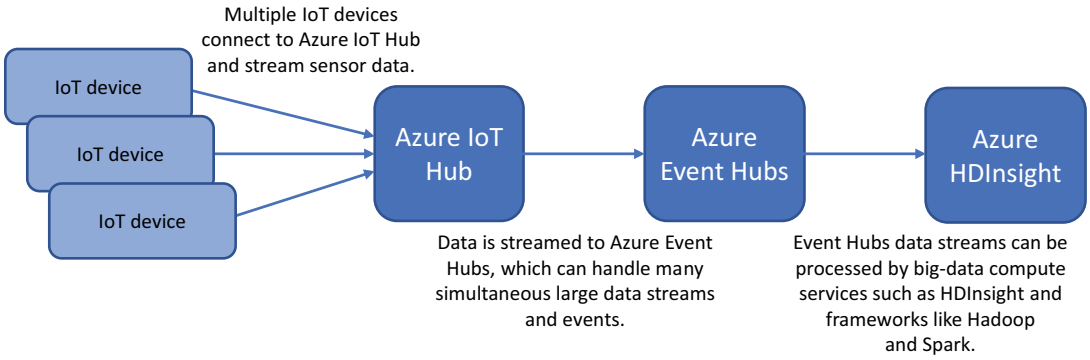


Figure 21.5 IoT devices connect to the IoT Hub and can stream all their sensor data. There could be hundreds or thousands of connected IoT devices. Azure Event Hubs handles all these separate data streams and allows services such as Azure HDInsight to process the raw data in Hadoop or Spark clusters to analyze and generate reports.

provides more advanced features, such as guaranteed ordering of messages, atomic operations, and sending messages in batches. Figure 21.6 outlines a common scenario for a service bus.

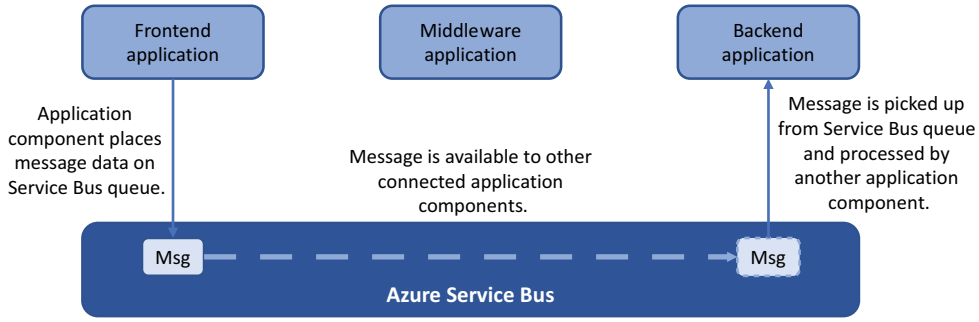


Figure 21.6 Messages are placed in a service bus queue by application components—a frontend app, in this example. Other middleware or backend applications can pick up these messages and process them as needed. Here, a backend application picks up the message and processes it. Advanced messaging features include guaranteeing the order of messages on the queue, locking messages, timeouts, and relays.

With three services that let you transmit, receive, and process data among applications and services in Azure, which one do you use, and when? Table 21.1 provides a high-level recap of the Event Grid, Event Hubs, and Service Bus services.

Table 21.1 Each service is designed to cover a different scenario. Event Grid lets you react to events, Event Hubs lets you stream large amounts of data, and Service Bus lets you transmit messages between services and application components.

Azure service	Provides	Use case
Event Grid	Distribution of events	Perform an additional action based on an event occurrence.
Event Hubs	Data streams	Receive and transmit large volumes of concurrent data.
Service Bus	Message transmission	Provide communication between services and apps.

Azure Logic Apps and Function Apps can be trigged by all three messaging platforms. Let's create a service bus that can be used to trigger a logic app.

21.2.3 Creating a service bus and integrating it with an IoT hub

In this scenario, let's use a service bus to transmit messages received from an IoT hub. Your simulated Raspberry Pi device from chapter 20 generates temperature readings and transmits them to your IoT hub. If the temperature is higher than 30°C, another

www.dbooks.org

piece of data is included in the message from the IoT device: `temperature-Alert = true`. Figure 21.7 outlines how you can integrate an IoT hub with the service bus to process messages with this temperature alert.

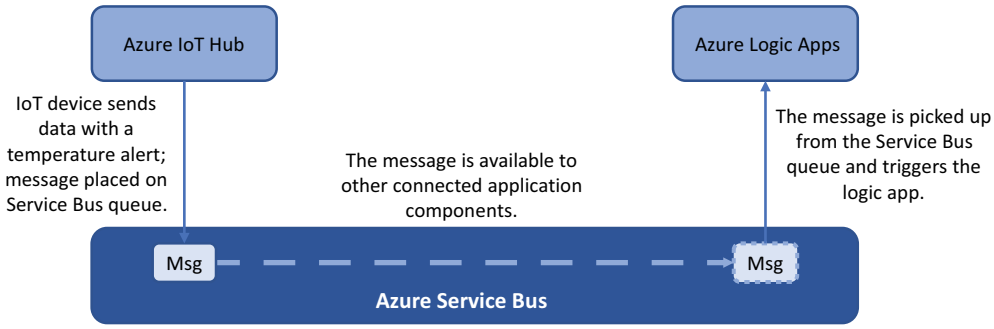


Figure 21.7 When your simulated Raspberry Pi IoT device sends message data, a temperature reading of 30°C or more generates an alert. Messages tagged with this alert are placed on a service bus. Then these messages can be used to trigger logic apps.

Try it now

To create a service bus, complete the following steps:

1. Open the Azure portal, and choose Create a Resource at top left in the menu.
2. Search for and select Service Bus, and then choose Create.
3. Provide a name, such as `azuremol`, and then select the basic pricing tier.
4. Create a new resource group, and provide a name, such as `azuremol-chapter21`. Make sure that the location is the same as for the resources created in chapter 20, such as East US. The interaction among a service bus queue, a logic app, and a function app may have issues if you aren't consistent with your locations.
5. Accept the other defaults, and choose to create the service bus.
6. When the resource has been created, select your resource group, and choose the service bus you created in step 5.
7. Select Queues; add a new queue; and enter a name, such as `azuremol`.
8. Accept all the other defaults, and choose Create.

With a service bus and a queue created, how do you configure an IoT hub to use them? In the IoT hub, you define *endpoints* as the destinations for messages received from IoT devices. A default endpoint exists in the IoT hub for all messages that don't meet the defined criteria. You can configure the service bus as an endpoint to receive messages. Then a *route* is defined that includes criteria for which messages should be

directed to an endpoint. In this example, that route requires any message that contains `temperatureAlert = true` in the message body to be routed to the service bus endpoint, as shown in figure 21.8.

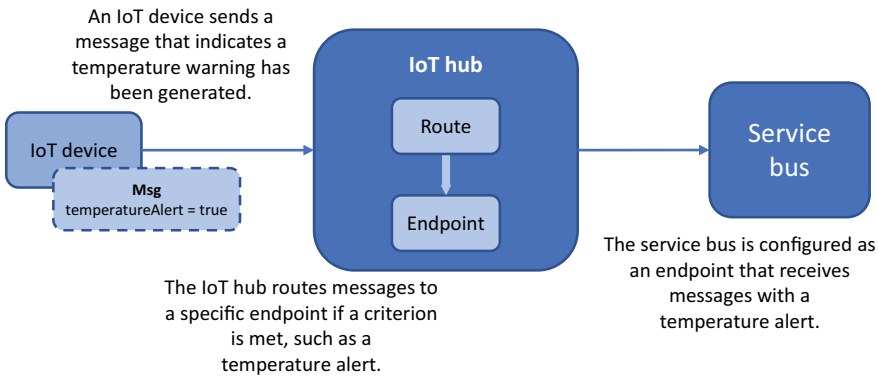


Figure 21.8 As messages are transmitted from IoT devices to an IoT hub, they can be routed to specific endpoints based on criteria you define. Messages that contain a temperature alert in the message body can be routed to an endpoint that uses the service bus queue. Then messages placed on the service bus queue that contain a temperature alert can be used to trigger things like Azure logic apps or function apps.

