Microsoft

Programming Microsoft Azure Service Fabric

Professional

Haishi Bai

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Microsoft

Programming Microsoft Azure Service Fabric

Haishi Bai

PUBLISHED BY
Microsoft Press
A division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2015953291
ISBN: 978-1-5093-0188-1

Printed and bound in the United States of America.

First Printing

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Support at mspinput@microsoft.com. Please tell us what you think of this book at http://aka.ms/tellpress.

Acquisitions Editor: Karen Szall
Developmental Editor: Karen Szall
Editorial Production: Cohesion
Technical Reviewer: John McCabe; Technical Review services provided by Content Master, a member of CM Group, Ltd.
Copyeditor: Ann Weaver
Indexer: Lucie Haskins
Cover: Twist Creative • Seattle

Contents at a glance

Table of contents

Introduction

Azure Service Fabric is Microsoft's new platform as a service (PaaS) offering for developers to build and host available and scalable distributed systems. Microsoft has used Service Fabric internally 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 DocumentDB. The same platform is now available to you to write your own highly available and highly scalable services.

Programming Microsoft Azure Service Fabric is designed to get you started and productive with Azure Service Fabric quickly. The book covers fundamentals, practical architectures, and design patterns for various scenarios such as the Internet of Things (IoT), big data, and distributed computing. For fundamentals, the book provides detailed step-by-step walkthroughs that guide you through typical DevOps tasks. For design patterns, the 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, the book briefly covers other Azure services that are 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 to use Azure Service Fabric and related Azure services in their application architecture.

Most of the book was written while the service was still in preview. So, this book is most suitable for readers who want to keep on the edge of Azure development. As one of the earliest Service Fabric books on the market, this book provides some early insights into a new service that is still under active development. 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 lifecycle management (ALM), IoT, big data, and big compute will help you understand these chapters.

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 pros.

Organization of this book

This book is divided into four sections, 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 lifecycle 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, "Patterns and scenarios," introduces practical design patterns and best practices in implementing typical application scenarios including scalable web applications, IoT, big data, multitenant applications, and gaming. Finally, Part IV, "Advanced topics," covers two advanced topics: advanced service hosting and modeling complex systems using Service Fabric.

Finding your best starting point in this book

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

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 13 and Chapter 14.
Interested in building scalable web applications	Focus on Chapters 12, 14, and 15. You may also want to skim through other chapters in Part III to discover some patterns that may be applicable to your scenarios.
Interested in gaming	Focus on Chapter 16. Also read Chapter 14, especially the section about the Web Socket communication stack, which provides satisfactory performance in many web-based multiplayer gaming scenarios.
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 application lifecycle management topics.
Interested in the Actor programming model	Focus on Chapter 4. Also browse through chapters in Part III because these chapters cover a number of Actor-based design patterns.
Interested in Service Fabric container integration	Focus on Chapter 17. Appendix C also gives you great background information on container integrations.
Interested in modeling complex systems with Service Fabric	Focus on Chapter 18.

Some of the book's chapters include hands-on samples that let you try out the concepts just learned. No matter which sections you choose to focus on, be sure to download and install the sample applications on your system.

System requirements

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

- Windows 7, Windows 8/Windows 8.1, Windows Server 2012 R2, or Windows 10.
- Visual Studio 2015.
- Latest Service Fabric runtime, SDK, and tools for Visual Studio 2015 (install via Web PI).
- Latest version of Azure SDK (2.8 or above, install via Web PI).
- Latest version of Azure PowerShell (1.0 or above, install via Web PI).

- Latest version of Azure CLI.
- 4 GB (64-bit) RAM.
- 30 GB of available hard disk space.
- An active Microsoft Azure subscription. You can get a free trial from *www.azure.com*.
- Internet connection to use Azure and to download software or chapter examples.

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

Downloads: Code samples

Most of the chapters in this book include sample code that lets you interactively try out new material learned in the main text. All sample projects can be found on the book's download webpage:

http://aka.ms/asf/downloads

Using the code samples

The book's webpage contains all samples in this book, organized in corresponding chapter folders. It also contains two additional folders: ComplexSystems and AdditionalSamples. The AdditionalSamples folder contains additional sample scenarios. The ComplexSystems folder contains frameworks and sample scenarios of complex systems.

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

  *HelloWordApplicatio*n: The hello world application.

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

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

- **Chapter 3** This folder contains samples from Chapter 3.

