

Programming Microsoft Azure Service Fabric

Second Edition

Professional



Haishi Bai

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Programming Microsoft Azure Service Fabric

Second Edition

Haishi Bai

PROGRAMMING MICROSOFT AZURE SERVICE FABRIC (Second edition)

Published with the authorization of Microsoft Corporation by:

Pearson Education, Inc.

Copyright © 2018 by Pearson Education, Inc.

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

No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-1-5093-0709-8

ISBN-10: 1-5093-0709-5

Library of Congress Control Number: 2018939252

1 18

TRADEMARKS

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

WARNING AND DISCLAIMER

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

SPECIAL SALES

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Editor-in-Chief: Greg Wiegand

Executive Editor: Laura Norman

Development Editor: Kate Shoup/
Polymath Publishing

Managing Editor: Sandra Schroeder

Senior Project Editor: Tracey Croom/
Danielle Foster

Copy Editor: Dan Foster

Indexer: Valerie Haynes Perry

Proofreader: Scout Festa

Editorial Assistant: Cindy Teeters

Cover Designer: Twist Creative, Seattle

Compositor: Danielle Foster

Graphics: Vived Graphics

To the entire Microsoft Service Fabric team, who made such a great product.

—HAISHI BAI

Contents at a Glance

Introduction

xxiii

PART I FUNDAMENTALS

CHAPTER 1	Hello, Service Fabric!	3
CHAPTER 2	Stateless Services	27
CHAPTER 3	Stateful Services	49
CHAPTER 4	Actor Pattern	67
CHAPTER 5	Service Deployments and Upgrades	91
CHAPTER 6	Availability and Reliability	117
CHAPTER 7	Scalability and Performance	145

PART II SERVICE LIFE CYCLE MANAGEMENT

CHAPTER 8	Service Fabric Scripting	173
CHAPTER 9	Cluster Management	199
CHAPTER 10	Diagnostics and Monitoring	213
CHAPTER 11	Continuous Delivery	239

PART III LINUX AND CONTAINERS

CHAPTER 12	Service Fabric on Linux	265
CHAPTER 13	Containers	279
CHAPTER 14	Container Orchestration	299

PART IV WORKLOADS AND DESIGN PATTERNS

CHAPTER 15	Scalable Web	325
CHAPTER 16	Scalable Interactive Systems	343
CHAPTER 17	System Integration	367

PART V ADVANCED TOPICS

CHAPTER 18	Serverless Computing	393
CHAPTER 19	Artificial Intelligence	409
CHAPTER 20	Orchestrating an Organic Compute Plane	437

PART VI APPENDICES

<i>Appendix A: Using Microsoft Azure PowerShell Commands</i>	<i>473</i>
<i>Appendix B: Pattern Index</i>	<i>479</i>
<i>Index</i>	<i>483</i>

Contents

<i>About the Author</i>	<i>xix</i>
<i>Foreword</i>	<i>xxi</i>
<i>Introduction</i>	<i>xxiii</i>

PART I FUNDAMENTALS

Chapter 1	Hello, Service Fabric!	3
	Microservices	3
	Containerization	3
	Scheduling	4
	State Reconciliation	4
	Data Replication	5
	Service Partitioning	5
	Service Fabric Concepts	6
	Architecture	6
	Nodes and Clusters	7
	Applications and Services	8
	Partitions and Replicas	8
	Programming Modes	9
	Stateless vs. Stateful	9
	Guest Applications and Containers	10
	Getting Started	10
	Setting Up a Development Environment in Windows	10
	Provisioning a Service Fabric Cluster on Azure	11
	Hello, World!	15
	Managing Your Local Cluster	19
	Visual Studio Cloud Explorer	19
	Service Fabric Explorer	21
	Windows PowerShell	24
	Service Fabric CLI	25
	Additional Information	26

Chapter 2	Stateless Services	27
	Implementing ASP.NET Core Applications	27
	Scalability and Availability of a Stateless Service	31
	Availability	31
	Scalability	32
	Implementing Communication Stacks	32
	Default Communication Stack	32
	WCF Communication Stack	41
	Custom Communication Stack	45
	Additional Information	48
Chapter 3	Stateful Services	49
	Architecture of Service Fabric Stateful Services	49
	Reliable Collections	50
	Reliable State Manager	50
	Transactional Replicator	51
	Logger	51
	Consistency	52
	The Simple Store Application	52
	The Shopping-Cart Service	52
	The Simple Store Website	56
	Service Partition	61
	Partitions and Replicas	63
	Replica Roles	63
	Resource Load Balancing	64
	Additional Information	66
Chapter 4	Actor Pattern	67
	Service Fabric Reliable Actors	68
	Actors	68
	Actor Lifetime	68
	Actor States	68
	Actor Communications	69
	Concurrency	69

An Actor-Based Tic-Tac-Toe Game.....	70
Actor Models.....	70
Creating the Application.....	71
Defining the Actor Interface.....	71
Implementing the Game Actor.....	72
Implementing the Player Actor.....	74
Implementing the Test Client.....	75
Testing the Game.....	77
Additional Thoughts.....	77
Timers, Reminders, and Events.....	78
Actor Timers.....	78
Actor Reminders.....	79
Actor Events.....	80
Service Diagnostics and Performance-Monitoring Basics.....	81
Event Tracing For Windows.....	81
Performance Counters.....	83
Actors and Reliable Services.....	87
Actor State Providers.....	88
Additional Information.....	90

Chapter 5 Service Deployments and Upgrades 91

The Service Fabric Application-Deployment Process.....	91
Package.....	91
Upload.....	96
Register/Provision.....	97
Create/Replace/Upgrade.....	97
The Service Fabric Health Model.....	97
Health States.....	99
Health Policy.....	99
Health Reporting and Aggregation.....	101
Rolling Upgrades.....	101
Upgrade Process.....	102
Upgrade Modes and Upgrade Parameters.....	104

Multiple Environments.....	107
Application Parameters and Parameter Files	107
Application Publish Profiles	108
Using Implicit Hosts	108
Defining Implicit Hosts.....	108
RunAs Policies.....	109
Hosting a Node.js Application.....	111
Resource Governance.....	115
Chapter 6 Availability and Reliability	117
“Broken” Services.....	117
Improving Availability	118
Improving Reliability.....	118
Service Fabric Services Availability.....	119
Replicas	119
Service Placements	120
Service Failovers.....	126
Routing and Load-Balancing	127
Advanced Rolling Upgrades.....	131
Service Fabric Services Reliability.....	133
Event Tracing for Windows.....	134
Azure Diagnostics	134
Chaos Testing	136
Service State Backup and Restore.....	141
Chapter 7 Scalability and Performance	145
Scalability Concepts	145
Vertical Scaling vs. Horizontal Scaling.....	145
Stateless Services vs. Stateful Services.....	146
Homogeneous Instances vs. Heterogeneous Instances	146
Single Tenancy vs. Multi-Tenancy.....	147
Manual Scaling vs. Autoscaling	148

Scaling a Service Fabric Cluster	151
Azure Resource Manager and Azure Virtual Machine Scale Sets	151
Manually Scaling a Service Fabric Cluster	152
Autoscaling a Service Fabric Cluster	154
Scaling with Content Delivery Network	157
Resolving Bottlenecks	159
State Bottlenecks	159
Communication Bottlenecks	164
Orchestration Bottlenecks	165

PART II SERVICE LIFE CYCLE MANAGEMENT

Chapter 8 Service Fabric Scripting 173

Azure Cloud Shell	173
Creating a Secured Service Fabric Cluster Using PowerShell	174
Using a Certificate to Protect Your Cluster	174
Using a Certificate for Client Authentication	178
Using Azure Active Directory for Client Authentication	178
Publishing Applications to a Secured Cluster from Visual Studio	180
Cluster Management Commands	181
Query Commands	181
Node Operations	190
Application-Management Commands	192
Deploying an Application	192
Upgrading an Application	193
Rolling Back an Application	194
Decommissioning an Application	195
Azure CLI	195
sfctl	197

Chapter 9	Cluster Management	199
	Anatomy of a Service Fabric Cluster	199
	Virtual Machine Scale Sets	200
	Virtual Machines and Virtual Network Cards	201
	Virtual Networks	202
	Load Balancers	203
	Storage Accounts.....	206
	Advanced Service Fabric Cluster Configuration.....	206
	Role-Based Access Control	206
	Network Security Groups	207
	Internal Load Balancer	211
	Updating Cluster Settings.....	212
Chapter 10	Diagnostics and Monitoring	213
	Diagnostics	213
	Diagnostics Data Pipeline	213
	Configure Azure Diagnostics	215
	Microsoft Diagnostics EventFlow	218
	Using Elasticsearch, Kibana, and EventFlow.....	221
	Azure Operations Management Suite.....	225
	Troubleshooting on Service Fabric Nodes.....	229
	Monitoring.....	229
	Service Fabric Explorer.....	230
	Application Insights.....	232
Chapter 11	Continuous Delivery	239
	CI, CD, and DevOps.....	239
	Continuous Integration	239
	Continuous Delivery	240
	DevOps	241
	Setting Up Continuous Integration	242
	Preparing the Visual Studio Team Services Project.....	242
	Creating a Build Definition	246

Setting Up Continuous Delivery	249
Creating a Release Definition	249
Requesting Deployment Approvals	251
Software Testability	252
Controllability	253
Observability	253
Isolability	254
Clarity	255
Setting Up Automated Tests	256
Implementing Unit Tests	256
Setting Up Gated Check-Ins	257
Running Load Tests with VSTS	259

PART III LINUX AND CONTAINERS

Chapter 12 Service Fabric on Linux 265

Service Fabric Hello, World! on Linux	265
Setting Up Your Linux Development Environment	265
Hello, World! Again	267
Using Communication Listeners	270
Other Service Types and Frameworks	274
Stateful Services	274
Actor Services	275
Guest Binary Services	276
Using Yeoman	277

Chapter 13 Containers 279

Docker Primer	279
Containerization on Linux	279
Windows Containers	281
Getting Started	282
Running Docker on Linux	282
Running Docker on Windows	286
Running Docker on Azure	286

Service Fabric and Docker.....	286
Hosting an ASP.NET Core Container on Windows.....	287
Hosting a Minecraft Server Container on Linux.....	290
Continuous Deployment with Jenkins.....	297

Chapter 14 Container Orchestration **299**

Microservices Application and Orchestration Engines	299
A Generic Microservice Application Model.....	299
Orchestration Engines	300
Container Orchestration with Service Fabric.....	301
DNS Service.....	301
Watchdogs.....	306
Docker Compose with Service Fabric	309
Defining the Master Image.....	310
Defining the Slave Image.....	311
Composing the Services with Docker Compose.....	312
Deploying and Testing the Application	312
Service Meshes.....	314
Envoy and Service Meshes	314
Deploying Envoy on Service Fabric	316