Try it now

To configure an IoT hub to route temperature alert messages to the service bus, complete the following steps:

1. Select your resource group from chapter 20, such as `azuremolchapter20`, and then choose the IoT hub.
2. Under Messaging in the navigation bar on the left, select Message Routing, and choose to add a custom endpoint for a service bus queue.
3. Provide an endpoint name, such as `azuremol`.
4. Select your service bus queue namespace, such as `azuremol`, and then select your actual queue.
5. To direct messages to this endpoint, create a route. In the Message Routing section in the navigation bar on the left, select Routes, and choose to add a new route.
6. Provide a name, such as `temperatureAlert`.
7. Choose the service bus endpoint you created in earlier step, such as `azuremol`.
8. For the routing query, enter the following:

```
temperatureAlert = "true"
```

9. When you're ready, save the route.

www.dbooks.org

Now you have a simulated Raspberry Pi device that sends data to the IoT hub, as well as a route to place messages that contain a temperature alert on a service bus message queue. You don't really have an application yet, though; there's nothing you can do with the data on the service bus queue. What might you want to do with a temperature alert? Sending an email notification is a common example, so let's see how you can trigger a logic app each time a message is placed on the service bus queue.

21.3 Creating an Azure logic app

As you saw when we discussed logic apps in section 21.1, a message received from a service bus queue can be used as a trigger to start the execution process. You use the IoT hub to process the messages received from IoT devices and route only to the service bus queue endpoint messages that contain `temperatureAlert = true` in the message body. With this approach, your logic app runs only when a temperature alert is generated.

Figure 21.9 outlines what your logic app does. When a message is placed on the service bus queue, the logic app runs and sends an email alert.

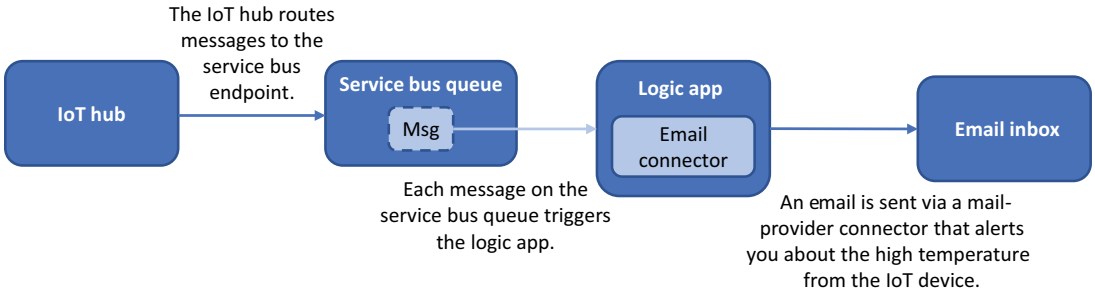


Figure 21.9 Each message received on the service bus queue from the IoT hub triggers the logic app. When the logic app runs, it sends an email notification through a defined mail provider.

Try it now

To create a logic app, complete the following steps:

1. In the Azure portal, choose Create a Resource at top left in the menu.
2. Search for and select Logic App, and then choose Create.
3. Provide a name, such as `azuremol`, and select your resource group, such as `azuremolchapter21`. Again, choose the same location as your other IoT resources from chapter 20.
4. Accept the other defaults, and choose Create.
5. When the resource has been created, select your resource group, and then open the logic app. For the option "Add common triggers," choose "When a message is received in a Service Bus queue."

6 Provide a name, such as `azuremol`; then select your service bus queue, such as azuremol.

7 Choose the default service bus policy listed, such as RootManageSharedAccessKey, and create the connection.

8 Select Continue; then choose your service bus queue name, such as `azuremol`.

9 Accept the defaults, such as the frequency to check for messages.

10 Choose to add a new step to the logic app.

11 To add an action, search for what you want to do. In this exercise, search for *email*. Select your provider, such as Gmail - Send an Email, Outlook.com - Send an Email, or SMTP - Send an Email, as shown in figure 21.10.

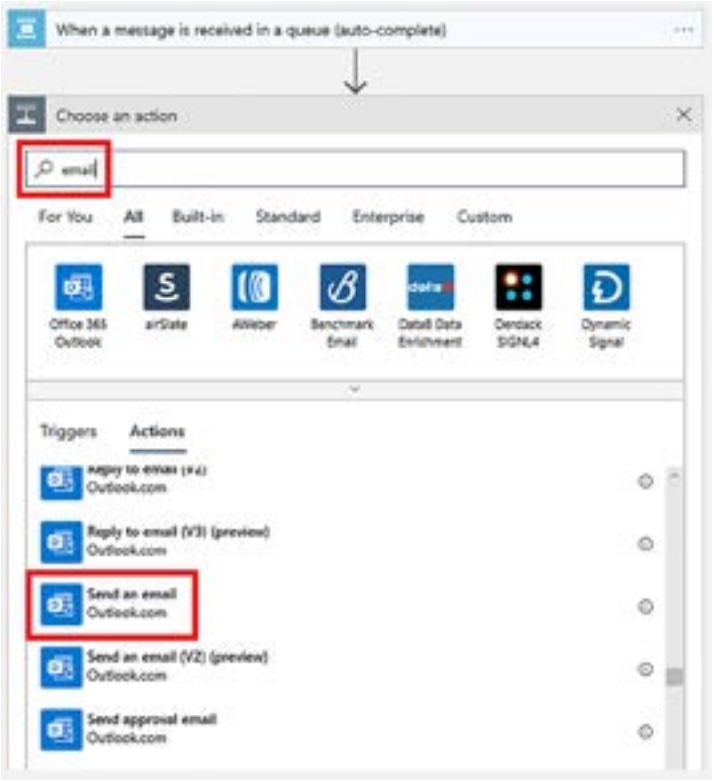


Figure 21.10 Search for and select your current email provider, such as Gmail or Outlook.com. You can also choose SMTP - Send an Email to configure a different provider manually.

12 Sign in to your email provider to authorize mail routing, and confirm that you wish to grant logic apps permissions to send email.

www.dbooks.org

13 Provide a recipient email address at which you receive email; an email subject, such as `Temperature alert`; and a message body, such as `High temperature detected on IoT device`.

14 Save the logic app.

Let's pause to review what you've built in the last few exercises, as shown in figure 21.11. This basic serverless application design doesn't include any controls that limit the number of messages to be sent. In the logic app, you could define that you want to send a maximum of five email alerts and then wait for 30 minutes before sending more. As part of your application design, you should consider how you want to be notified of situations like this one. You could also configure the logic app to read in the message data from the service bus queue and include the timestamp of the IoT device message and the actual recorded temperature. We'll discuss how to do this in the next exercise.

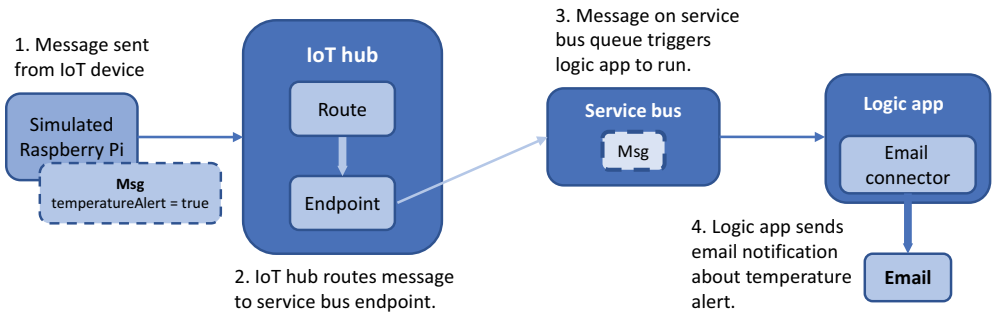


Figure 21.11 The simulated Raspberry Pi device sends a message to the IoT hub every two seconds that contains temperature sensor readings. If the temperature is above 30°C, a temperature alert is noted. The IoT hub routes any messages that contain a temperature alert to a service bus queue. Messages on this queue trigger an Azure logic app to run. The logic app is connected to an email provider, such as Outlook or Gmail, and sends an email notification about the temperature warning from the IoT device.