  *SimpleStoreApplicatio*n: The simple store application.

  SimpleStoreApplication-NamedPartitions: The simple store application using named partitions.

- **Chapter 4** This folder contains samples from Chapter 4.

  ActorTicTacToeApplication: The tic-tac-toe game using Actors.

- **Chapter 5** This folder contains samples from Chapter 5.

  C*onsoleRedirectTestApplication*: The sample application used in package format samples.

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

  SensorAggregationApplication-Pull: The IoT scenario that aggregates sensor states using pull mode.

  SensorAggregationApplication-Push: The IoT scenario that aggregates sensor states using push mode.

  IoTE2E End-to-end: The IoT sample scenario.

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

  *NumberConverter*App: The number converter service.

  *ECommerceApplicatio*n: The sample e-commerce application.

- **Chapter 15** This folder contains samples from Chapter 15.

  MetadataDrivenApplication: The sample metadata-driven application (shows actor polymorphism).

  ThrottlingActorApplication: The sample application shows the Throttling Actor pattern.

- **Chapter 16** This folder contains samples from Chapter 16. Both scenarios are under development. Please see release announcements in the repository for releasable versions.

  MessyChess: The Messy Chess sample (under development).

  AIQuest: The A.I. Quest sample (under development).

- **Chapter 17** This folder contains samples from Chapter 17.

  *GuestApplicatio*n: A simple guest application sample with a watchdog.

- **Chapter 18** This folder contains samples from Chapter 18.

  TermiteModel: A simulation of termites moving and collecting wood chips.

ActorSwarmApplication: A simulation of people moving closer to neighbors with similar attributes. This sample shows a preliminary implementation of an actor swarm.

AdditionalSamples: This folder contains additional sample scenarios (see README.md under the folder).

ComplexSystems: This folder contains frameworks and samples for modeling complex systems.

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 .csproj as the file extension.

Acknowledgments

I'd like to thank my wonderful editor Karen Szall who has guided me through every single step along the way to get this book published. I'd also like to thank John McCabe for his insightful reviews. Especially, I'd like to thank Boris Scholl who, regardless of his busy schedule to get the service released on time, has helped me tremendously reviewing the book and providing me insights into container integrations.

I'd also like to thank the amazing team behind Service Fabric. I've personally worked with many of the team members including Mark Fussell, Matthew Snider, Vaclav Turecek, and Sean McKenna. The creativity and dedication of the team has inspired me while writing this book. I also gained a lot of knowledge from the Microsoft internal community, including the *www.azure.com* author group and the Yammer group.

Last but not least, I'd like to thank my wife Jing and my daughter Sabrina who have tolerated my late hours and busy weekends in the past six months. I couldn't do this without your support.

Free ebooks from Microsoft Press

From technical overviews to in-depth information on special topics, the free ebooks from Microsoft Press cover a wide range of topics. These ebooks are available in PDF, EPUB, and Mobi for Kindle formats, ready for you to download at:

http://aka.ms/mspressfree

Check back often to see what is new!

Quick access to online references

Throughout this book are addresses to webpages that the author has recommended you visit for more information. Some of these addresses (also known as URLs) can be painstaking to type into a web browser, so we've compiled all of them into a single list that readers of the print edition can refer to while they read.

The list is included in the companion content, which you can download here:

http://aka.ms/asf/downloads.

The URLs are organized by chapter and heading. Every time you come across a URL in the book, find the hyperlink in the list to go directly to the webpage.

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:

http://aka.ms/asf/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 *http://support.microsoft.com*.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://aka.ms/tellpress

We know you're busy, so we've kept it short with just a few questions. Your answers go directly to the editors at Microsoft Press. (No personal information will be requested.) Thanks in advance for your input!

Stay in touch

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

PART I

Fundamentals

This page intentionally left blank

CHAPTER 1

Hello, Service Fabric!

This book is about Microsoft Azure Service Fabric, a distributed systems platform that makes it easy to build large-scale, highly available, low-latency, and easily manageable services. Service Fabric brings you the same technology that empowers cloud-scale applications such as Cortana, Skype for Business, and SQL databases so that you can easily design, implement, scale, and manage your own services leveraging the power of next-generation distributed computing.

Before you embark on the journey with Service Fabric, let's reflect on what makes a great platform as a service (PaaS) and why you need a new PaaS to build the next generation of cloud-based services.