PART IV WORKLOADS AND DESIGN PATTERNS

Chapter 15 Scalable Web **325**

The Azure PaaS Ecosystem.....	325
App Services	325
Azure Container Service (AKS).....	326
Virtual Machine Scale Sets	326
Service Fabric	326
Choosing a PaaS Platform.....	327
Scaling with Reduction.....	328
CDN	329
Home Views.....	329
Caching.....	330

Precomputed Views	331
Data Manipulation.....	331
Scaling with Partition	332
Tenant Manager.....	332
Service Meshes (Part 2)	335
Scaling with Bursting	338
Designing an Extensible Control Plane.....	339
A Generic Control Plane Architecture	340
Workload Scheduling.....	342

Chapter 16 Scalable Interactive Systems 343

Interactive System Techniques.....	343
Latency	343
Throughput	347
CQRS and Event Sourcing	348
Basic Ideas Behind CQRS	348
Commands and Events.....	349
Event Sourcing	349
Real-Time Data-Streaming Pipelines.....	350
Composable Processing Pipelines	350
Implementing a Processing Sequence	351
Processing Topologies with Actors.....	357
Parallel Batching.....	358
Streaming Top N	358
Join by Field.....	359
Cached Lookup Grid	359
Exercise: Using WebSocket for Live Data Processing	360
Product Actor	361
Country/Region Actor	362
Global Actor.....	363
Gateway	364
WebSocket Listener.....	364
Test Client.....	365

Chapter 17 System Integration	367
Data Storage	367
Relational Databases	368
NoSQL Databases	371
Security	373
Azure Active Directory	373
Azure Key Vault	377
Enable SSL with Custom Domain	378
Integration with Service Brokers	379
Open Service Broker API	380
Open Service Broker for Azure	381
Service Fabric Service Catalog Service	381
Integration Patterns with Messaging	383
Dead-Letter Channel	383
Messaging Gateway	384
Transaction Coordinator	385
Message Translators	386
Composing Service Fabric Services	387

PART V ADVANCED TOPICS

Chapter 18 Serverless Computing	393
What Is Serverless Computing?	393
Serverless Deployment	393
Serverless Platform	394
Serverless Architecture	395
Benefits of Serverless	396
Serverless on Azure	397
Azure Container Instances	397
Azure Event Grid	398
Azure Functions	399
Azure Logic Apps	401

Reactive Messaging Patterns with Actors	403
Message-Driven Systems	403
Responsive Systems	403
Resilient Systems	406
Elastic Systems	406
Sea Breeze Design Principles	407
Fully Managed Environment	407
Container-Based Environment	408
Community Engagement	408

Chapter 19 Artificial Intelligence 409

A Brief Introduction to Artificial Intelligence	409
What Is AI?	409
Machine Learning	410
Neural Networks	411
Challenges and Pitfalls	411
Recommendations	413
Using Azure Machine Learning Studio	413
Calling the Service from Service Fabric	417
Using the Cognitive Services Recommendation API	418
Computer Vision	418
Building an OCR Application	419
Exploring Image-Analysis Applications	421
Natural Language Processing	422
Audio Transcription	423
Understanding the User's Intention	425
Conversational UI	428
Using the Bot Framework and Bot Service	428
Embedding a Web-Based Bot UI in Your Application	430
ArchiBot	431
TensorFlow and Service Fabric	432
Deploying a TensorFlow Cluster Using Service Fabric	433
Running a Clustered Jupyter Notebook with TensorFlow Containers	434

Chapter 20 Orchestrating an Organic Compute Plane 437

Moving Data Through Static Compute..... 437

- Data Generation and Feedback..... 438
- Command and Control 438
- Data Ingress..... 439
- Data Transformation and Analysis 439
- Storage 440
- Presentation and Actions 441
- Sample Patterns with Static Compute..... 441
- An End-to-End Scenario 448

Moving Compute to Data 462

- Service Fabric on Edge..... 463
- Workload Distribution 466

Closing Thoughts 469

PART VI APPENDICES

Appendix A: Using Microsoft Azure PowerShell Commands 473

Appendix B: Pattern Index 479

Index..... 483

About the Author



HAISHI BAI, principal software engineer at Microsoft, focuses on the Microsoft Azure compute platform, including IaaS, PaaS, networking, and scalable computing services.

Ever since he wrote his first program on an Apple II when he was 12, Haishi has been a passionate programmer. He later became a professional software engineer and architect. During his 21 years of professional life, he's faced various technical challenges and a broad range of project types that have given him rich experiences in designing innovative solutions to solve difficult problems.

Haishi is the author of a few cloud computing books, and he's an active contributor to a few open-source projects. He also runs a technical blog (<http://blog.haishibai.com>) with millions of viewers. His twitter handle is @HaishiBai2010.

Foreword

Service Fabric can be traced back to 2001, when I was trying to solve large-scale distributed system challenges such as leader election, quorum-based replication, and perfect failure detection. As I worked on projects in 2007, such as CloudDB (which morphed into Azure DB and is powered by Service Fabric), it became clear that a generic platform would be a valuable asset for empowering developers and enterprises to implement scalable and highly available distributed systems. In 2009, I began creating such a system to support large first-party and third-party workloads. This effort led to Service Fabric, which has proven itself in production for more than a decade. Service Fabric powers critical Microsoft services such as Azure DB, Cosmos DB, Skype for Business, Microsoft Intune, Azure Resource Providers (Compute RP, Storage RP, and Network RP), Azure Software Load Balancer, Azure Network Manager, Event Hubs, Event Grid, IoT Hubs, Azure Incident Manager, Azure Monitor, Bing Cortana, and more. As of March 2018, Service Fabric runs on about 4 million cores and individually monitors and fully lifecycle-manages about 10 million microservices. Technologies like Service Fabric are treated as trade secrets by many companies and are not made available to external customers. In March 2015, Service Fabric was released to the public. By making Service Fabric publicly available (and open-sourced in March 2018), Microsoft is living by the “first party == third party” principle.

Service Fabric is a comprehensive platform for building Internet-scale, high-throughput, low-latency services. Besides container orchestration, it solves many fundamental distributed systems problems, such as failure detection, replicated state machines, reliable message delivery, and so on. It allows developers to naturally decompose the business application into a logical set of microservices that are individually responsible for a single business function and interact with one another over well-defined protocols for implementing business workflows. Service Fabric lets developers focus on business logic and its associated state by abstracting away machine and distributed systems details with many built-in transactionally consistent reliable data structures like dictionaries and queues that survive process crashes and machine failures. It enables programmers to think and program exactly like they do today by replacing locks with transactions. Programmers assume that the process hosting their code never crashes, and the data structures storing their state never lose or corrupt their data. With its ability to run on any OS and on any cloud, including on-premises and Edge, developers preserve their code investments across a wide variety of deployment targets. Put differently, Service Fabric allows programming large-scale applications to be just like writing simple applications.

I have known Haishi for about three years now. He has keenly followed the evolution of Service Fabric as a public service and has a deep understanding of its developer and operational aspects. He also understands intuitively how various Service Fabric layers and subsystems combine to provide the solutions to many distributed-systems problems. His first edition of this book focused on Service Fabric programming models and design patterns. This second edition is a more comprehensive follow-up companion book on Service Fabric that focuses more on the developer and operational aspects of Service Fabric and newer parts of the Service Fabric-like containers, Linux support, and more.

If you are interested in microservices and stateless and stateful variants and are using or intend to use Service Fabric for developing microservices, this a must-read book for you.

—Gopal Kakivaya
CVP, Microsoft Azure Development

Introduction

Azure Service Fabric is Microsoft's platform as a service (PaaS) offering for developers to build and host available and scalable distributed systems. Microsoft has used Service Fabric for years to support some of Microsoft's cloud-scale applications and Azure services such as Skype for Business, Cortana, Microsoft Intune, Azure SQL Database, and Azure Cosmos DB. The same platform is now available as an open-source project for you to write your own highly available and highly scalable services.

Programming Microsoft Azure Service Fabric is designed to get you started and quickly productive with Azure Service Fabric. This book covers fundamentals, practical architectures, and design patterns for various scenarios, such as intelligent cloud, intelligent edge, big data, and distributed computing. For the fundamentals, this book provides detailed step-by-step walkthroughs that guide you through typical DevOps tasks. For design patterns, this book focuses on explaining the design philosophy and best practices with companion samples to get you started and moving in the right direction.

Instead of teaching you how to use Azure Service Fabric in isolation, the book encourages developers to make smart architecture choices by incorporating existing Azure services. When appropriate, this book briefly covers other Azure services relevant to particular scenarios.

Who should read this book

This book is intended to help new or experienced Azure developers get started with Azure Service Fabric. This book is also useful for architects and technical leads using Azure Service Fabric and related Azure services in their application architecture.

Service Fabric is under continuous development, and its momentum will only accelerate by community contributions. The second edition of this book offers the latest development of Service Fabric at the time of this writing. For the latest updates, consult the book's companion resource repository (<https://github.com/Haishi2016/ProgrammingServiceFabric>) and Service Fabric's online documentation (<https://docs.microsoft.com/azure/service-fabric>). Although the precise operational steps and programming APIs might change, the design patterns presented in this book should remain relevant into the foreseeable future.

Assumptions

This book expects that you are proficient in .NET, especially C# development. This book covers a broad range of topics and scenarios, especially in later chapters. Prior understanding of DevOps, application life cycle management (ALM), IoT, big data, and machine learning will help you get the most out of this book.

Although no prior Azure knowledge is required, experience with the Azure software development kit (SDK), Azure management portal, Azure PowerShell, Azure command line interface (CLI), and other Azure services definitely will be helpful.

This book might not be for you if...

This book might not be for you if you are a beginner in programming. This book assumes you have previous experience in C# development and ASP.NET development. Although this book covers topics in service operations, its primary audience is developers and architects, not IT professionals.

Organization of this book

This book is divided into five parts, each of which focuses on a different aspect of Azure Service Fabric. Part I, “Fundamentals,” provides complete coverage of designing and developing Service Fabric applications using stateless services, stateful services, and reliable actors. Part II, “Service Life Cycle Management,” focuses on the operations side and introduces how to manage Service Fabric clusters and how to manage, test, and diagnose Service Fabric applications. Part III, “Linux and Containers,” introduces Service Fabric Linux programming with Java and support for Docker containers. Part IV, “Workloads and Design Patterns,” introduces patterns and scenarios including practical design patterns and best practices in implementing typical application scenarios including scalable web applications, IoT, big data, and multi-tenant applications. Finally, Part V, “Advanced Topics,” covers three advanced topics: serverless computing, machine learning, and the intelligent cloud and the intelligent edge.