Let's see this basic serverless application in action.

Try it now

To run your simulated Raspberry Pi device and test your logic app, complete the following steps:

1 Open a web browser to the simulated Raspberry Pi IoT device from chapter 20 (https://azure-samples.github.io/raspberry-pi-web-simulator).

2 Verify that your IoT hub connection string is still added in the code window that you configured in chapter 20.

3 Choose to run the app.

Simulated temperature and humidity sensor readings are generated every two seconds, and a message is sent to the IoT hub. It may take a few messages for a simulated temperature reading of 30°C to be generated and shown in the output window.

The IoT hub routes any messages that contain `temperatureAlert: true` to the service bus endpoint. As these messages are placed on the service bus queue, the logic app picks them up and sends an email through the defined provider. Then you receive an email notifying you of a high temperature reading. This process should take only a few seconds end to end.

4 The simulated Raspberry Pi device generates messages every two seconds, so stop the app unless you like a lot of email alerts!

When you receive email alerts, the message doesn't contain a lot of information. Your logic app doesn't extract the message contents from the service bus and format the information. It would be great if the alert email could include the IoT device name or the temperature recorded. How can you process each message and perform some analysis on it? What about the other Azure serverless service we've looked at: Azure Function Apps?

21.4 Creating an Azure function app to analyze IoT device data

To extend your current serverless application, you can trigger an Azure function app from within your logic app. Message data from the service bus can be sent to a function app for analysis of the recorded temperature. Then the email notification sent by the logic app can include information about the IoT device name and the recorded temperature. The interaction between the logic app and the function app is shown in figure 21.12.

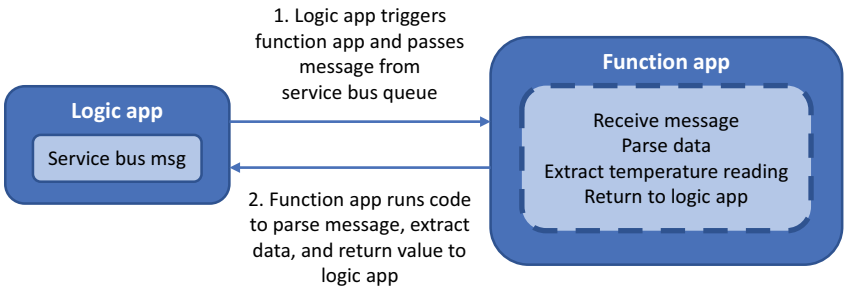


Figure 21.12 The logic app triggers the function app. The message received on the service bus queue is passed into the function. Code in the function app parses the message, extracts the temperature, and returns that value to the logic app. It takes a few milliseconds for the function app to run this code, so the cost of performing these compute tasks is a fraction of a cent.

www.dbooks.org

Try it now

To create a function app and trigger it from the logic app, complete the following steps:

1. In the Azure portal, select Create a Resource at top left in the menu.
2. Search for and select Function App, and then choose Create.
3. Select your resource group, such as `azuremolchapter21`, and provide a name, such as `azuremol`. You want to be in the same region as your previous resources.

   You want to publish code, but note that you can also publish a Docker container image (chapter 19). You wouldn't even need to create a container instance or any additional infrastructure; a short-lived container would run as needed and then stop.
4. For this basic application, choose the Node.js runtime, as we use some simple JavaScript.
5. You have three hosting-plan options. A *consumption plan* lets you pay per execution, and the resources you require are assigned dynamically at runtime. For more consistent, production-ready applications, you can use a *premium* or *dedicated hosting plan* that provide a more fixed, predictable cost. Premium plans provide additional features, such as securing connectivity to a defined set of Azure virtual networks and always having an instance ready to prevent some delays in a cold-start scenario for your app. For this exercise, choose a consumption plan.
6. Accept the other defaults to create a named storage account and Application Insights, and then choose Review + Create.
7. When you're ready, create the function app. It takes a minute or two to create the function app.
8. When the resource has been created, select your resource group, open your logic app from the previous exercise, and select Edit.
9. In the Logic Apps Designer, choose to add a new step.
10. Search for and select Azure Functions, choose the function created in the previous steps (such as `azuremol`), and then choose Create New Function.
11. Provide a function name, such as `analyzeTemperature`.
12. Delete any existing code, replace it with the code from the following listings, and then choose Create.

    This code is also available in the GitHub repo at w.

Listing 21.1 `analyzeTemperature` JavaScript code for a function app

```
module.exports = function (context, req) {
    var buffer = new Buffer(req.body.ContentData, 'base64')
    var decodedString = buffer.toString();
    var objects = JSON.parse(decodedString);
    var temperature = objects["temperature"];
    context.res = {
        body: {
            analysis: "Recorded temperature was  " + temperature + "!"
        }
    };
    context.log("Recorded temperature was " + temperature);
    context.done();
};
```

Creates the function. Every JavaScript function app starts with exporting a function that contains a context object. This context object is used to pass data back and forth.

Reads in message content from the service bus

Creates a JSON object of decoded service bus messages

Decodes from base64

Extracts the recorded temperature from the IoT device

Builds a response to send back to the Logic App

Outputs the temperature to the console log

Ends the function. Every JavaScript function app must end with a call to context.done, which tells the function app that your code is finished.

13 Back in the Logic Apps Designer for your function step, select the Request Body text box, and choose Service Bus Message from the list of dynamic content on the right side.

14 In the Logic Apps Designer, drag and drop to reorder the steps so that the Send an Email action is below the `analyzeTemperature` function app step, as shown in figure 21.13.

15 Select the Send an Email action, and choose the text box for the body of the email message.

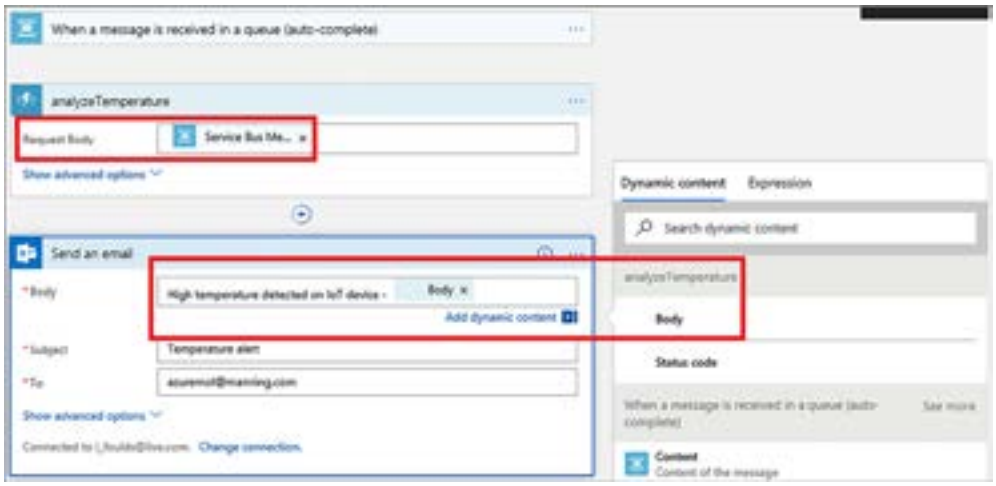


Figure 21.13 Drag the Send an Email action below the `analyzeTemperature` function. Select the end of the message Body, and the Dynamic content dialog appears. To insert the temperature value computed by the function app, select the message Body from the `analyzeTemperature` function.

www.dbooks.org

16 From the `analyzeTemperature` function, select the Body response, as shown in figure 21.13.

17 In the Logic Apps Designer, choose Save.

Your serverless application has a lot of moving parts. Let's examine what you've built before you run the simulated Raspberry Pi IoT device to generate email alerts that include the temperature reading as computed by the function app. Figure 21.14 provides an overview of all the components now in use in the serverless application.