A modern PaaS

A PaaS is designed with agility, scalability, availability, and performance in mind. Microsoft Azure Service Fabric is a PaaS that is built from the ground up to support large-scale, highly available cloud applications.

Designed for agility

The software industry is all about agility. Developers have the privilege to work in a virtual world without physical constraints to drag us down. Innovations can happen at a speed that is unimaginable to other fields. And as a group, we've been in a relentless pursuit for speed: from software frameworks to automation tools, from incremental development to Heroku's 12-factor methodology (Wiggins 2012), from minimum viable product (MVP) to continuous delivery. Agility is the primary goal so that developers can innovate and improve continuously.

Microservices

The essence of Microservices is to decompose complex applications into independent services. Each service is a self-contained, complete functional unit that can be evolved or even reconstructed without necessarily impacting other services.

All software can be abstracted as components and communication routes between them. Monolithic applications are hard to maintain or revise. An overly decomposed system, in contrast, is hard to understand and often comes with unnecessary overhead due to the complex interaction paths

across different components. To a great extent, the art of a software architect is to strike a balance between the number of components and the number of communication paths.

A PaaS designed for Microservices encourages separation of concerns, emphasizes loose coupling, and facilitates flexible inter-component communication. While allowing other architectural choices, Service Fabric is designed for and recommends Microservices. A Service Fabric application is made up of a number of services. Each service can be revised, scaled, and managed as an independent component, and you still can manage the entire application as a complete logical unit. The Service Fabric application design is discussed in Chapter 2, "Stateless services," Chapter 3, "Stateful services," and Chapter 4, "Actor pattern." We'll review several application patterns and scenarios in Part III.

Note Architecture choices

Microservices is strongly recommended but not mandatory. You can choose to use other architectures such as n-tiered architecture, data-centric architecture, and single-tiered web applications or APIs.

Simplicity

A PaaS platform is not just about scheduling resources and hosting applications. It needs to provide practical support to developers to complete the tasks at hand without jumping through hoops.

At a basic level, a PaaS platform helps developers deal with cross-cutting concerns such as logging, monitoring, and transaction processing. Taking it a step further, a PaaS platform provides advanced nonfunctional features such as service discovery, failover, replication, and load balancing. All these nonfunctional requirements are essential to a scalable and available system. And providing built-in constructs to satisfy these requirements leads to a significant productivity boost. Because PaaS takes care of all these troubles, developers can focus on building up core business logic. To achieve this, these nonfunctional features should be available without getting in the way. As you progress through this chapter and book, you'll see how Service Fabric enables you to focus on business logic and to incorporate these features whenever you need them.

Can you go a step further? What if a PaaS platform provides easy programming models that help you tackle complex problems? And what if the PaaS platform also provides guidance and patterns for typical scenarios? We'll come back to this in the discussion of different service types in Chapter 2, Chapter 3, and Chapter 4.

Comprehensive application lifetime management

Continuous improvement is at the core of the agile software movement and the Lean movement in various industries. The faster you can iterate through revision cycles, the quicker you can innovate, reduce waste, and create additional value. A mature PaaS platform has to offer comprehensive application lifecycle management (ALM) functionalities to keep the innovation engine running without friction.

Because more companies are adopting continuous delivery, software is being released at a faster pace than in the past. Some companies claim they do hundreds of deployments on a daily basis. This calls for automated testing, continuous integration, rapid deployments, robust version management, and fast rollbacks. Only when a PaaS platform provides all these features can developers and independent software vendors (ISVs) realize such continuous delivery scenarios.

A comprehensive ALM strategy is critical to DevOps. If you look carefully, you'll see that a lot of so-called friction between development and operations is rooted in discrepancies among different environments. PaaS platforms such as Service Fabric allow applications to be placed in self-contained packages that can be deployed consistently to different environments—such as development, test, QA, and production.

Part II of this book is dedicated to ALM.

Designed for QoS

A successful cloud service is based on a healthy partnership between the service developer and the cloud platform. The service developer brings business know-how and innovation, and the cloud platform brings Quality of Service (QoS) opportunities such as scalability, availability, and reliability.

Scalability

Through innovation, you can do unprecedented things. However, the increasing complexity of problems constantly challenges developers to improve methodologies to maintain momentum. A PaaS platform should be designed with scalability in mind so that applications can be scaled out naturally without much effort from the developers.