Finding your best starting point in this book

This book is an introduction to Service Fabric. It is recommended that you read the chapters in the first two parts sequentially. Then, you can pick the topics that interest you in Parts III, IV, and V.

If you are	Follow these steps
New to Service Fabric	Read through Part I and Part II in order.
Interested in applying Service Fabric in IoT scenarios	Focus on Chapter 20.
Interested in building scalable web applications	Focus on Chapters 7, 15, and 16. You may also want to read Chapter 17 for integrations with other Azure services, and Chapter 18 for serverless options.
Interested in machine learning	Focus on Chapter 19.
Interested in operating a Service Fabric cluster	Chapters 8, 9, 10, and 11 introduce related tools and services. You may also want to browse through Chapters 5, 6, and 7 to understand Service Fabric application characteristics.
Interested in the actor programming model	Focus on Chapter 4. Also browse through chapters in Part III because these chapters cover several actor-based design patterns.
Interested in Service Fabric container integration	Focus on Chapters 13 and 14.
Interested in Service Fabric Linux development	Focus on Chapter 12.

Some of this book’s chapters include hands-on samples that let you try out the concepts just learned. Regardless of which sections you choose to focus on, be sure to download and install the sample applications on your system (see the “Downloads” section on the next page).

System requirements

You will need the following hardware and software to run the sample code in this book:

- Windows 8/Windows 8.1, Windows Server 2012 R2, or Windows 10.
- Visual Studio 2015 or Visual Studio 2017.
- The latest Service Fabric SDK for Visual Studio 2015 or Visual Studio 2017 (install via Web PI).
- The latest version of Azure SDK (2.8 or above, install via Web PI).
- The latest version of Azure PowerShell (1.0 or above, install via Web PI).
- The latest version of Azure CLI.
- 4 GB (64-bit) RAM.

- 50 GB of available hard disk space.
- An active Microsoft Azure subscription. You can get a free trial from www.azure.com.
- An Internet connection to use Azure and to download software or chapter examples.
- An Ubuntu 14.0 or above machine or virtual machine for Linux-based exercises.

Depending on your Windows configuration, you might require Local Administrator rights to install or configure Visual Studio 2015, Visual Studio 2017, and related SDKs and tools.

Downloads: Code samples

Most of the chapters in this book include exercises that let you interactively try out new material learned in the main text. All sample projects can be downloaded from the following page:

<https://aka.ms/AzureServFabric2e/downloads>

You can also find the latest sample projects from the book's companion source repository:

<https://github.com/Haishi2016/ProgrammingServiceFabric>

Using the code samples

This book's webpage contains all samples in this book, organized in corresponding chapter folders. It also contains a V1-Samples folder that contains samples from the original version of the book, thanks to Alessandro Avila's contributions.

- **Chapter 1** This folder contains samples from Chapter 1.

HelloWorldApplication: The "Hello, World!" application.

- **Chapter 2** This folder contains samples from Chapter 2.

CalculatorApplication: The calculator application used in the communication stack samples.

gRPCApplication: The calculator application using gRPC.

StatelessApplication: The ASP.NET Core Web API stateless application.

- **Chapter 3** This folder contains samples from Chapter 3.
SimpleStoreApplication: The simple store application.
- **Chapter 4** This folder contains samples from Chapter 4.
ActorTicTacToeApplication: The tic-tac-toe game using actors.
CarApplication: The simple car simulation program using actors.
- **Chapter 5** This folder contains samples from Chapter 5.
ConsoleRedirectTestApplication: The sample application used in package format samples.
HelloWorldWithData: The sample application with a data package.
NodeJsHelloWorldApplication: The sample application hosts a Node.js application.
ResourceGovernance: The sample application with resource governance policy.
UpgradeProcess: The rolling update sample application.
- **Chapter 6** This folder contains samples from Chapter 6.
BadApplication: The sample unreliable application using in-memory states.
ChaosTest: The sample application that drives a chaos test.
ConfigurationUpdate: The sample application that responds to configuration updates.
Diagnostics: The sample application used in the Azure Diagnostics sample.
FailoverTest: The sample application used in the failover sample.
- **Chapter 7** This folder contains samples from Chapter 7.
CustomSerializerTest: The sample application that uses a custom serializer.
- **Chapter 8** This folder contains samples from Chapter 8.
DeploymentTest: The application used in the deployment sample.
- **Chapter 10** This folder contains samples from Chapter 10.
ApplicationInsightsTestApplication: The sample application that uses Application Insights.

CustomHealthReportApplication: The sample application that sends custom health reports.

EventFlowTestApplication: The sample application that uses Event Flow for diagnostics.

OMSTestApplication: The application used in the OMS sample.

- **Chapter 11** This folder contains samples from Chapter 11.

SudokuApplication: The sample Sudoku application.

- **Chapter 12** This folder contains samples from Chapter 12.

ActorApplication: The sample actor application in Java.

CalculatorApplication: The sample calculator application in Java.

GuestPythonApplication: The sample Python application hosted as a guest executable.

HelloWorldLinux: The "Hello, World!" application in Java.

StatefulApplication: The sample stateful application in Java.

- **Chapter 13** This folder contains samples from Chapter 13.

Docker-HelloWorld-Windows: The sample ASP.NET Core web application in Windows container.

DockerCompose-HelloWorld: The sample container application described by a Docker Compose file.

Minecraft: The HA Minecraft server deployment using persistent volume.

- **Chapter 14** This folder contains samples from Chapter 14.

IrisApp: The sample two-tiered application that uses the Python Sklearn library in containers.

ServiceMesh: The service mesh sample that uses Envoy.

Watchdog: The sample service that uses a C# watchdog to monitor a containerized sample Java application.

- **Chapter 15** This folder contains samples from Chapter 15.
PortSharing: The port sharing sample application.
TenantManager: The sample implementation of the tenant manager pattern.
- **Chapter 16** This folder contains samples from Chapter 16.
ECommerceApp: The sample e-commerce application.
NumberConverterApp: The number converter service.
- **Chapter 17** This folder contains samples from Chapter 17.
AudioTranscriptionApp: The audio transcription application that uses Bing Speech API.
- **Chapter 18** This folder contains samples from Chapter 18.
aci: The Azure Container Instance example.
ReactiveActors: The sample reactive application using actors.
- **Chapter 19** This folder contains samples from Chapter 19.
ArchiBot: The architecture bot sample that uses Bot Framework and Cosmos DB.

To complete an exercise, access the appropriate chapter folder in the root folder and open the project file. If your system is configured to display file extensions, C# project files use the .csproj file extension.

Errata, updates, & book support

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

<https://aka.ms/AzureServFabric2e/errata>

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

If you need additional support, email Microsoft Press Book Support at mspinput@microsoft.com.

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

Stay in touch

Let's keep the conversation going! We're on Twitter: <http://twitter.com/MicrosoftPress>.

PART III

Linux and Containers

Chapter 12	Service Fabric on Linux	265
Chapter 13	Containers	279
Chapter 14	Container Orchestration	299

Service Fabric provides a microservices programming model that you can use to build native cloud microservices applications. However, as discussed in Part I, implementing microservices doesn't mandate a specific programming model. If an application can be packaged in a self-contained format and be consistently deployed on different environments, it can enjoy many cloud benefits, such as failover, scaling, and load balancing. While packaging application artifacts isn't hard, making sure the application has all its dependencies is not easy. A legacy application may have dependencies on external libraries and services; it may assume specific folder structures; it may require specific environment variables; and for Windows, it may depend on certain registry values. It's impossible to isolate and package such system-level dependencies by a simple file-based package mechanism.

One way to package these system-level dependencies is to use desired state configuration (DSC). With DSC, such dependencies are captured as solution-specific metadata. When an application is deployed, the metadata is checked against the actual host environment. If any discrepancies are found, predefined scripts are executed to bring the host environment into the desired state. For example, if an application requires a specific DirectX version, the requirement can be captured as

DSC metadata. And when the application is deployed on a new host, the DSC system will check whether DirectX is installed. If not, it will run predefined scripts (most DSC systems have “ingredients” that perform common configuration tasks such as installing software packages) to make sure the specific version of DirectX is installed. Still, bringing an arbitrary machine state to a desired state is a hard job. In some cases, two applications are simply incompatible and cannot be installed on the same host. For example, if application A and application B use a `c:\data` folder, they can’t co-exist on the same host without interfering with each other.

Virtualization is an effective way of addressing these challenges. With virtualization, an application resides on a virtualized operating system that virtualizes process spaces, file systems, and registries (for Windows). In such an environment, because the application has exclusive access to the entire virtualized environment, it doesn’t need to worry about any potential conflicts with any other applications. The use of virtual machines (VMs) is a mature virtualization technique that has been broadly used in both on-premises and cloud datacenters. However, VMs are quite heavy. They require significant resources, and it takes a long time to provision, update, and deprovision a VM. Hence, VMs don’t provide the application mobility microservices requires.

Containers provide fine-grained isolation by isolating processes, files, and registries. Containers running on the same host share the same system kernel and can be launched and destroyed in the same way as regular processes. For Linux systems, this means sub-second launch times (Windows containers take longer to launch, but it’s still much faster than booting up a VM.) You can also pack many containers on the same host to gain high compute density. Containers are perfect for microservices because they package applications into lightweight, isolated, and consistently deployable units. This is the exact application mobility microservices requires to perform failovers, replications, scaling, and load balancing.

Given how powerful containers are, it makes sense for Service Fabric to provide native support for them. Furthermore, as a microservices platform, Service Fabric needs to embrace microservices that are not written using the Service Fabric programming model. In the past two years, Service Fabric has built up first-class support for Linux and containers. This will be the focus of this part of the book.

Service Fabric on Linux

Service Fabric running on Linux may come as a surprise. Why would Microsoft invest in non-Windows platforms? The reality, however, is that Service Fabric on Linux resonates with the openness of Azure strategy. Azure has never been just for Microsoft technologies. It's an inclusive platform that welcomes all types of workloads on all technical stacks. In fact, a huge portion of Azure compute power resides on Linux. The percentage of Linux VMs on Azure was 33% in 2016 and 40% in 2017, and it continues to increase.

As a microservices platform, Service Fabric must embrace not only Windows-based workloads, but also Linux-based workloads. In the past two years, the Service Fabric team has built up native Linux support with Java tooling. This chapter introduces the Linux development experiences using the Service Fabric programming model. Chapter 13 and Chapter 14 focus on containers and container orchestrations.



Note At the time of this writing, Service Fabric on Linux has just recently become generally available. Tooling experiences and product behaviors are subject to change. Please visit this book's companion GitHub repository for updated samples.

Service Fabric Hello, World! on Linux

The Service Fabric team has chosen Java as its primary programming language on Linux. In this section, you'll learn how to set up a development environment on Linux and use Java to write a simple Service Fabric application.