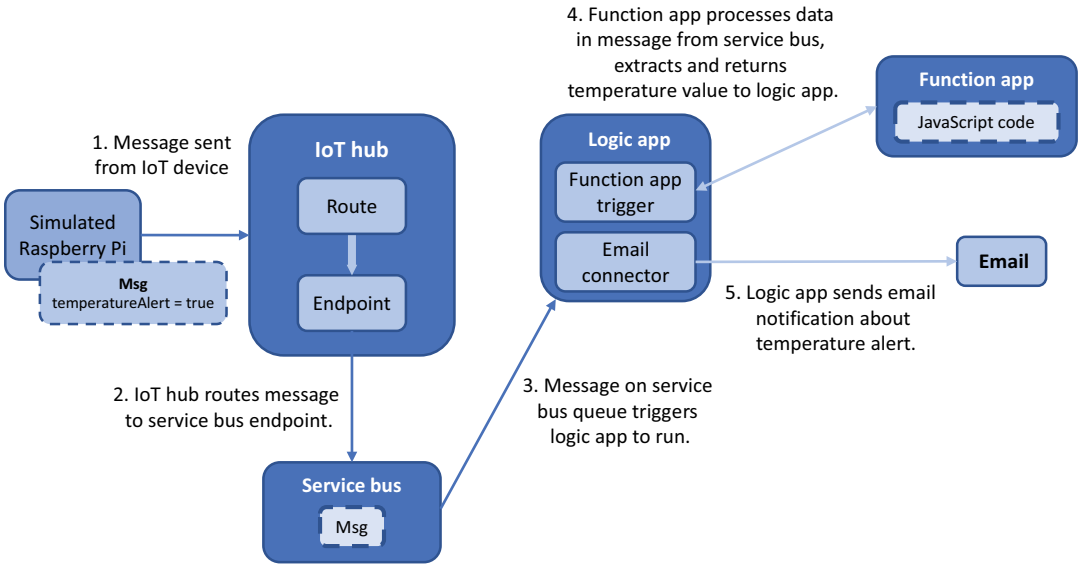


Figure 21.14 As messages are received from the simulated Raspberry Pi device, any messages that contain a temperature alert are routed to the service bus queue endpoint. Messages on the service bus queue trigger a logic app, which passes the message to a function app. A JavaScript function parses the temperature reading and returns it to the logic app, which sends an email notification that includes the temperature recorded by a sensor on the IoT device.

18 Open your simulated Raspberry Pi device in a web browser, and run the application. Each time the temperature alert is generated, the logic app triggers the function app to extract the temperature data from the message body and include that in the email notification. It may take a few moments for a temperature reading to be higher than 30°C, which then flags the message with a temperature alert. When that alert is sent and the messaged is processed, you receive an email notification that reports that temperature.

Take a deep breath, and pat yourself on the back. That was a lot to do on your lunch break!

Authentication errors from your logic app to the function app

You can see the run history in the overview window of your logic app in the Azure portal. If you get a lot of repeated errors, select one of the errors to see more about where it's failing.

A common issue is that the logic app isn't automatically authorized to talk to the function app. Redeploying the logic app usually fixes this error, but the real resolution is probably to add what's called the *function* key to the header of your logic app.

To get this key, select your function app, and then choose the function you created, such as `analyzeTemperature`. Under the Manage option, the default function key can be viewed and copied. Copy this key, go back to your logic app, and open the designer.

In the `analyzeTemperature` function, choose to add a parameter, and then to add a header. You want to send a little bit of information at the start of the call to the function app that sends the key. The process is a little backward as you enter the key pair, but enter `x-functions-key` for the key, and then paste your actual function key as the value.

It takes a few moments to update the logic app/function app integration. After that, the run history for the logic app should show the events working correctly, and the email notifications should start being delivered.

21.5 Don't stop learning

This chapter contained a lot of new concepts. In fact, the past few chapters contained many new ideas and technologies! Don't worry if you're struggling to understand how you can begin to implement all these Azure services, such as containers, AI and ML, and serverless computing. These chapters were designed to show you what's possible in Azure and prove that you aren't limited to performing a lift and shift of legacy applications. As you start to build and run applications in Azure, take the opportunity to modernize applications and review management or deployment workflows. Many Azure services simplify and speed up the application lifecycle, so don't feel like you have to stick with running VMs because that's what the business is comfortable using.

Yes, Azure offers lots of shiny new services, but they all largely build on core infrastructure components discussed in part 1 of this book. Developers can start to use the latest application design approaches that involve Kubernetes or serverless computing, and admins can reuse their on-premises data center knowledge with cloud computing fundamentals and troubleshooting techniques. As your business needs grow, Azure can support them.

In chapter 1, I was open and honest, stating that I wouldn't cover every service in Azure. There are many more Azure services to learn about, and you can get into a lot more depth than we did in the book. I hope you've found at least a few areas that interest you and motivate you to explore some more. My favorites include virtual machine scale sets, Cosmos DB, and Azure Kubernetes Service.

www.dbooks.org

21.5.1 Additional learning materials

I'm biased, but I think that a great place to continue learning about Azure is https://docs.microsoft.com/azure. This web page has the core Azure service documentation, architecture guides, reference and SDK resources, and samples. Each Azure service has its own set of quick-starts, tutorials, and samples, along with conceptual information and individual how-to guides.

If you get serious, you can investigate certification options for Azure. Individual exams include *Microsoft Azure Administrator (AZ-104), Microsoft Azure Architect Technologies and Design (AZ-303 and AZ-304),* and *Microsoft Azure Security Technologies (AZ-500).* This book and the lab exercises you completed covered a lot of the areas those exams test your knowledge on, but you'd need to study some additional areas of Azure AD and design best practices before taking the exams. The Microsoft Learn site at https://docs.microsoft.com/learn provides some additional learning paths for the different Azure certification options to help you prepare.

21.5.2 GitHub resources

Throughout this book, you've used code samples, templates, and sample applications from https://github.com/fouldsy/azure-mol-samples-2nd-edition. These samples should stay updated as new versions of the Azure CLI are released; the GitHub repo also includes PowerShell examples and templates for all the exercises. This book focuses on the Azure CLI in the Azure Cloud Shell, but feel free to explore what each exercise looks like in PowerShell or a template.

If you notice any problems with the samples, please create an issue in GitHub at https://github.com/fouldsy/azure-mol-samples-2nd-edition/issues. Things move fast in Azure, and I want to make sure that you always have the latest working samples to learn from. Feel free to make suggestions too! All the Azure docs at https://docs.microsoft.com/azure also accept feedback, issues, and edits, so as you explore the rest of what's on offer in Azure, feel free to get involved and help others learn and grow.