Increasing complexity and scale

Increasing complexity can be demonstrated easily with some examples. According to the "NASA Study on Flight Software Complexity" (NASA Office of Chief Engineer, 2009), flight software complexity has been increasing exponentially with a growth rate of a factor of 10 approximately every 10 years. *Apollo 8* had about 8,500 lines of code in 1968. In contrast, the International Space Station (ISS) was launched with 1.5 million lines of code in 1989.

Besides software complexity, the sheer volume of data presents a new set of problems. According to Twitter statistics (Company Facts at *https://about.twitter.com/company*), Twitter is handling 500 million tweets every day. Data ingress, transformation, storage, and analysis at such a scale is an unprecedented challenge. Modern services also need to deal with the potential for rapid growth. Over the past five years or so, Azure Storage has grown into a service that needs to handle 777 trillion transactions per day (Charles Babcock, "Microsoft Azure: More Mature Cloud Platform," InformationWeek, Sept 30, 2015, *http://aka.ms/asf/maturecloud*).

On a cloud platform, *scaling up*, which means increasing the processing power of a single host, is a less preferable approach. Typically, virtual machines are offered with preconfigured sizes. To scale up, you'll need to migrate your workload to a virtual machine with a bigger size. This is a long and disruptive process because services need to be brought down, migrated, and relaunched on the new machine, causing service interruptions. Furthermore, because there are finite choices of machine sizes, scaling options run out quickly. Although Azure provides a large catalog of virtual machine sizes, including some of the largest virtual machines in the cloud, large-scale workloads still can exceed the processing power of a single machine.

In contrast, *scaling out* dynamically adjusts system capacity by adding more service instances to share the workload. This kind of scaling is not disruptive because it doesn't need to shut down existing services. And theoretically, there's no limit to how much you can scale because you can add as many instances as you need.

When scaling out, there are two fundamental ways to distribute workloads. One way is to distribute the workloads evenly across all available instances. The other way is to partition the workloads among service instances. Service Fabric supports both options, which we'll discuss in detail in Chapter 7, "Scalability and performance."

Availability

Availability commonly is achieved by redundancy—when a service fails, a backup service takes over to maintain business continuity. Although the idea sounds simple, difficulties can be found in the details. For example, when a service fails, what happens to its state that it has been maintaining locally? How do you ensure that the replacement service can restore the state and pick up wherever it left off? In a different case, when you apply updates, how do you perform a zero-downtime upgrade? And how do you safely roll back to previous versions if the new version turns out to be broken? The solution to these questions involves many parts such as health monitoring, fault detection, failover, version management, and state replication. Only a carefully designed PaaS can orchestrate these features into a complete and intuitive availability solution. Reliability and availability is the topic of Chapter 6, "Availability and reliability."

Reliability

Reliability is compromised by system faults. However, in a large-scale, distributed system, monitoring, tracing, and diagnosing problems often are challenging. If a PaaS doesn't have a robust health subsystem that can monitor, report, and react to possible system-level and application-level problems, detecting and fixing system defects becomes incredibly difficult.

We'll examine what Service Fabric has to offer in terms of reliability in Chapter 6.

Separation of workload and infrastructure

The cloud era brings new opportunities and new challenges. One advantage of cloud infrastructure as a service (IaaS) is that it shields you from the complexity of physical or virtualized hardware management—and that's only the starting point. To enjoy the benefits of the cloud fully, you need PaaS to

help you forget about infrastructure altogether. After all, for a program to run, all you need are some compute and storage resources such as CPU, memory, and disk space. Do you really need to control which host is providing these resources? Does it really matter if your program stays on the same host throughout its lifetime? Should it make a difference if the program is running on a local server or in the cloud? A modern PaaS such as Service Fabric provides a clear separation of workload and infrastructure. It automatically manages the pool of resources, and it finds and assigns resources required by your applications as needed.

Placement constraints

Sometimes, you do care how components in your application are laid out on a PaaS cluster. For example, if your cluster comprises multiple node types with different capacities, you might want to put certain components on specific nodes. In this case, your application can dictate where PaaS places different components by defining placement constraints. In addition, if you want to minimize the latency between two components that frequently interact with each other, you can suggest that PaaS keep them in close proximity. In some other cases, you might want to distribute the components far apart so that a failing host won't bring down all the components. We'll discuss placement constraints later in this book.