Setting Up Your Linux Development Environment

Follow these steps to set up your Linux development environment. (These instructions are based on Ubuntu 16.04.)

1. To install the Service Fabric runtime, Service Fabric common SDK, and a sfctl CLI, use the following script:

```
sudo curl -s https://raw.githubusercontent.com/Azure/service-fabric-scripts-and-templates/master/scripts/SetupServiceFabric/SetupServiceFabric.sh | sudo bash
```

2. Use this script to set up a local cluster:

```
sudo /opt/microsoft/sdk/servicefabric/common/clustersetup/devclustersetup.sh
```

3. After the cluster has been configured, open a web browser and navigate to <http://localhost:19080/Explorer>. The Service Fabric Explorer should open. Or, if you want to try out `sfctl`, you can use the following command to obtain a list of node names:

```
sfctl cluster select --endpoint http://localhost:19080
sfctl node list | grep name
```

4. Service Fabric uses Yeoman to scaffold Service Fabric applications. Yeoman is an open-source tool originally designed for this task. Use the following commands to install and configure Yeoman with Service Fabric application generators:

```
sudo apt-get install npm
sudo apt install nodejs-legacy
sudo npm install -g yo
sudo npm install -g generator-azuresfcontainer
sudo npm install -g generator-azuresfguest
```

5. If you plan to build Service Fabric services using Java, use the following commands to install JDK 1.8 and Gradle:

```
sudo apt-get install openjdk-8-jdk-headless
sudo apt-get install gradle
```

6. For the IDE, Service Fabric chose Eclipse for Java development. To install Eclipse, download the package from www.eclipse.org/downloads/eclipse-packages (this book uses the Oxygen.1 version), extract all files from the package, and launch **eclipse-inst**.
7. Service Fabric provides an Eclipse plug-in to facilitate application creation. After you install Eclipse, launch it, open the **Help** menu, and choose **Install New Software**.
8. In the **Work With** box, type <http://dl.microsoft.com/eclipse>. Then click the **Add** button.
9. Select the **ServiceFabric plug-in**. Then follow the wizard to install the plug-in.



Note If you want to enable the desktop UI on your Azure Ubuntu VM, use the following commands (tested on Ubuntu 16.04):

```
sudo apt-get install xrdp
sudo apt-get update
sudo apt-get install xfce4
echo xfce4-session >> ~/.xsession
sudo service xrdp restart
```

Then configure networking on the VM to allow inbound RDP connections through port 3389. After that, you should be able to connect to your Ubuntu desktop using RDP.

Hello, World! Again

Now it's time to send greetings to a brand-new world. In the following exercise, you'll create a new Service Fabric stateless service using Eclipse. Follow these steps:

1. In Eclipse, open the **File** menu, choose **New**, and select **Other**.
2. Expand the **Service Fabric** folder, select **Service Fabric Project**, and click **Next**. (See Figure 12-1.)

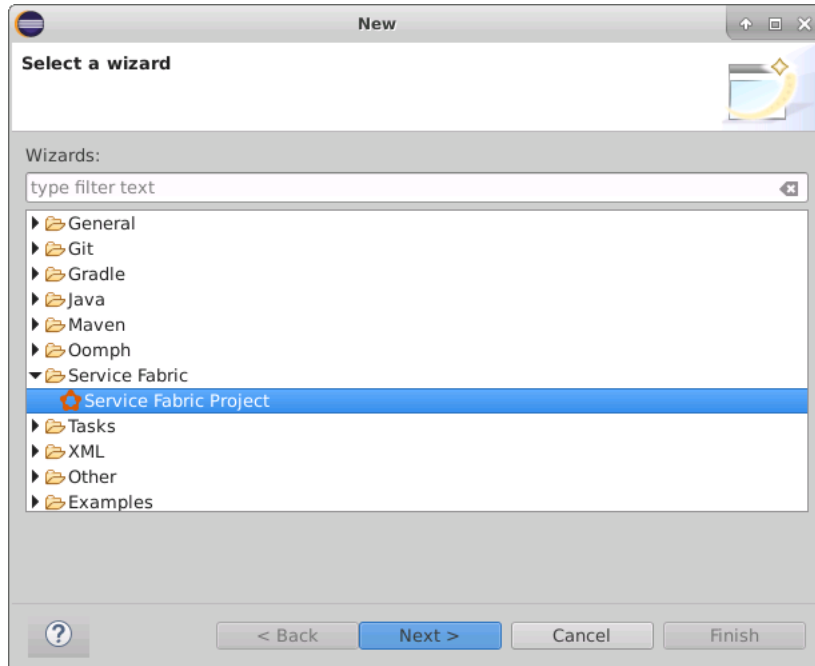


FIGURE 12-1 The Eclipse New Project wizard.

3. In the next screen, type **HelloWorldLinux** in the **Project Name** box and click **Next**.
4. In the Add Service screen, select the **Stateless Service** template, type **HelloWorldService** in the **Service Name** box, and click **Finish** to add the service. (See Figure 12-2.)

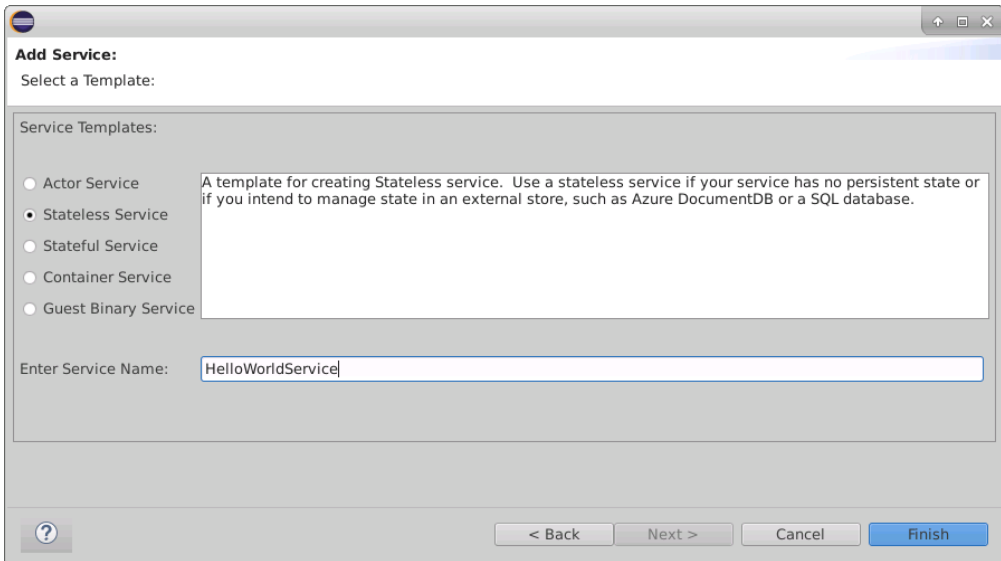


FIGURE 12-2 Add a service.

- When the wizard prompts you to open the Service Fabric perspective, click **Open Perspective**.
- After the application is created, poke around the package tree to familiarize yourself with the package structure.

Fortunately, the Java project is similar to a C# project. You can see how instance listeners are created, how the `runAsync` method is implemented in `HelloWorldServiceService.java`, and how the service is registered in `HelloWorldServiceServiceHost.java`. (See Figure 12-3.)

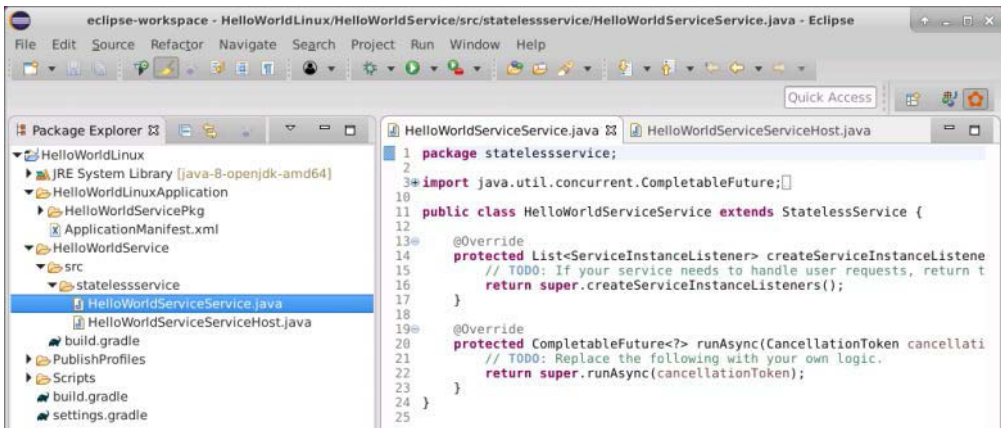


FIGURE 12-3 Java package structure.

7. In Package Explorer, right-click **HelloWorldLinux**, select **Service Fabric**, and choose **Deploy Application** to build and deploy the application. After the application is deployed, you should see your service instance running through the Service Fabric Explorer.



Note Provisioning a Service Fabric cluster consumes about 22 GB of your system drive. If you are running low on disk space, you may want to move the cluster to a different volume. Here's how:

```
cd /opt/microsoft/sdk/servicefabric/common/clustersetup
sudo ./devclustercleanup.sh
sudo rm -rf /home/sfuser/sfdevcluster
sudo ./devclustersetup.sh --clusterdataroot=/some/other/volume
```

8. Open **HelloWorldService\src\statelesservice\HelloWorldServiceService.java** and examine the stateless service implementation. The scaffolded implementation isn't exciting; indeed, it doesn't do anything:

```
protected CompletableFuture<?> runAsync(CancellationTok en cancellationTok en) {
    return super.runAsync(cancellationTok en);
}
```

9. In a moment, you'll modify the `runAsync` method to make it behave in the same way as the default C# stateless service and maintain an incrementing local counter. You'll also add a `FileHandler` to record log entries into files under the services' log folder. First, though, you'll need to import the following namespaces:

```
import java.util.logging.FileHandler;
import java.util.logging.SimpleFormatter;
```

10. Issue the following command to add a logger as a static member of the `HelloWorldServiceService` class:

```
private static final Logger logger =
    Logger.getLogger>HelloWorldServiceService.class.getName());
```

11. Modify the `runAsync` method as shown in the following code:

```
@Override
protected CompletableFuture<?> runAsync(CancellationTok en cancellationTok en) {
    try
    {
        String logPath = super.getServiceContext()
            .getCodePackageActivationContext().getLogDirectory();
        FileHandler handler = new FileHandler(logPath
            + "/mysrv-log.%u.%g.txt", 1024000, 10, true);
        handler.setFormatter(new SimpleFormatter());
        handler.setLevel(Level.ALL);
        logger.addHandler(handler);
    } catch (Exception exp) {
        logger.log(Level.SEVERE, null, exp);
    }
}
```

```

return CompletableFuture.runAsync(() -> {
    try
    {
        int iteration = 0;
        while (!cancellationToken.isCancelled()) {
            logger.log(Level.INFO, "Working-" + iteration++);
            Thread.sleep(1000);
        }
    } catch (Exception exp) {
        logger.log(Level.SEVERE, null, exp);
    }
});
}

```

Note a couple of things in the preceding code:

- You can use the `getServiceContext()` method to obtain the service context, through which you can navigate an object tree that is very similar to the .NET object tree to obtain contextual information—for example to obtain the service’s log directory using the object model.
 - The `runAsync` method is supposed to return a `java.util.concurrent.CompletableFuture<T>` instance.
12. Build and deploy the application to your local cluster.
 13. Locate the log path of your service instance on the hosting node. You’ll find generated log files under the `services log` folder.