21.5.3 One final thought

Take a deep breath, and realize that change is normal. New features and services release almost daily. Azure, like all major cloud computing providers, may look and feel ever so slightly different from the last time you used it (an hour ago). If you have the fundamental skills and understanding that I hope you've learned in this book, you can adapt and grow with all the new opportunities Azure offers. You always have something new to learn, and I'd love to hear what you end up building and running in Azure!

www.dbooks.org

index

Symbols

&& character 27
$ResourceGroups object 276
$servicePrincipalConnection object 275

A

-A parameter 121
Access Control (IAM) button 79
access_token variable 226
accounts
  in Azure Automation, creating 271–272
  in Cosmos DB
    adding global redundancy to 149–152
    creating 145–149, 152
    populating 145–149
ACI (Azure Container Instance) 284, 290, 292
ACR (Azure Container Registry) 293
agents 180
AI (artificial intelligence) 253–268
  Azure Cognitive Services 259–260
  LUIS 261–264
  machine learning and 254–259
  overview of 254–255
  Web App bots
    building with LUIS 264–267
    creating 260
    running with LUIS 264–267
AKS (Azure Kubernetes Service) 284, 293–297
  creating clusters with 294–295
  running websites in Kubernetes 295–297
  viewing information on 294
alerts 178–182
alias records 160
AllowAzureLoadBalancerInBound rule 67
AllowVnetInBound rule 67
Amazon Web Services (AWS) 191
anycast networking 160
APIs (application programming interfaces) 31
App Service environments 36
App Service plans 35–38
Application Gateway 107–108
Application security group level 185
Apply and autocorrect mode, DSC 279
Apply and monitor mode, DSC 279
Apply only mode, DSC 279
apps
  Function Apps 328–331
  life cycles of 76–77
  Logic Apps 325–328
  service plans 38
apps. *See also* Azure Web Apps
APT (Advanced Packing Tool) 27
artificial intelligence. See AI
assets, in Azure Automation 272–274
authentication errors 332
Auto Swap 46
Automation Hybrid Worker 272
autoscale rules 133–136
Availability Sets 91
  distributing VMs across 98–101
  viewing distribution of VMs across 101–102
  VM redundancy with 96–102
    fault domains 96–97
    update domains 97–98
Availability Zones 91
  creating network resources across 94–95
  creating VMs in 95
  infrastructure redundancy with 95
AWS (Amazon Web Services) 191

az cosmosdb show command 152
az group create command 131
az keyvault create command 212
az keyvault secret show command 221
az storage account create command 210
az vm command 95
az vm create command 51, 95, 197
az vm disk attach command 51
az vm list-sizes command 127
az vm resize command 127, 129
az vm show command 105
Azure account, creating 5–7
Azure AD (Azure Active Directory) 222
Azure Application Gateway 108
Azure Application Insights 180
Azure Automation 243, 269–283
  assets 272–274
  creating accounts in 271–272
  overview of 179, 269–274
  PowerShell DSC 278–282
    Azure Automation pull servers and 280–282
    defining 280–282
  runbooks 272–274
    running 276–277
    sample of 274–277
    viewing output from 276–277
Azure Backup 191–201, 243
  Backup schedules 196–198
  policies and retention 193–196
    RPO (recovery point objective) 194–195
    RTO (recovery time objective) 195–196
  restoring VMs 198–201
    complete VM restore 199–201
    file-level restore 199
Azure Bastion 24, 115
Azure CLI 7, 12–13, 28, 31, 81–82, 152, 161
Azure Cloud Shell 12–13
Azure Cognitive Services 259–260
Azure Container Instance. *See* ACI
Azure Container Registry. *See* ACR
Azure DevOps utilities 7
Azure DNS (Domain Name Service) 158–162
Azure DNS record types 160
Azure Event Grid 320–321
Azure Event Hubs 321–322
Azure Firewall 238
Azure Front Door 163–164
Azure Function Apps 318
Azure IoT (Internet of Things) 300–316
  creating function apps to analyze device data 328–331
  Hub, centrally managing devices with 303–309
  integrating with Service Bus 322–325
  overview of 300–302
  review of components 315
  streaming hub data into web apps 309–315
Azure IoT Edge 304–305
Azure Key Vault 216–233, 304
  creating certificates 229–232
  injecting certificates 229–232
  MSIs (managed service identities) 221–229
  overview of 211
  securing information in clouds 216–221
    creating key vaults and secrets 219–221
    software vaults and HSMs 217–218
  storing encryption keys in 211–213
Azure Kubernetes Service. *See* AKS
Azure Logic Apps 318
Azure Machine Learning service 258
Azure Monitor 243
Azure Network Watcher 182–188
  capturing network packets 186–188
  verifying IP flows 183–184
  viewing effective NSG rules 184–186
Azure Networking 58–72
  building sample web applications with secure traffic 68–72
    creating remote access network connections 68–69
    creating VMs 69–70
    using SSH agents to connect to VMs 70–72
  securing and controlling traffic with NSGs 64–68
    associating NSGs with subnets 66–67
    creating NSG filtering rules 67–68
    creating NSGs 64–65
  virtual network components 58–64
    creating subnets 59
    creating virtual networks 59
    DNS resolution 62–64
    public IP addresses 62–64
    virtual network interface cards 61
Azure platform
  management tools 11–13
    Azure Cloud Shell 12–13
    Azure portal 12
    Azure PowerShell 13
    local Azure CLI 13
  overview of 8–13
  storage in 18
  troubleshooting 31–32
  virtualization in 10–11
Azure portal 12
Azure PowerShell 11, 13, 31, 81–82, 161
Azure Quickstart templates 7
Azure Resource Manager 75–89
  approach to 75–81
    designing around application life cycle 76–77
    managing and grouping resources with tags 80–81

www.dbooks.org

protecting resources with locks 79–80
securing and controlling resources 78–79
templates for 81–87
creating 82–84
creating multiples of resource types 84–85
storing 87
tools to build 85–86
Azure Security Center 234, 249
Azure Service Bus 310, 321–322
Azure Service Fabric 289
Azure Site Recovery 201–204, 243
Azure Storage 47–57
adding disks to VMs 50–52
benefits of 52–57
queue storage 55–56
redundancy 56–57
storage availability 56–57
table storage 53–54
VM storage 47–50
data disks 49–50
disk-caching options 50
standard vs. premium storage 48–49
temporary disks 49–50
Azure Traffic Manager. *See* Traffic Manager
Azure Update Management 241–249
OMS (Operations Management Suite) 243
reviewing and applying updates 245–249
Azure Web Apps 33–45
building bots with LUIS 264–267
building with secure traffic 68–72
creating remote access network connections 68–69
creating VMs 69–70
using SSH agents to connect to VMs 70–72
creating 37–42
creating basic web apps 37
deploying sample HTML sites 39–42
creating bots 260
deploying application to web app running multiple instances 140
deployment slots and 35, 44–46
diagnostic logs, viewing 42–44
managing 42–44
overview of 34–35
running bots with LUIS 264–267
scaling 127–139
streaming Azure IoT hub data into 309–315
supported languages and environments 34–35
Azure-to-Azure replication 203

B

B-series VM 18
backend IP pool, in load balancers 107
backend pools 107, 116–119
backslash character 40, 68
Backup schedules 196–198
backups 191–204
Azure Backup 191–201
Backup schedules 196–198
policies and retention 193–196
restoring VMs 198–201
Azure Site Recovery 201–204
Bash shell 12
Basic service plan 36
bastion host 23–24
Bing Autosuggest service 259
Bing Custom Search service 259
Blob storage 52
boot diagnostics 175–177
bots for web apps
building with LUIS 264–267
creating 260
running with LUIS 264–267
branches, in Git 41
Building the Web of Things (Guinard and Trifa) 315

C

capturing network packets 186–188
caret symbol 27
CD (continuous delivery) 75
certificates 270
creating 229–232
injecting 229–232
CI (continuous integration) 75
CLI (command-line interface) 12
clouds, securing information in 216–221
creating key vaults and secrets 219–221
software vaults and HSMs 217–218
clusters with AKS 294–295 collections 146
command-line interface (CLI) 12
commands, wrapping long lines 40
compute-optimized VM sizes 17
Computer Vision service 259
concat function, Resource Manager 85
configuring
health probes 110–112
VMs with load balancers 119–122
connections 270
connectionString variable 307
container orchestrator 293
containers 146, 284–299
ACI (Azure Container Instance) 289–292
AKS (Azure Kubernetes Service) 293–297
creating clusters with 294–295
running websites in Kubernetes 295–297
overview of 284–288
Content Moderator service 259

continuous delivery (CD) 75
continuous integration (CI) 75
Contributor role 78
controlling
  resources 78–79
  traffic with NSGs 64–68
    associating NSGs with subnets 66–67
    creating NSG filtering rules 67–68
    creating NSGs 64–65
copy function, Resource Manager 84
copyIndex() function 84, 98, 102, 104
Cosmos DB 141–157
  accessing globally distributed data 152–156
  adding global redundancy to 149–152
  creating accounts and databases 145–152
  creating and populating databases 145–149
  deploying web app using 156–157
  overview of 141–144
    scaling databases 143–144
    structured (SQL) databases 142
    unstructured (NoSQL) databases 142–143
crash dumps 180
credentials 270
curl request 226–227
Custom Script Extension 179
custom SSL certificates 207