Such clear separation of concerns brings several significant benefits. First, it enables workloads to be transferred from host to host as needed. When a host fails, the workloads on the failing host can be migrated quickly to another healthy host, providing fast failovers. Second, it allows higher compute density because independent workloads can be packed into the same host without interfering with one another. Third, as launching and destroying application instances usually is much faster than booting up and shutting down machines, system capacity can be scaled dynamically to adapt to workload changes. Fourth, such separation also allows applications to be architected, developed, and operated without platform lock-in. You can run the same application on-premises or in the cloud, as long as these environments provide the same mechanism to schedule CPU, memory, and disk resources.

Service Fabric concepts

In this section, you first briefly review the architecture of Service Fabric. Then, you learn about some of the key concepts of Service Fabric in preparation for service development.

Architecture

An overview of Service Fabric architecture is shown in Figure 1-1. As you can see, Service Fabric is a comprehensive PaaS with quite a few subsystems in play. The discussion here gives you a high-level overview of these subsystems. We'll go into details of each of the subsystems throughout this book, so don't worry if you are not familiar with some of the terms.

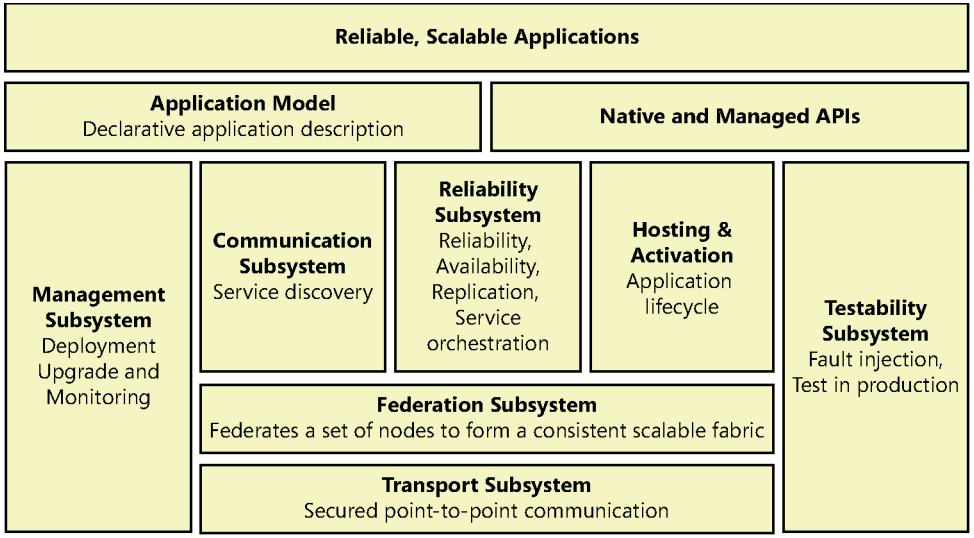


FIGURE 1-1 Service Fabric architecture

The subsystems shown in Figure 1-1 are as follows:

- **Transport subsystem** The transport subsystem is a Service Fabric internal subsystem that provides secured point-to-point communication channels within a Service Fabric cluster and between a Service Fabric cluster and clients.
- **Federation subsystem** The federation subsystem provides failure detection, leader election, and consistent routing, which form the foundation of a unified cluster. We'll examine these terms in upcoming chapters.
- **Reliable subsystem** The reliable subsystem manages state replication, failovers, and load balancing, which a highly available and reliable system needs.
- **Management subsystem** The management subsystem provides full application lifetime management, including services such as managing application binaries; deploying, updating and deprovisioning applications; and monitoring application health.
- **Hosting subsystem** The hosting subsystem is responsible for managing application life cycles on a cluster node.
- **Communication subsystem** The primary task of the communication subsystem is service discovery. With complete separation of workloads and infrastructure, service instances may migrate from host to host. The communication subsystem provides a naming service for clients to discover and connect to service instances.
- **Testability subsystem** The idea of test in production was popularized by the Netflix Chaos Monkey (and later the Netflix Simian Army). The testability subsystem can simulate various failure scenarios to help developers shake out design and implementation flaws in the system.

Nodes and clusters

To understand Service Fabric clusters, you need to know about two concepts: node and cluster.

- **Node** Technically, a node is just a Service Fabric runtime process. In a typical Service Fabric deployment, there's one node per machine. So you can understand a node as a machine (physical or virtual). A Service Fabric cluster allows heterogeneous node types with different capacities and configurations.
- **Cluster** A cluster is a set of nodes that are connected to form a highly available and reliable environment for running applications and services. A Service Fabric cluster can have thousands of nodes.