Using Communication Listeners

To create a service that listens to client requests, you need to create and register a `CommunicationListener` implementation. This process is very similar to what you did in Chapter 2. Perform the following steps to create a Java-based calculator service that provides a REST API for add and subtract calculations:

1. In Eclipse, create a new Service Fabric application named `CalculatorApplication` with a stateless service named `Calculator`.
2. Add a new `CalculatorServer` class to the project. This class contains nothing specific to Service Fabric. It uses `com.sun.net.httpserver.HttpServer` to handle add and subtract requests from clients.

```

package statelesservice;

import com.sun.net.httpserver.*;
import java.net.InetSocketAddress;
import java.io.IOException;
import java.io.OutputStream;
import java.io.UnsupportedEncodingException;
import java.util.HashMap;

```

```

import java.util.Map;

public class CalculatorServer {
    private HttpServer server;
    private int port;
    public CalculatorServer(int port) {
        this.port = port;
    }
    public void start() throws IOException {
        server = HttpServer.create(new InetSocketAddress(port),0);
        HttpHandler add = new HttpHandler() {
            @Override
            public void handle(HttpExchange h) throws IOException {
                byte[] buffer = CalculatorServer.handleCalculation
                    (h.getRequestURI().getQuery(), "add");
                h.sendResponseHeaders(200, buffer.length);
                OutputStream os = h.getResponseBody();
                os.write(buffer);
                os.close();
            }
        };
        HttpHandler subtract = new HttpHandler() {
            @Override
            public void handle(HttpExchange h) throws IOException {
                byte[] buffer = CalculatorServer.handleCalculation
                    (h.getRequestURI().getQuery(), "subtract");
                h.sendResponseHeaders(200, buffer.length);
                OutputStream os = h.getResponseBody();
                os.write(buffer);
                os.close();
            }
        };
        server.createContext("/api/add", add);
        server.createContext("/api/subtract", subtract);
        server.setExecutor(null);
        server.start();
    }
    public void stop() {
        server.stop(10);
    }
    public static Map<String, String> queryToMap(String query) {
        Map<String, String> map = new HashMap<String, String>();
        for (String param: query.split("&")) {
            String pair[] = param.split("=");
            if (pair.length > 1) {
                map.put(pair[0], pair[1]);
            } else {
                map.put(pair[0], "0");
            }
        }
        return map;
    }
    public static byte[] handleCalculation(String query, String type)
        throws UnsupportedEncodingException {
        byte[] buffer = null;
        Map<String, String> parameters = CalculatorServer.queryToMap(query);
        int c = 0;

```



```

try
{
    int a = Integer.parseInt(parameters.get("a"));
    int b = Integer.parseInt(parameters.get("b"));
    if (type.equals("add")) {
        c = a + b;
    } else {
        c = a - b;
    }
    buffer = Integer.toString(c).getBytes("UTF-8");
} catch (NumberFormatException e) {
    buffer = ("Invalid parameters").getBytes("UTF-8");
}
return buffer;
}
}

```

3. Add a `WebCommunicationListener` class to the project. This class implements `microsoft.servicefabric.services.communication.runtime.CommunicationListener` and overrides the `openAsync`, `closeAsync`, and `abort` methods.

```

package statelesservice;

import java.util.concurrent.CompletableFuture;
import java.io.IOException;
import microsoft.servicefabric.services.communication.runtime.CommunicationListener;
import microsoft.servicefabric.services.runtime.StatelessServiceContext;
import system.fabric.description.EndpointResourceDescription;
import system.fabric.CancellationToken;

public class WebCommunicationListener implements CommunicationListener {
    private StatelessServiceContext context;
    private CalculatorServer server;
    private String webEndpointName = "ServiceEndpoint";
    private int port;
    public WebCommunicationListener(StatelessServiceContext context) {
        this.context = context;
        EndpointResourceDescription endpoint =
            this.context.getCodePackageActivationContext().getEndpoint(
                webEndpointName);
        this.port = endpoint.getPort();
    }

    @Override
    public CompletableFuture<String> openAsync(CancellationToken cancellationToken) {
        CompletableFuture<String> str = new CompletableFuture<>();
        String address = String.format("http://%s:%d/api",
            this.context.getNodeContext().getIpAddressOrFQDN(), this.port);
        str.complete(address);
        try
        {
            {
                server = new CalculatorServer(port);
                server.start();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }
}

```

```

        return str;
    }

    @Override
    public CompletableFuture<?> closeAsync(Cancellation token cancellationToken) {
        CompletableFuture<Boolean> task = new CompletableFuture<>();
        task.complete(Boolean.TRUE);
        if (server != null) {
            server.stop();
        }
        return task;
    }

    @Override
    public void abort() {
        if (server != null) {
            server.stop();
        }
    }
}

```

4. Modify CalculatorService to return WebCommunicationListener from the overridden createServiceInstanceListeners method:

```

import java.util.ArrayList;
...
public class CalculatorService extends StatelessService {
    @Override
    protected List<ServiceInstanceListener> createServiceInstanceListeners() {
        ArrayList<ServiceInstanceListener> listeners = new ArrayList();
        listeners.add(new ServiceInstanceListener((context) -> {
            return new WebCommunicationListener(context);
        }));
        return listeners;
    }
}

```

5. Modify the CalculatorApplicationApplication\CalculatorPkg\ServiceManifest.xml file to define an endpoint resource named ServiceEndpoint:

```

<Resources>
  <Endpoints>
    <Endpoint Name="ServiceEndpoint" Protocol="http" Port="8182" />
  </Endpoints>
</Resources>

```

6. Build and deploy the application. Afterward, you should be able to use a browser and send requests such as `http://localhost:8182/api/add?a=100&b=200` and `http://localhost:8182/api/subtract?a=100&b=200`. You should also get corresponding outputs (300 and -100).

Other Service Types and Frameworks

Using Service Fabric Java SDK to implement other service types, including stateful services, actor services, and guest application services, is very similar to using the .NET SDK. Of course, because of language differences, the Java SDK is adapted to work more naturally for Java developers.



Note One productivity goal of the Service Fabric team is to enable popular and proven programming paradigms on Service Fabric. You can expect to see increasingly more languages and frameworks receive native support through Service Fabric tooling.

The next few sections provide a quick glimpse into how Java SDK supports different service types. To try out different service types, simply create a Service Fabric application and add a service with the type. The scaffolded code gives you quick examples on basic usage of the corresponding service types. You'll see a great similarity between the Java code and the .NET code.

Stateful Services

Interacting with state managers in Java SDK is slightly different from the .NET SDK. The Java SDK uses a `CompletableFuture<T>` type, which can be taken as an approximation of the .NET `Task<T>` type. Or, if you are familiar with promises, you can take it as an implementation of a promise. When you create a new stateful service, the SDK scaffolds a default `runAsync` method implementation, as shown in the following snippet:

```
@Override
protected CompletableFuture<?> runAsync(CancellationTokentoken) {
    Transaction tx = stateManager.createTransaction();
    return this.stateManager.<String, Long>getOrAddReliableHashMapAsync("myHashMap")
        .thenCompose((map) -> {
            return map.computeAsync(tx, "counter", (k, v) -> {
                if (v == null)
                    return 1L;
                else
                    return ++v;
            }, Duration.ofSeconds(4), cancellationToken).thenApply((l) -> {
                return tx.commitAsync().handle((r, x) -> {
                    if (x != null) {
                        logger.log(Level.SEVERE, x.getMessage());
                    }
                    try {
                        tx.close();
                    } catch (Exception e) {
                        logger.log(Level.SEVERE, e.getMessage());
                    }
                    return null;
                });
            });
        });
}
```

This code first creates a new transaction. Then, it tries to get or add a `microsoft.servicefabric.data.collections.ReliableHashMap<K, V>` instance, which is equivalent to `IReliableDictionary<K, V>` in the .NET SDK. Finally, it tries to create or update the "counter" entry in the map and commits the transaction.

Actor Services

Actor services with Java SDK work in the same way as .NET-based actor services. To construct and use an actor service in Java, you need the same set of artifacts as in the .NET SDK:

- **Actor interface** An actor interface is defined as a regular Java interface that inherits a default Actor interface—for example:

```
public interface MyActor extends Actor {
    @ReadOnly
    CompletableFuture<Integer> getCountAsync();
    CompletableFuture<?> setCountAsync(int count);
}
```

- **Actor implementation** An actor implementation inherits from a `microsoft.servicefabric.actors.FabricActor` base class and implements the actor interface. The following snippet shows that the actor implementation in Java corresponds almost line by line with the .NET implementation:

```
@ActorServiceAttribute(name = "MyActorActorService")
@StatePersistenceAttribute(statePersistence = StatePersistence.Persisted)
public class MyActorImpl extends FabricActor implements MyActor {
    private Logger logger = Logger.getLogger(this.getClass().getName());
    public MyActorImpl(FabricActorService actorService, ActorId actorId){
        super(actorService, actorId);
    }
    @Override
    protected CompletableFuture<?> onActivateAsync() {
        logger.log(Level.INFO, "onActivateAsync");
        return this.stateManager().tryAddStateAsync("count", 0);
    }
    @Override
    public CompletableFuture<Integer> getCountAsync() {
        logger.log(Level.INFO, "Getting current count value");
        return this.stateManager().getStateAsync("count");
    }
    @Override
    public CompletableFuture<?> setCountAsync(int count) {
        logger.log(Level.INFO, "Setting current count value {0}", count);
        return this.stateManager().addOrUpdateStateAsync("count", count,
            (key, value) -> count > value ? count : value);
    }
}
```

- **Actor proxy for a client to connect to an actor service** The Java SDK provides a `microsoft.servicefabric.actors.ActorProxyBase` class, through which you can create actor proxies for your actor interface:

```
MyActor actorProxy = ActorProxyBase.create(MyActor.class, new ActorId("From Actor 1"),
    new URI("fabric:/ActorApplicationApplication/MyActorActorService"));
int count = actorProxy.getCountAsync().get();
System.out.println("From Actor:" + ActorExtensions.getActorId(actorProxy)
    + " CurrentValue:" + count);
actorProxy.setCountAsync(count+1);
```

Guest Binary Services

As on the Windows platform, you can package a guest binary and host it on Service Fabric as a stateless service. The following steps show you how to package a Python-based application as a guest binary service. In this simple example, you'll create a web server using Flask and then host the Python application as a stateless service on your local Service Fabric cluster.