D

data center operating system (DC/OS) 293
data disks 49–50
data rests 208
data science virtual machines (DSVMs) 258
data scientists, tools for 257–259
database servers, vertical scale for 126
database_password variable 228
databases
  in Cosmos DB
    adding global redundancy to 149–152
    creating 145, 149–152
    populating 145–149
  scaling 143–144
DC/OS (data center operating system) 293
DDoS (distributed denial of service) 182
Decision service 259
default quotas 102
default-allow-ssh rule 241
delegating real domains 160–162
deleting protected VMs 205
deny state 238
DenyAll rules 185
DenyAllInBound rule 67, 184
dependencies 82
dependsOn 104
Deploy to Azure button 98
deploying HTML sites 39–42
deployment slots 44–46
diagnostic logs 42–44
direct traffic, routing 114–116
disaster recovery (DR) 201
disks
  adding to VMs 50–52
  caching options 50
  data disks 49–50
  temporary 49–50
distributed denial of service (DDoS) 182
DKIM (DomainKeys Identified Mail) 160
DNS resolution 62–64, 158
Docker 284, 287
Docker Swarm 293
Dockerfiles 291–292
DomainKeys Identified Mail (DKIM) 160
domains
  fault 96–97
  real, delegating to Azure DNS 160–162
  update 97–98
DR (disaster recovery) 201
DSC (Desired State Configuration) 179, 278, 282–283
DSVMs (data science virtual machines) 258
dynamic assignment 62

E

enableHttpsTrafficOnly parameter 210
encryption 206–215
  at rest 208–209
  of VMs 211–214
  overview of 206–208
  SSE (Storage Service Encryption) 209–210
  storing keys in Azure Key Vault 211–213
endpoint discovery 153
endpoint locations 153
endpoint monitor protocol 168
endpoint probing interval 168
endpoints 323
error messages 31
ETW (Event Tracing for Windows) 180
events endpoint 310, 312
ExpressRoute 19, 183
extensions 305

F

Face service 259
fault domains 96–97
Federal Information Processing Standard (FIPS) 218
File storage 53
file-level restore 199

www.dbooks.org

filtering 67–68
FIPS (Federal Information Processing Standard) 218
forum, for this book 5
FQDN (fully qualified domain name) 63
Free/Shared service plan 36
frontend IP pools 107–110
Function Apps 328–331
function key 332

G

Gardner, Lyza Danger 315
general-purpose VM sizes 17
geographic routing 163–164
georedundant storage (GRS) 56
Git 12
  deploying sample HTML site using 39–42
  learning 37
  password for, resetting 314
git push azure master command 156
git push dev master command 45
GitHub
  account for, creating 7
  Azure Automation and source control with 274
  Azure quick-start samples on 87
  overview of 39
  repo for this book 5
  resources 333
global redundancy 149–152
global routing, with Traffic Manager 162–173
  creating Traffic Manager profiles 164–166
  globally distributing traffic to closest instance 167–173
globally distributed data 152–156
Google Maps example 256
GPU (graphical processing unit) 267
GPU VM sizes 17
grouping resources 80–81
GRS (georedundant storage) 56
Guinard, Dominique D. 315

H

HashiCorp 86
health probes
  configuring 110–112
  creating 110–112
  overview of 107
high-performance SSDs 18
HPC (high-performance computing) 267
HSMs (hardware security modules) 212, 217–218
HTML sites, deploying 39–42
HTTP 20, 168, 206
HTTP path-based mode, health probes 110
HTTPS 20, 168, 206
Hyper-V 15

I

IaaS (Infrastructure as a Service) 9, 14, 33–34
IaC (infrastructure as code) 82
IIS (Internet Information Services) 29, 233
images, VM 16–17
IMDS (Instance Metadata Service) 222
incremental backups 193
Infrastructure as a Service (IaaS) 9, 14, 33
infrastructure as code (IaC) 82
infrastructure redundancy, with Availability Zones 95
  creating network resources across Availability Zones 94–95
  creating VMs in Availability Zones 95
injecting certificates 229–232
install command 27
installing web servers 24–27
Instance Metadata Service (IMDS) 222
instances, creating 290–292
interactive boot-console access 176
interface cards 61
internal load balancer 108
Internet Information Services (IIS) 29, 233
internet load balancer 108
interval parameter, health probes 111
iotconnectionstring variable 312
IP address ranges 60
IP flows, verifying 183–184
IP pools 107
IPv4 addresses 109
IPv4 host records 160
IPv6 addresses 109
IPv6 host records 160
isolated environments 36

J

JavaScript on Things (Gardner) 315
JIT (just-in-time) updates 237–249
jq parser 226
JSON (JavaScript Object Notation) 82–83, 86
JWT (JSON Web Token) 226

K

key pairs 20
keys
  creating key vaults 219–221
  storing encryption keys in Azure Key Vault 211–213

Kubernetes 293, 295–299
See also AKS
Kubernetes in Action (Luksa) 298

L

LAMP web server 27, 72
Language service 259
languages supported 34–35
LCM (Local Configuration Manager) 278
Learn Docker in a Month of Lunches (Stoneman) 297
Learn Git in a Month of Lunches (Umali) 37
learning materials 333
Let's Encrypt project 207
life cycles of apps 76–77
Linux
running Web Apps on 34
using DSC with 282–283
load balancers 94
components of 106–119
assigning groups of VMs to backend pools 116–119
creating frontend IP pools 108–110
defining traffic distribution with load-balancer rules 112–114
health probes 110–112
routing direct traffic with Network Address Translation rules 114–116
creating and configuring VMs with 119–122
defining traffic distribution with rules 112–114
in action 120–122
load-balancing applications 106–123
Local Configuration Manager (LCM) 278
locally redundant storage (LRS) 56
locks 79–80
Log Analytics workspaces 243
logic apps 35, 182, 325–328
logs. *See* diagnostic logs
Long Term Support (LTS) 22
LRS (locally redundant storage) 56
LTS (Long Term Support) 22
LUIS (Language Understanding Intelligent Service)
building Web App bots with 264–267
overview of 257–264
running Web App bots with 264–267
Luksa, Marko 298

M

machine learning. *See* ML
managed disks 18
Managed Object Format (MOF) file 280
Marketplace, Azure 7
Maven 12
memory (vRAM) 11
memory-optimized VM sizes 17
Message Analyzer, Microsoft 186
Message Text property 56
messaging platforms 319–325
Azure Event Grid 320–321
Azure Event Hubs 321–322
Azure Service Bus 321–322
creating service bus 322–325
integrating Service Bus with IoT hubs 322–325
metric alerts 182
Microsoft's Message Analyzer 186
ML (machine learning) 253–268
artificial intelligence and 254, 256
Azure Cognitive Services 259–260
LUIS (Language Understanding Intelligent Service) 261–264
overview of 255–256
relationship with artificial intelligence 257–259
tools for data scientists 257–259
Web App bots
building with LUIS 264–267
creating 260
running with LUIS 264–267
modules 270
MOF (Managed Object Format) file 280
monitoring 175
alerts 178–182
Azure Network Watcher 182–188
capturing network packets 186–188
verifying IP flows 183–184
viewing effective NSG rules 184–186
performance metrics 178–182
VM diagnostics 175–177
monolithic application 288
MSIs (managed service identities) 221–229