Figure 1-2 is a simple illustration of a Service Fabric cluster. Notice that all nodes are equal peers; there are no master nodes or subordinate nodes. Also notice that although in the diagram the nodes are arranged in a ring, all the nodes can communicate directly with each other via the transport subsystem.

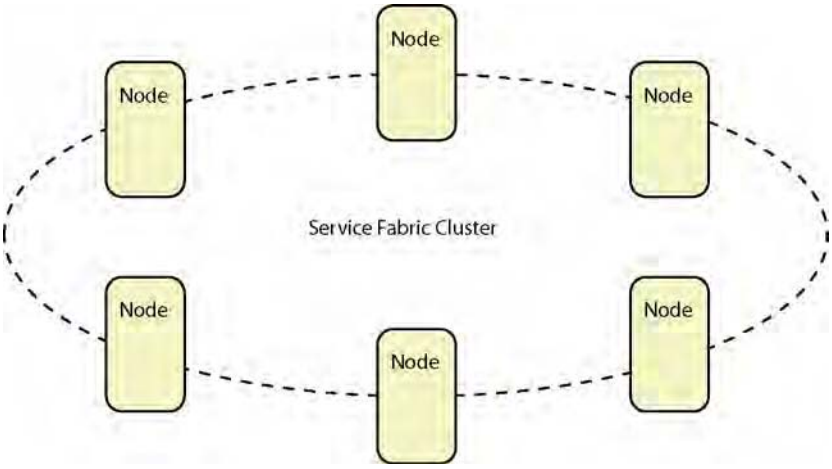


FIGURE 1-2 A Service Fabric cluster

Note Node and containers

In addition to physical machines and virtual machines, nodes can reside in Windows-based Docker containers, which are part of Windows Server 2016. Containerization is described in more detail in Chapter 17, "Advanced service hosting."

A Service Fabric cluster provides an abstraction layer between your workloads and the underlying infrastructure. Because you can run Service Fabric clusters on both physical machines and virtual machines, either on-premises or in the cloud, you can run your Service Fabric applications without modifications in a variety of environments such as on-premises datacenters and Microsoft Azure.

Applications and services

A Service Fabric *application* is a collection of services. A *service* is a complete functional unit that delivers certain functionalities.

You author a Service Fabric application by defining the Application Type and associated Service Types. When the application is deployed to a Service Fabric cluster, these types are instantiated into application instances and service instances, respectively.

An application defines an isolation unit in Service Fabric. You can deploy and manage multiple applications independently on the same cluster. Service Fabric keeps their code, configuration, and data isolated from one another. You can deploy multiple versions of an application on the same cluster.

Partitions and replicas

A service can have one or more partitions. Service Fabric uses partitions as the scaling mechanism to distribute workloads to different service instances.

A partition can have one or more replicas. Service Fabric uses replicas as the availability mechanism. A partition has one primary replica and may have multiple secondary replicas. The states of replicas are synchronized automatically. When a primary replica fails, a secondary replica automatically is promoted to primary to keep service availability. And the number of secondary replicas is brought back to desired level to keep enough redundancy.

We'll introduce partitions and replicas in more detail in Chapter 2, Chapter 3, and Chapter 7.

Programming modes

Service Fabric provides two high-level frameworks to build applications: the Reliable Service APIs and the Reliable Actor APIs.

- The *Reliable Service APIs* provide direct access to Service Fabric constructs such as reliable collections and communication stacks.
- The *Reliable Actor APIs* provide a high-level abstraction layer so that you can model your applications as a number of interacting actors.

With Reliable Service APIs, you can add either stateless services or stateful services to a Service Fabric application. The key difference between the two service types is whether service state is saved locally on the hosting node.

Stateless vs. stateful

Some services don't need to maintain any states across requests. Let's say there's a calculator service that provides both an Add operation and a Subtract operation. For each of the service calls, the service takes in two operands and generates a result. The service doesn't need to maintain any contextual information between calls because every call can be carried out based solely on given parameters. The

service behavior is not affected by any contextual information; that is, adding 5 and 3 always yields 8, and subtracting 6 from 9 always yields 3.

The majority of services, in contrast, need to keep some sort of states. A typical example of such a service is a shopping cart service. As a user adds items to the cart, the state of the cart needs to be maintained across different requests so that the user doesn't lose what she has put in the cart.