1. Create a new Service Fabric application named `GuestPythonApplication` with a guest binary service named `FlaskWebServer`.
2. When asked for guest binary details, simply click the **Finish** button. You'll add these files manually later.
3. Use the following code, which uses Flask to implement a very simple web server, to create a new `flaskserver.py` file in the `GuestPythonApplicationApplication\FlaskWebServerPkg\Code` folder:

```
from flask import Flask
app = Flask("myweb")
@app.route("/")
def hello():
    return "Hello from Flask!"
app.run(host='0.0.0.0', port=8183, debug = False)
```

4. In the same folder, add a new `launch.sh` file to launch the web server. The following script first installs the flask module using pip. Then, it locates the path of the current script and feeds the correct server file path to Python. Strictly speaking, installing Flask should have been done in a setup entry point because it's a host environment configuration step. I'll leave this exercise to interested readers.

```
#!/bin/bash
sudo python -m pip install flask >> ../log/flask-install.txt 2>&1
pushd $(dirname "${0}") > /dev/null
BASEDIR=$(pwd -L)
popd > /dev/null
logger ${BASEDIR}
python ${BASEDIR}/flaskserver.py
```

5. Update the ServiceManifest.xml file as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<ServiceManifest Name="FlaskWebServerPkg" Version="1.0.0" xmlns="http://schemas.
microsoft.com/2011/01/fabric" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Description>Service that implements a FlaskWebServer service</Description>
  <ServiceTypes>
    <StatelessServiceType ServiceTypeName="FlaskWebServerType" UseImplicitHost="true"/>
  </ServiceTypes>
  <CodePackage Name="Code" Version="1.0.0">
    <EntryPoint>
      <ExeHost>
        <Program>launch.sh</Program>
        <Arguments />
        <WorkingFolder>CodePackage</WorkingFolder>
      </ExeHost>
    </EntryPoint>
    <EnvironmentVariables></EnvironmentVariables>
  </CodePackage>
  <ConfigPackage Name="Config" Version="1.0.0" />
  <DataPackage Name="Data" Version="1.0.0" />
  <Resources>
    <Endpoints>
      <Endpoint Name="ServiceEndpoint" Protocol="http" Port="8183" Type="Input"/>
    </Endpoints>
  </Resources>
</ServiceManifest>
```

6. Build and deploy the application.
7. Using a web browser, navigate to <http://localhost:8183/>. You should see a "Hello from Flask!" message.

Using Yeoman

In addition to the Eclipse experience, Service Fabric provides a few generators that enable you to create Service Fabric applications using Yeoman. Yeoman (<http://yeoman.io/>) is an application scaffolding tool with an extensible generator ecosystem that hosts generators for various application types, including Service Fabric applications.

When you install the Service Fabric SDK, Yeoman is installed and configured automatically. To launch Yeoman, issue the `yo` command in a terminal.

To recreate the previous guest binary application in Yeoman, follow these steps:

1. Create a new `~/pythonflask` folder and copy the `flaskserver.py` and `launch.sh` files into it.
2. Use the `yo azuresfguest` command to launch Yeoman with the `azuresfguest` generator. You should see output like that shown in Figure 12-4.

Index

Numbers

404 errors, 129

A

ACI (Azure Container Instances), 338–339.

See also containers

ACLs (access control lists), 110

actor pattern, overview, 67

actor services, Linux, 275–276

actor states, state bottlenecks, 161–162

actors. *See also* tic-tac-toe game

APIs, 9

communications, 69

concurrency, 69–70

defined, 68

digital twins, 67

ETW (Event Tracking for Windows), 81–83

events, 80–81

instances, 87–88

lifetime, 68

performance counters, 83–87

reactive messaging patterns, 403–406

reminders, 79–80

state providers, 88–90

states, 68, 87–88

timers, 78–79

WebSocket for live data processing, 360–366

actors and processing topologies

cached lookup grid, 359–360

join-by-field, 359

overview, 357

parallel batching, 358

streaming top N, 358–359

AD (Active Directory)

client authentication, 178–180

OAuth, 374–375

security, 373–377

ws-Federation, 373–374

aggregation coordination pattern, 165–166,
443–448, 480

AI (artificial intelligence). *See also* Machine
Learning Studio

ArchiBot, 431–432

Bot Service and Bot Framework, 428–431

challenges and pitfalls, 411–412

image-analysis applications, 421–422

machine learning, 410–411

natural language processing, 422–425

neural networks, 411

OCR application, 419–421

overview, 409–410

TensorFlow compute engine, 432–435

users' intentions, 425–426

AKS (Azure Container Service), PaaS ecosystem, 326

APIs (application programming interfaces), 9

for autoscaling, 154–155

OSB (Open Service Broker) API, 380

Application Insights, 232–237

application manifest, 37

application-deployment

code package, 93–95

configuration package, 95

create/replace/upgrade, 97

data package, 95–96

folder structure, 92

implicit hosts, 108–114

package, 91–92

parameters and files, 107

publish profiles, 108

register/provision, 97

resource governance, 115

rolling upgrades, 101–107

upload, 96–97

application-level queries

- application-level queries, 186–187
- application-management commands, 192–197
- applications
 - decommissioning, 195
 - deploying, 192–193
 - rolling back, 194
 - and services, 8
 - upgrading, 193–194
- ArchiBot, 431–432
- architecture, Service Fabric, 6–7
- ARM (Azure Resource Manager), 151–152
- ASM (Azure Service Management) API, 475
- ASP.NET
 - core applications, 27–31
 - core container, 287–289
- audio transcription, natural language
 - processing, 423–425
- authentication
 - AD (Active Directory), 376–377
 - claim-based, 335, 481
- autoscaling
 - clusters, 154–157
 - explained, 480
 - vs. scaling, 148
- availability
 - advanced rolling upgrades, 131–133
 - batching load reports, 122
 - configuration and data changes, 131–133
 - DNS service, 129–130
 - improving, 118
 - Naming Service, 127–128
 - placement constraints, 122–123
 - replicas, 119
 - resource reserves, 126
 - resource-balancing metrics, 120–122
 - reverse proxy, 128–129
 - routing and load-balancing, 127–130
 - service affinity, 124–125
 - service defragmentation, 125–126
 - service failovers, 126–127
 - service placements, 120–126
 - upgrading with diff packages, 133
- Azure
 - running Docker, 286
 - VM (Virtual Machine) scale sets, 151–152
- Azure Cloud Shell, 26, 173
- Azure Diagnostics, 134–136
- Azure Insights, using for autoscaling, 155–157
- Azure Quick Start templates, 151–152
- Azure Service Fabric
 - applications and services, 8
 - architecture, 6–7
 - CLI (command-line interface), 25–26
 - Cloud Shell, 26
 - containerization, 3–4
 - containers, 10
 - data replication, 5
 - development environment, 10–11
 - documentation, 26
 - evolution, 6
 - guest applications, 10
 - Hello, World! 15–19
 - microservices, 3–6
 - nodes and clusters, 7–8
 - partitions and replicas, 8–9
 - programming modes, 9
 - provisioning cluster, 11–15
 - scheduling, 4
 - SDK, 10
 - Service Fabric Explorer, 21–24
 - service partitioning, 5
 - state reconciliation, 4
 - stateless vs. stateful, 9–10
 - subscription, 11
 - updates, 26
 - Visual Studio Cloud Explorer, 19–21
 - Windows PowerShell, 24–25
- B
 - backup and restore, service state, 141–144
 - batching, 122, 165, 480
 - blades, 11
 - Blob Storage, using, 371
 - Bot Service and Bot Framework, 428–432
- bottlenecks
 - communication bottlenecks, 164–165
 - orchestration bottlenecks, 165
 - resolving, 159
 - state bottlenecks, 159–163
 - throughput and response time, 159
- broken services, 117. *See also* services
- bronze durability tier, 149
- bronze reliability tier, 150

C

- cached lookup grid topology, 359–360
- calculator service
 - creating, 33–35
 - gRPC framework, 45–48
 - replica ID, 39–41
 - WCF clients, 42–44
- Cassini case study, 347–348
- CD (continuous delivery)
 - CI (continuous integration), 239–240
 - deployment approvals, 251–252
 - gated check-ins, 257–259
 - load tests with VSTS, 259–261
 - overview, 240–241
 - release definition, 249–251
 - software testability, 252–256
 - unit testing, 256–257
- CD software testability
 - clarity, 255
 - controllability, 253
 - isolability, 254–255
 - observability, 253–254
 - overview, 252–253
- CDN (Content Delivery Network), scaling with, 157–158, 329, 480
- certificates
 - using for client authentication, 178
 - using to protect clusters, 174–177
- chaos testing, reliability, 136–141
- CI (continuous integration)
 - build definition, 246–249
 - team preparation, 242–246
 - TFS (Team Foundation Server), 242
 - TFS (Team Foundation Service), 242
 - VSO (Visual Studio Online), 242
 - VSTS (Visual Studio Team Services), 242
 - VSTS nomenclature, 242
- claim-based authentication, scaling with partition, 335, 481
- CLI (command-line interface), 25, 195–197
- client authentication
 - AD (Active Directory), 178–180
 - certificates, 174–178
 - Visual Studio, 180–181
- client-side latency, 345–346
- Cloud Shell, 26, 173
- cluster configuration
 - internal load balancer, 211
 - NSGs (network security groups), 207–211
 - RBAC (role-based access control), 206–207
- cluster management commands. *See also* scripting
 - application-level queries, 186–187
 - cross queries on cluster nodes, 189–190
 - node operations, 190–192
 - node-level queries, 184–185
 - partition-level queries, 188
 - query commands, 181–184
 - replica-level queries, 188
 - service-level queries, 187–188
- cluster settings, updating, 212
- cluster-level queries, 181–184
- clusters. *See also* local cluster; nodes; PowerShell; provisioning clusters; VMs (Virtual Machines)
 - autoscaling, 154–157
 - blades, 199
 - load balancers, 203–205
 - managing, 196–197
 - map in Service Fabric Explorer, 201
 - NAT rules, 203–204
 - protecting, 174–177
 - provisioning, 11–15
 - resources, 200
 - scaling, 151–158
 - Service Fabric on Edge, 463–466
 - storage accounts, 206
 - virtual networks, 202
 - VM scale sets, 200
- code package, 36–37, 93–95
- Cognitive Services, 418