N

name server records 160
NAT (Network Address Translation) 107, 114–116
network connectivity (vNIC) 11
network interface cards (NICs) 61
network packets 186–188
network resources 94–95
network security groups. *See* NSGs
network traffic
managing 158–174
routing 158–174
Network Watcher 184
networking. *See* Azure Networking
NICs (network interface cards) 61, 117
–no-self-perms parameter 220

www.dbooks.org

NoSQL (unstructured databases) 142–143
NSGs (network security groups) 20
  associating with subnets 66–67
  creating 64–65, 118
  creating filtering rules 67–68
  in Azure Security Center 234
  overview of 112
  securing and controlling traffic with 64–68
  viewing effective rules 184–186
numbering systems, zero-based 99
nx module 283

O

OMS (Operations Management Suite) 243, 272
Owner role 78

P

PaaS (Platform as a Service) 10, 33, 37, 137
parallel VMs 102
parameters 82, 84, 89
performance conditions, alerts for 181–182
performance metrics 178–182, 188
performance routing 163–164
Perl programming language 34
Personalizer service 259
PHP 34
Platform as a Service (PaaS) 10, 33
pointer records 160
policies 193–196
  RPO (recovery point objective) 194–195
  RTO (recovery time objective) 195–196
pools
  backend 116–119
  frontend IP pools 108–110
populating databases 145–149
port-based mode, health probes 110
PowerShell DSC (Desired State Configuration) 278–282
  Azure Automation pull servers and 179, 280–282
  defining 280–282
PowerShell. *See* Azure PowerShell
Premium service plan 36
premium SSD (solid-state drive) disks 18–19
Priority routing method, Traffic Manager 163
private IP addresses 108
private key, of SSH key pair 71
production slot 46
profiles, in Traffic Manager 164–166
protected VMs, deleting 205
protecting resources 79–80
public IP addresses 20, 62–64, 94, 108
public key, of SSH key pair 20–22, 71
pull servers 280–282
Python programming language 28, 34

Q

Queue storage 53, 55–56
quotas 102, 132

R

RA-GRS (read-access georedundant storage) 57
Raspberry Pi 306–309
RBAC (role-based access controls) 78, 161, 184, 211
RDP (Remote Desktop Protocol) connection 20, 71
read-access georedundant storage (RA-GRS) 57
read-only cache policy 50
read/write caching 50
Reader role 78
readLocations 153
recovery points 193
recovery time objective (RTO) 193
redundancy
  benefits of 90–91
  of VMs with Availability Sets 96–102
  overview of 56–57
redundancy. *See also* infrastructure redundancy, with Availability Zones
remote access network connections 68–69
Remote Desktop Protocol (RDP) connection 20, 71
remotes 41
Representational State Transfer (REST) 31
resizing VMs 126–127
resolution, with Traffic Manager 162–173
  creating Traffic Manager profiles 164–166
  globally distributing traffic to closest instance 167–173
resource groups 315
resource types 84–85
resources 5, 7
  cleaning up 30
  controlling 78–79
  protecting with locks 79–80
  scaling horizontally 128–129
  securing 78–79
  with tags
    grouping 80–81
    managing 80–81
REST (Representational State Transfer) 31
REST APIs 161
restoring virtual machines 198–201
  complete VM restore 199–201
  file-level restore 199
retention 193–196
  RPO (recovery point objective) 194–195
  RTO (recovery time objective) 195–196
reviewing updates 245–249
role separation 62

role-based access control (RBAC) 78, 161, 211
routing direct traffic with Network Address Translation rules 114–116
RPO (recovery point objective) 193–195
RTO (recovery time objective) 193, 195–196
Run As accounts 272
runbooks, for Azure Automation 274–277
  executing 182
  overview of 272–274
  running 276–277
  viewing output from 276–277

S

SaaS (Software as a Service) 10
SAS (shared access signature) token 87
scalable apps 124–140
  benefits of 124–129
    scaling resources horizontally 128–129
    scaling VMs vertically 125–127
    scaling web apps vertically 127–128
  scaling web apps 136–139
  VM scale sets 129–136
    creating 131–133
    creating autoscale rules 133–136
scale sets, for VMs 129–136
  creating 131–133
  creating autoscale rules 133–136
scaling
  databases 143–144
  down VMs 127
  resources horizontally 128–129
  VMs vertically 125–127
    resizing VMs 126–127
    scaling down 127
  Web Apps
    overview of 136–139
    vertically 127–128
schedules 134, 270
Search service 259
secrets
  creating 219–221
  obtaining from within VMs with MSIs 224–229
secure traffic, building web apps with 68–72
  creating remote access network connections 68–69
  creating VMs 69–70
  using SSH agents to connect to VMs 70–72
securing
  resources 78–79
  traffic with NSGs 64–68
    associating NSGs with subnets 66–67
    creating NSG filtering rules 67–68
    creating NSGs (network security groups) 64–65
security 115
Security Center Overview window 236
Sender Protection Framework (SPF) 160
serial VMs 102
Server Message Block (SMB) 53
serverless computing 317–333
  creating function apps to analyze IoT device data 328–331
  creating logic apps 325–328
  GitHub resources 333
  messaging platforms 319–325
Azure Event Grid 320–321
    Azure Event Hubs 321–322
    Azure Service Bus 321–322
    creating service bus 322–325
    integrating Service Bus with IoT hubs 322–325
  overview of 317–319
Service Bus
  creating 322–325
  integrating with IoT hubs 322–325
service endpoints 146
service plans for apps 35–38
service principal 222
service records 160
service-level agreements (SLAs) 247
servicePrincipalName 224
session affinity mode 112–113
shared access signature (SAS) token 87
single-VM scale set 130
sinks 180
SLAs (service-level agreements) 247
SMB (Server Message Block) 53
Software as a Service (SaaS) 10
software vaults 217–218
SONiC (Software for Open Networking in the Cloud) 11
source code 5
Speaker Recognition service 259
Speech service 259
SPF (Sender Protection Framework) 160
SQL (Structured Query Language) 53, 142
SQL structured databases 142
SSE (Storage Service Encryption) 209–210
SSH (Secure Socket Shell)
  agents to connect to VMs 70–72
  connecting to VMs with 24–27
SSH key pairs 20–22
ssh-keygen command 21
SSL certificate 207
staging 41
standard HDD disks 18
Standard service plan 36
standard SSDs 18–19

www.dbooks.org

start-of-authority (SOA) records 160
static assignment 63
storage
  availability of 56–57
  in Azure 18
  in VMs 47–50
    data disks 49–50
    disk-caching options 50
    standard vs. premium storage 48–49
    temporary disks 49–50
  queue storage 55–56
  redundancy 56–57
storage (vDisk) 11
storage-optimized VM sizes 17
storing templates 87
streaming IoT hub data 309–315
streaming log files 43
structured data 144
Structured Query Language (SQL) 53, 142
Subnet level 185
subnets
  associating NSGs with 66–67
  creating 59
swap with preview 46
system-assigned managed identities 222

T

Table storage 52 tags
  grouping resources with 80–81
  managing resources with 80–81
templates, for Azure Resource Manager 81–87
  creating 82–84
  creating multiples of resource types 84–85
  storing 87
  tools to build 85–86
temporary disks 49–50
Terraform 86
Test in Web Chat option 266
third-party tools 86
threshold parameter, health probes 111
Time to Live (TTL) 167
traffic
  defining distribution with load-balancer rules 112–114
  globally distributing to closest instances 167–173
  routing direct traffic with Network Address Translation rules 114–116
  securing and controlling with NSGs 64–68
    associating NSGs with subnets 66–67
    creating NSG filtering rules 67–68
    creating NSGs 64–65
Traffic Manager
  creating profiles in 164–166
  deploying web apps to 174
  global routing and resolution with 162–173
  globally distributing traffic to closest instances 167–173
  Troubleshoot area 183
Translator Text service 259
Trifa, Vlad M. 315
troubleshooting 175
  alerts 178–182
  Azure Network Watcher 182–188
    capturing network packets 186–188
    verifying IP flows 183–184
    viewing effective NSG rules 184–186
  Azure platform 31–32
  performance metrics 178–182
  VM diagnostics 175–177
TTL (Time to Live) 167

U

Ubuntu Linux 14, 26
Umali, Rick 37
Universal Coordinated Time (UTC) 196
Unmount Disks option 199
unstructured data 144
update domains 96
Update Management Overview window 243
updates 234–249
  Azure Security Center NSGs 234
  Azure Update Management 241–249
    OMS (Operations Management Suite) 243
    reviewing and applying updates 245–249
  JIT (just-in-time) 237–241, 249
User access administrator role 78
user-assigned managed identities 222
UTC (Universal Coordinated Time) 196

V

variables 82, 84, 89, 271
vaults, key 219–221
verifying IP flows 183–184
VHD (virtual hard disk) 53
virtual CPU (vCPU) 11
virtual hard disk (VHD) 53
Virtual Machine Contributor role 79
virtual machines. *See* VMs
virtual networks 58–64
  creating 59
  creating subnets 59
  DNS resolution 62–64
  interface cards 61
  public IP addresses 62–64

Virtual NIC level 185
virtual private networks (VPNs) 19, 36, 38
virtualization 10–11
Vision service 259
Visual Studio editor 85–86
VMs (virtual machines)
  adding disks to 50–52
  allowing web traffic to reach 27–29
    creating rules to allow web traffic 28
    viewing web server in action 28–29
  assigning groups of to backend pools 116–119
  configuration 15–20
    Azure storage 18–19
    virtual networking 19–20
    VM images and 16–17
    VM sizes 17–18
  configuring with load balancers 119–122
  connecting to 120–122
    with SSH 24–27
    with SSH agents 70–72
  creating 14–32, 69–70
    cleaning up resources 30
    cost savings and 18
    from web browsers 22
    in Availability Zones 95
    troubleshooting Azure 31–32
    Windows VM 29–30
    with load balancers 119–122
  deallocating 30
  deleting 30
  deploying from templates 102–105
  diagnostic extensions 178
  diagnostics 175–177
  distributing across Availability Sets 98–101
  encryption of 211–214
    lab 214–215
    storing encryption keys in Azure Key Vault 211–213
  installing web servers 24–27
  obtaining secrets from with MSIs 224–229
  redundancy with Availability Sets 96–102
    fault domains 96–97
    update domains 97–98
  resizing 126–127
  restoring 198–201
    complete VM restore 199–201
    file-level restore 199
  scale sets 129–136
    creating 131–133
    creating autoscale rules 133–136
    installing applications on 139
  scaling down 127
  scaling vertically 125–127
  sizes of 17
  SSH key pair, creating for authentication 20–22
  storage 47–50
    data disks 49–50
    disk-caching options 50
    standard vs. premium storage 48–49
    temporary disks 49–50
  viewing distribution across Availability Sets 101–102
VMware 15
VPNs (virtual private networks) 19, 36, 38, 183

W

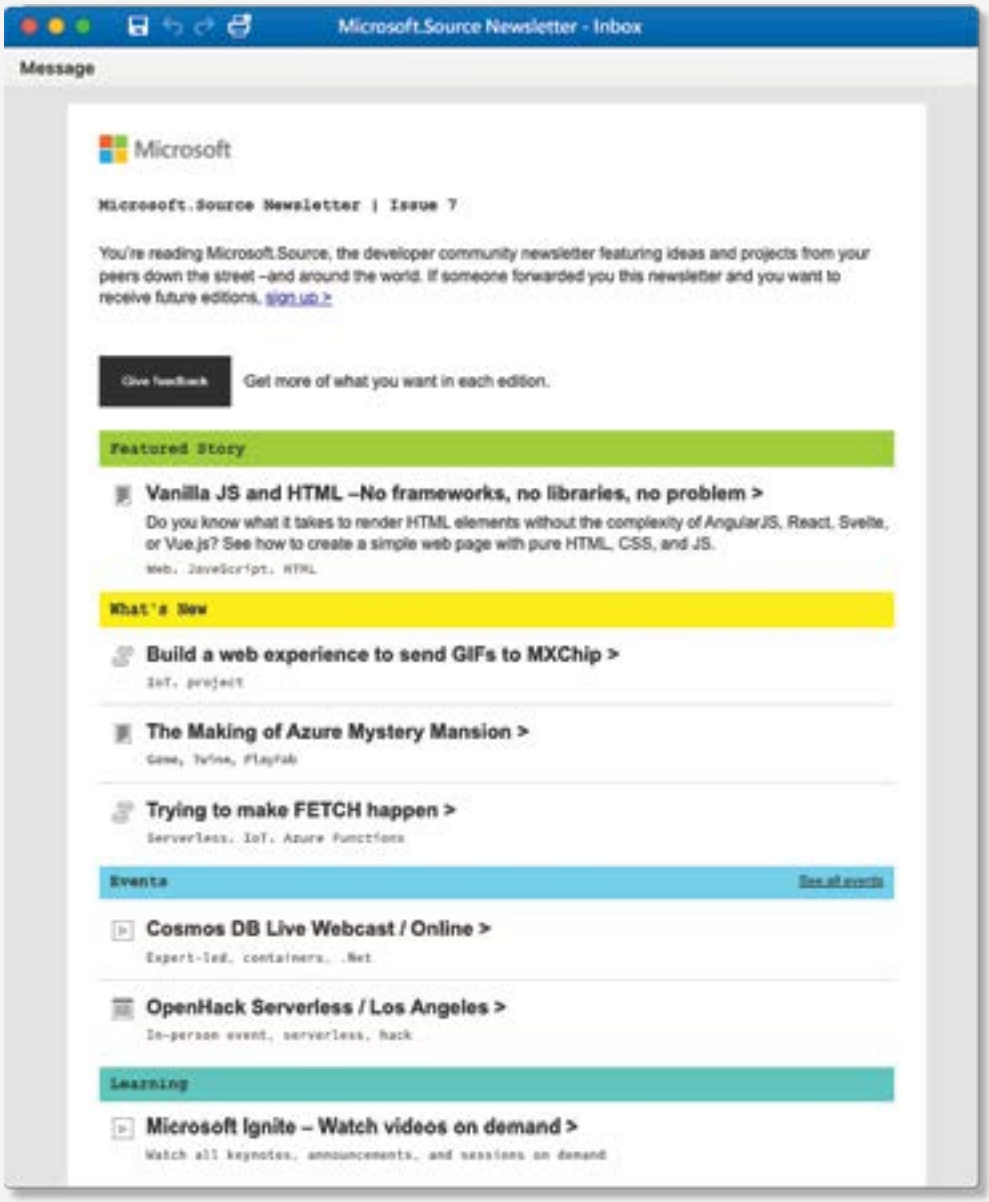
web browsers, creating VMs from 22
  Azure storage 18
  VM sizes 17
web servers
  in action 28–29
  installing 24–27
web traffic
  allowing to reach VMs 27–29
  creating rules to allow 28
webhooks 274
Website Contributor role 79
websites, running in Kubernetes 295–297
WebSockets 312
Weighted routing method, Traffic Manager 163
Windows, running Web Apps on 34

Z

zero-based numbering system 99
zonal services 93
–zone parameter 95
zone-redundant services 93
ZRS (zone-redundant storage) 56

www.dbooks.org

Microsoft



By developers, for developers

Microsoft.Source newsletter

Get technical articles, sample code, and information on upcoming events in Microsoft.Source, the curated monthly developer community newsletter.

- Keep up on the latest technologies
- Connect with your peers at community events
- Learn with hands-on resources

Sign up