- Command Query Responsibility Segregation (CQRS), 348–349
- communication bottlenecks, 164–165
- communication stacks. *See also* stateless services
 - customizing, 45–48
 - default, 32–41
 - WCF, 41–44
- communication subsystem, 7
- competing consumers, 167, 479
- composable processing, explained, 479
- compute to data
 - moving, 462
 - Service Fabric on Edge, 463–466
 - workload distribution, 466–467

ConfigPackage

- ConfigPackage, 37
- configuration and data changes, availability, 131–133
- configuration package, 95–96
- container orchestration
 - DNS service, 301–306
 - Docker Compose, 309–313
 - service meshes, 314–322
 - watchdog service, 306–309
- containerization, 3–4
- containers. *See* ACI (Azure Container Instances); Docker
 - and guest applications, 10
 - high availability, 290
 - patterns, 479
 - running as daemons, 283
 - Ubuntu, 282–283
 - Windows, 281–282
- control plane, 339–342, 383, 481
- Cosmos DB, 371–372
- coupling, 164
- CPU consumption limits, 115
- CQRS and event sourcing, 348–349, 480
- create/replace/upgrade, 97
- cross queries on cluster nodes, 189–190

D

- daemons, running containers as, 283
- dashboard, defined, 11
- data authenticity, 174–177
- data integrity, 174–177
- data privacy, 174–177
- data replication, Microsoft Azure Service Fabric, 5
- data storage
 - overview, 367
 - relational databases, 368–371
- databases, 368–371
- DataPackage, 37
- data-streaming pipelines
 - composable processing, 350–351
 - processing sequence, 351–355
 - test client, 355–357
- dead-letter channel, 383–384
- DefaultServices, 37
- defragmentation, service placements, 125–126
- development environment
 - Linux, 265–266
 - setting up, 10–11

- device actors, static compute, 441–443
- diagnostic data, collecting, 134–136
- diagnostics
 - configuring, 215–218
 - data pipeline, 213–214
 - Elasticsearch, 221–225
 - EventFlow, 218–225
 - Kibana, 221–225
- diff packages, upgrading with, 133
- disk encryption, 373
- distributed computing, 169, 479–480
- DMZs (demilitarized zones), 208, 481
- DNS and container orchestration
 - application, 304–306
 - deployment and testing, 308–309
 - Docker images, 302–304
 - overview, 301
 - Spring Java server, 307–308
 - watchdog service, 306–308
- DNS service, availability, 129–130
- Docker. *See also* containers
 - container ecosystem, 387
 - containerization on Linux, 279–281
 - running on Azure, 286
 - running on Linux, 282–285
 - running on Windows, 286
- Docker and Service Fabric
 - ASP.NET Core on Windows, 287–289
 - Minecraft container on Linux, 290–296
 - using, 286
- Docker Compose. *See also* services
 - composing services, 312
 - deployment and testing, 312–313
 - master image, 310–311
 - overview, 309–310
 - slave image, 311
- Docker images, working with, 284–285
- Docker volume drivers, Minecraft server
 - container, 283–284
- documentation, Microsoft Azure Service Fabric, 26
- DSC (desired state configuration), Linux, 263–264
- durability characteristics, 149

E

- e-commerce application, 361
- Edge Envoy, 320–322
- elastic pools, 369–370

- elastic systems, 406
- Elasticsearch, 221–225
- encrypting disks, 373
- entities
 - managing with `sfctl`, 197
 - query commands, 181–190
- Envoy edge and service proxy
 - application, 319–320
 - deploying, 316–322
 - Edge Envoy, 320–322
 - Go container, 317–319
 - overview, 314–315
 - Postgres container, 316–317
- ETW (Event Tracking for Windows), 18
 - actors, 81–83
 - event source providers, 135
 - reliability, 134
- Event Grid architecture, 398–399
- event ingress, static compute, 449–452
- event sourcing, 348–350
- event storage, static compute, 452–460
- EventFlow, 218–225

F

- FabricCounters performance counter, 83–84
- failover tests, performing, 140–141
- fault domain, 101
- federation subsystem, 6–7
- field-gateway simulation, 448–461
- functions, creating, 399–401

G

- Go container, Envoy edge and service proxy, 317–319
- gold durability tier, 149
- gold reliability tier, 150
- graceful faults, 141
- gRPC framework, 45–48
- guest applications, 10
- guest binary services, Linux, 276–277

H

- health model
 - overview, 97–99
 - policy, 99–100

- query commands, 182
 - reporting and aggregation, 101
 - states, 99
- Hello, World!
 - Linux, 267–270
 - Microsoft Azure Service Fabric, 15–19
- heterogeneous instances, 146–147
- homogeneous instances, 146–147
- horizontal scaling, 3, 145–146
- hosting subsystem, 7
- HTTP and HTTPS
 - cluster protection, 177
 - endpoints, 110–111
- hub, defined, 11
- Hyper-V containers, Windows, 282

I

- image-analysis applications, 421–422
- implicit hosts
 - defining, 108–109
 - Node.js application, 111–114
 - RunAs policies, 109–111
- instances, homogeneous vs. heterogeneous, 146–147
- interactive system techniques
 - latency, 343–346
 - throughput, 347–348
- internal load balancer, 211
- IoT pipeline, 437, 449, 459, 462
- isolation levels, 160

J

- Java SDK vs. .NET SDK, 274–276
- Java Spring server, building, 307–308
- Jenkins server, continuous deployment, 297–298
- join-by-field topology, 359

K

- Key Vault security, 377–378
- Kibana, 221–225
- Kubernetes, 326, 328
- KVS (key-value store), 88

L

latency

- client-side, 345–346
- server-side, 343–345

Linux

- actor services, 275–276
 - cgroups, 279–280
 - container runtimes, 281
 - containerization with Docker, 279–281
 - and containers, 263–264
 - copy-on-write, 279
 - development environment, 265–266
 - Docker images, 281
 - DSC (desired state configuration), 263–264
 - guest binary services, 276–277
 - Hello, World! 267–270
 - namespaces, 279
 - running Docker, 282–285
 - single-container networking, 285
 - stateful services, 274–275
 - Yeoman application scaffolding, 277–278
- ### listeners
- CommunicationListener, 270–273
 - support, 48
 - WebSocket for live data processing, 364–365
- ### load balancers, 31, 203–205. *See also* resource load balancing
- ### local cluster, 19–26, managing. *See also* clusters; provisioning clusters
- ### local store settings, 90
- ### log analytics, 226–228, 373
- ### logger, stateful services, 51
- ### Logic Apps, 401–402
- ### loose coupling, 164–165, 481

M

- ### Machine Learning Studio. *See also* AI (artificial intelligence)
- calling services, 417–418
 - Cognitive Services, 418
 - natural language processing, 422–428
 - OCR (optical character recognition), 418–422
 - overview, 413–417
- ### management subsystem, 7
- ### memory consumption limits, 115
- ### message delivery, failure, 383–384
- ### message translators, 386

- ### message-driven systems, 403
- ### messaging gateway, 384–385
- ### Metaparticle library, PaaS ecosystem, 328
- ### metric triggers, 155
- ### microservices, 3–6, 299–300
- ### Microsoft Azure Service Fabric
- applications and services, 8
 - architecture, 6–7
 - CLI (command-line interface), 25
 - Cloud Shell, 26
 - containerization, 3–4
 - containers, 10
 - data replication, 5
 - development environment, 10–11
 - documentation, 26
 - evolution, 6
 - guest applications, 10
 - Hello, World! 15–19
 - microservices, 3–6
 - nodes and clusters, 7–8
 - partitions and replicas, 8–9
 - programming modes, 9
 - provisioning cluster, 11–15
 - scheduling, 4
 - SDK, 10
 - Service Fabric Explorer, 21–24
 - service partitioning, 5
 - state reconciliation, 4
 - stateless vs. stateful, 9–10
 - subscription, 11
 - updates, 26
 - Visual Studio Cloud Explorer, 19–21
 - Windows PowerShell, 24–25
- ### Minecraft server container on Linux
- application, 291–292
 - Azure storage account, 292–293
 - deploying, 294
 - Docker volume drivers, 283–284
 - redundancy, 290–291
 - scaling, 296
 - testing, 295
 - testing failovers, 296
- ### mock components, 254–255, 481
- ### monitoring
- Application Insights, 232–237
 - Explorer, 230–232
- ### Monte Carlo simulation, 169
- ### multi-tenancy
- patterns, 480
 - vs. single tenancy, 147–148

N

N elements, streaming, 358–359
 NamedPartition, 62
 Naming Service, 37–41, 127–128
 NAT (network address translation) rules, 203–204
 natural language processing, 422–428
 .NET SDK vs. Java SDK, 274–276
 neural networks, 411
 Nginx server, launching as daemon, 283
 Node.js application, hosting, 111–114
 node-level queries, 184–185
 nodes. *See also* clusters
 and clusters, 7–8
 operations, 190–192
 troubleshooting, 229
 types, 148–150
 noisy neighbor problem, 115
 NoSQL databases, 371–372
 NSGs (network security groups), 207–211

O

OAuth, AD (Active Directory), 374–375
 OCR (optical character recognition), 418–422
 OMS (Operations Management Suite), 225–229
 OOP (object-oriented programming), 67
 orchestration
 bottlenecks, 165–168
 engines, 299–301
 explained, 4
 OSB (Open Service Broker) API, 380–381

P

PaaS ecosystem
 AKS (Azure Container Service), 326
 app services, 325
 Metaparticle library, 328
 platforms, 327–328
 Service Fabric, 326–327
 VMSS (Virtual Machine Scale Sets), 326
 parallel batching topology, 358
 partition-level queries, 188
 partitions, 5, 8–9
 patterns
 containers, 479
 distributed computing, 479–480
 multi-tenancy, 480

 performance and scalability, 480
 system architecture, 481
 Paxos algorithm, 66
 performance
 counters and actors, 83–84
 patterns, 480
 Performance Monitor, 85, 90
 persistent data volumes, 479
 pi, estimating, 169
 placement constraints, availability, 122–123
 platinum reliability tier, 150
 player actor, implementing, 74–75
 port sharing, scaling with partition, 337–338
 portal terms, 11
 Postgres container, Envoy edge and service
 proxy, 316–317
 Postman UI, 223
 PowerShell. *See also* clusters
 clusters, 174–181
 Create command, 475
 Delete command, 475
 installation, 473
 Key Vault security, 377–378
 Read command, 475
 resource groups, 475–478
 signing in, 473–474
 Update command, 475
 probe properties, updating, 205
 programming modes, Microsoft Azure
 Service Fabric, 9
 provisioning clusters, 11–15. *See also* clusters;
 local cluster
 Python application, guest binary service, 276–277

Q

QoS (quality of service) improvements, 3
 query commands, 181–184

R

RBAC (role-based access control), 206–207
 reactive systems, 403–406, 481
 ReadOnly method, using with actor timers, 79
 reads, separating from writes, 348–350
 redundancy and high availability, 290
 register/provision, 97
 relational databases, 368–371

reliability

reliability

- Azure Diagnostics, 134–135
 - chaos testing, 136–141
 - characteristics, 150
 - defined, 133
 - ensuring, 133–134
 - ETW (Event Tracking for Windows), 134
 - graceful faults, 141
 - improving, 118–119
 - restoring services, 143–144
 - state backup and restore, 141–144
 - subsystem, 6–7
 - ungraceful faults, 141
- Reliable State Manager, 50–51
- remote accesses, 44
- replica roles, Simple Store application, 63–64
- replica-level queries, 188
- replicas. *See also* partitions; services
- availability, 119
 - defined, 21
- replicator behavior settings, 89
- resilient systems, 406
- resource governance, 115
- resource load balancing, 64–65, 120–122. *See also* load-balancers
- resource reserves, availability, 126
- resources, defined, 151
- response time and throughput, 159
- responsive systems, 403–405
- reverse proxy
- availability, 128–129
 - scaling with partition, 336
- RM (Resource Manager), 475
- rolling upgrades, 131–133. *See also* upgrades
- routing and load-balancing, 127–130
- RunAs policies, 109–111

S

scalability

- ACI (Azure Container Instances), 338–339
- and availability, 32
- durability characteristics, 149
- dynamic workload patterns, 407
- extensible control plane, 339–342
- heterogeneous instances, 146–147
- homogeneous instances, 146–147
- manual scaling vs. autoscaling, 148

- node types, 148–149
 - patterns, 480
 - reliability characteristics, 150
 - single tenancy vs. multi-tenancy, 147–148
 - stateless vs. stateful, 146
 - vertical vs. horizontal, 145–146
- scalable interactive systems
- data-streaming pipelines, 350–357
 - latency, 343–346
 - throughput, 347–348
 - topologies and actors, 357–360, 479
 - WebSocket for live data processing, 360–366
- scaling,
- vs. autoscaling, 148
 - by partition, PaaS, 326
- scaling clusters
- ARM (Azure Resource Manager), 151–152
 - autoscaling, 154–157
 - AZM (Azure Management Portal), 152–154
 - CDN (Content Delivery Network), 157–158
 - on manual basis, 152–154
 - VM (Virtual Machine) Scale Sets, 151–152
- scaling with bursting, overview, 338–339, 479
- scaling with partition
- claim-based authentication, 335
 - explained, 480
 - port sharing, 337–338
 - reverse proxy, 336
 - service meshes, 335–338
 - tenant manager, 332–335
- scaling with reduction
- aggregated views, 330
 - caching, 330–331
 - CDN (Content Delivery Network), 329
 - data manipulation, 331
 - explained, 480
 - home views, 329–330
 - overview, 328
 - precomputed views, 331
 - recommendation views, 330
- scheduling, 4
- scripting. *See also* cluster management commands
- application-management commands, 192–197
 - Cloud Shell, 173
 - PowerShell, 174–181
- Sea Breeze
- design principles, 407–408
 - features, 395

- security
 - AD (Active Directory), 373–377
 - disk encryption, 373
 - Key Vault, 377–378
 - log analytics, 373
 - SSL with custom domain, 378–379
- sensor actors, 443, 481
- serverless computing
 - architecture, 395–396
 - benefits, 396–397
 - container instances, 397–398
 - deployment, 393–394
 - elastic systems, 406
 - Event Grid, 398–399
 - explained, 481
 - Functions, 399–401
 - Logic Apps, 401–402
 - message-driven systems, 403
 - platform, 394–395
 - resilient systems, 406
 - responsive systems, 403–405
 - Sea Breeze, 395
- server-side latency, 343–345
- service affinity, 124–125, 480
- service brokers
 - explained, 481
 - extensible control plane, 383
 - integration, 379–383
 - OSB (Open Service Broker) API, 380–381
 - OSB for Azure, 381
- service defragmentation, availability, 125–126
- Service Fabric
 - applications and services, 8
 - architecture, 6–7
 - CLI (command-line interface), 25–26
 - Cloud Shell, 26
 - containerization, 3–4
 - containers, 10
 - data replication, 5
 - development environment, 10–11
 - documentation, 26
 - evolution, 6
 - guest applications, 10
 - Hello, World! 15–19
 - microservices, 3–6
 - nodes and clusters, 7–8
 - partitions and replicas, 8–9
 - programming modes, 9
 - provisioning cluster, 11–15
 - scheduling, 4
 - SDK, 10
 - Service Fabric Explorer, 21–24
 - service partitioning, 5
 - state reconciliation, 4
 - stateless vs. stateful, 9–10
 - subscription, 11
 - updates, 26
 - Visual Studio Cloud Explorer, 19–21
 - Windows PowerShell, 24–25
- Service Fabric Explorer, Microsoft Azure Service Fabric, 21–24
- Service Fabric on Edge, 463–466
- service failovers, availability, 126–127
- service manifest, 36–37
- service meshes
 - Envoy, 314–322
 - explained, 479
 - scaling with partition, 335–338
- service partitioning, 5, 61–63
- service placements
 - affinity, 124–125
 - availability, 120–126
 - batching load reports, 122
 - constraints, 122–123
 - defragmentation, 125–126
 - resource reserves, 126
 - resource-balancing metrics, 120–122
- service-level queries, 187–188
- services. *See also* broken services; Docker Compose; replicas
 - APIs, 9
 - composing services, 387–389
 - discovery process, 38
 - parts, 35–36
 - service catalog, 381–383
- ServiceTypes, 36
- SetupAdminUser account, 110
- sfctl command line, 197
- SFNuGet project, 387–389
- shopping-cart service, Simple Store
 - application, 52–56
- silver durability tier, 149
- silver reliability tier, 150
- Simple Store application. *See also* stateful services
 - replica roles, 63–64
 - resource load balancing, 64–65
 - service partition, 61–63
 - shopping-cart service, 52–56
 - website, 56–61

single tenancy vs. multi-tenancy

- single tenancy vs. multi-tenancy, 147–148
- Spring Java server, building, 307–308
- SQL Databases
 - event storage, 452–454
 - provisioning, 368–369
 - word-statistics sample, 370–371
- SQL elastic pools, 369–370
- SSL, enabling with custom domain, 378–379
- SSL with custom domain, 378–379
- state backup and restore, reliability, 141–144
- state bottlenecks
 - actor states, 161–162
 - explained, 159
 - isolation levels, 160
 - serialization, 162–163
- state management performance, 85–86
- state reconciliation, Microsoft Azure Service Fabric, 4
- state replicators, 88
- stateful services. *See also* Simple Store application
 - collections, 50
 - consistency, 52
 - Linux, 274–275
 - logger, 51
 - PaaS ecosystem, 326
 - Reliable State Manager, 50–51
 - Transactional Replicator, 51
- stateless services. *See also* communication stacks
 - ASP.NET core applications, 27–31
 - scalability and availability, 31–32
 - vs. stateful, 9–10, 146
- static compute
 - aggregators, 443–448
 - command and control, 438–439
 - data generation and feedback, 438
 - data ingress, 439
 - data transformation and analysis, 439–440
 - device actors, 441–443
 - end-to-end scenario, 448–449
 - event ingress, 449–452
 - event storage, 452–460
 - IoT pipeline, 437
 - monitoring sensor data, 460
 - presentation and actions, 441
 - storage, 440
- storage, static compute, 440
- storage accounts, 206, 292–293
- subscription, obtaining, 11
- subsystems, 6–7

- system architecture, patterns, 480
- system integration
 - composing services, 387–389
 - data storage, 367–372
 - patterns with messaging, 383–386
 - security, 373–379
 - service brokers, 379–383

T

- tenancy, single vs. multi, 147–148
- Tenant Manager
 - explained, 480
 - scaling with partition, 332–335
- TensorFlow compute engine, 432–435
- testability subsystem, 7
- TFS (Team Foundation Server), CI (continuous integration), 242
- throughput and response time, 159
- tic-tac-toe game. *See also* actors
 - actor interface, 71
 - actor models, 70
 - creating, 71
 - game actor, 72–74
 - improving, 77
 - player actor, 74–75
 - test client, 75–76
 - testing, 77
- tile, defined, 11
- timers. *See* actors
- top N processing topology, 358–359
- transaction coordinator, 385–386
- Transactional Replicator, stateful services, 51
- transport subsystem, 6–7
- troubleshooting, nodes, 229

U

- Ubuntu container, using, 282–283
- ungraceful faults, 141
- UniformInt64Partition, 61–62
- updates
 - domain, 101
 - Microsoft Azure Service Fabric, 26
 - probe properties, 205
- upgrades, 101–107. *See also* rolling upgrades

upgrading
 applications, 193–194
 with diff packages, 133

V

ValidationFailedEvent, raising, 137
 vertical scaling, 145–146
 virtual networks, 202
 Visual Studio
 Cloud Explorer, 19–21
 development environment, 10–11
 publishing applications to, 180–181
 VMs (Virtual Machines). *See also* clusters
 durability characteristics, 149
 scale sets, 200
 and virtual network cards, 201–202
 VMSS (Virtual Machine Scale Sets), PaaS
 ecosystem, 326
 VSO (Visual Studio Online), CI (continuous
 integration), 242
 VSTS (Visual Studio Team Services)
 load tests, 259–261
 nomenclature, 242
 project preparation, 242–246
 VSTS (Visual Studio Team Services), CI (continuous
 integration), 242

W

watchdog service
 building, 306–307
 explained, 480
 WCF communication stack, 41–44
 Web Platform Installer, 10–11
 website, Simple Store application, 56–61
 websites
 Azure AD tenants, 179
 Azure CLI (command-line interface), 195
 Azure Quick Start templates, 151–152

CDN (Content Delivery Network), scaling with, 158
 cluster-management commands, 191
 documentation, 26
 Elasticsearch, 221
 Hello, World! 15
 Paxos algorithm, 66
 Performance Monitor, 90
 Service Fabric SDK, 10
 sfctl command line, 197
 WCF communication stack, 41
 Yeoman application scaffolding, 277–278
 WebSocket for live data processing
 country/region actor, 362–363
 gateway, 364
 global actor, 363
 listener, 364–365
 overview, 360–361
 product actor, 361–362
 test client, 365–366
 Windows
 ASP.NET core container, 287–289
 containers, 281–282
 Hyper-V containers, 282
 Performance Monitor, 85, 90
 PowerShell, 24–25
 running Docker, 286
 word statistics, saving, 370
 workflows, coordination, 167, 480
 workload auction pattern, 479
 writes, separating from reads, 348–350
 ws-Federation, AD (Active Directory), 373–374

Y

Yeoman application scaffolding, Linux, 277–278

Z

zero-downtime upgrade, 101–102