Services that don't need to maintain states or don't save states locally are called *stateless* services. Services that keep local states are called *stateful* services. The only distinction between a stateful service and a stateless service is whether the state is saved locally. Continuing with the previous shopping cart example, the service can be implemented as a stateless service that saves shopping cart states in external data storage or as a stateful service that saves shopping cart states locally on the node.

Note "Has state" and "stateful"

Most services have states. However, this doesn't mean they are stateful. The only difference between stateful services and stateless services is where states are stored.

A stateful service can cause some problems. When a service is scaled out, multiple instances share the total workload. For a stateless service, requests can be distributed among the instances because it doesn't matter which instance handles the specific request. For a stateful service, because each service instance records its own state locally, a user session needs to be routed to the same instance to ensure a consistent experience for the user. Another problem with a stateful service is reliability. When a service instance goes down, it takes all its state with it, which causes service interruptions for all the users who are being served by the instance.

To solve these problems, a stateful service can be transformed into a stateless service by externalizing the state. However, this means every service call will incur additional calls to an external data source, increasing system latency. Fortunately, Service Fabric provides a way to escape this dilemma, which we'll discuss in Chapter 3.

Getting started

To get started with Service Fabric development, you need two things:

- A development environment
- A Service Fabric cluster

In this section, first you'll set up a local development, which includes a local multinode cluster that allows you to deploy and test your applications. Then, you'll provision a managed Service Fabric cluster on Microsoft Azure. This book primarily focuses on developments using C# in Visual Studio 2015. However, we'll briefly cover developments using other languages such as Node.js.

Setting up a development environment

To set up a development environment, you'll need Visual Studio 2015 and Service Fabric SDK. You can install Service Fabric SDK via Microsoft Web Platform Installer (Web PI, *https://www.microsoft.com/web/downloads/platform.aspx*). Just follow the installation wizard and accept all default options to complete the installation. This book uses the Preview 2.0.135 version.

In addition, install the following tools:

- Latest version of Microsoft Azure SDK for .NET (using Web PI, this book uses 2.8.1)
- Latest version of Microsoft Azure PowerShell (using Web PI, this book uses 1.0)

Service Fabric SDK provides a local multinode Service Fabric cluster to which you can deploy and test your applications.

Provisioning a Service Fabric cluster on Azure

Although you can use the local cluster provided by Service Fabric SDK for local development and tests, you'll want a hosted cluster on Azure for your production deployments.

Note Microsoft Azure subscription

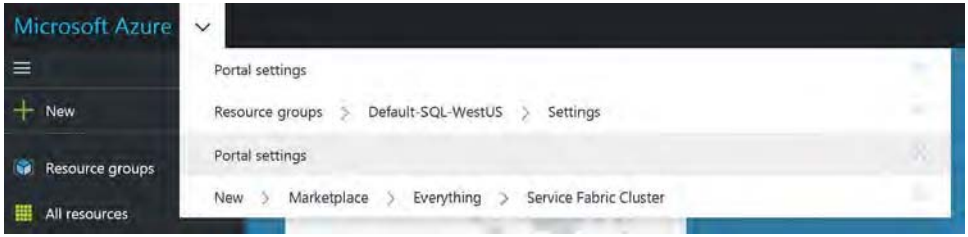
To use Microsoft Azure, you need a Microsoft Azure subscription. If you don't have one, you can apply for a free one-month trial at *https://azure.microsoft.com/pricing/free-trial/*.

You can follow these steps to create a new Service Fabric cluster.

Microsoft Azure management portal terms

As you click links in the portal, the display areas that expand to the right are called *blades*. You also may see the following terms used in talks and articles:

- **Hub** A hub gathers and displays information from multiple data sources. For instance, all notifications from different services are displayed in a centralized notification hub, which can be brought up by the Bell icon on the top command bar.
- **Dashboard** The home page after you log in is called a dashboard, where you can pin various types of resources for quick access.
- **Tile** Each item you pin on the dashboard is represented by a tile.
- **Journey** As you go through a workflow, your navigation steps are recorded as a journey. You can see the history of your journey at the top of the page, and you can click any of the steps to track back or to jump ahead. Journeys are recorded automatically, and you can access previous journeys by clicking the down arrow icon beside the Microsoft Azure label, as shown in the following figure.



To provision a Service Fabric cluster, complete the following steps:

1. Sign in to Microsoft Azure management portal (*https://portal.azure.com*).
2. Click the New icon in the upper-left corner of the home page. Then, click Marketplace, as shown in Figure 1-3.

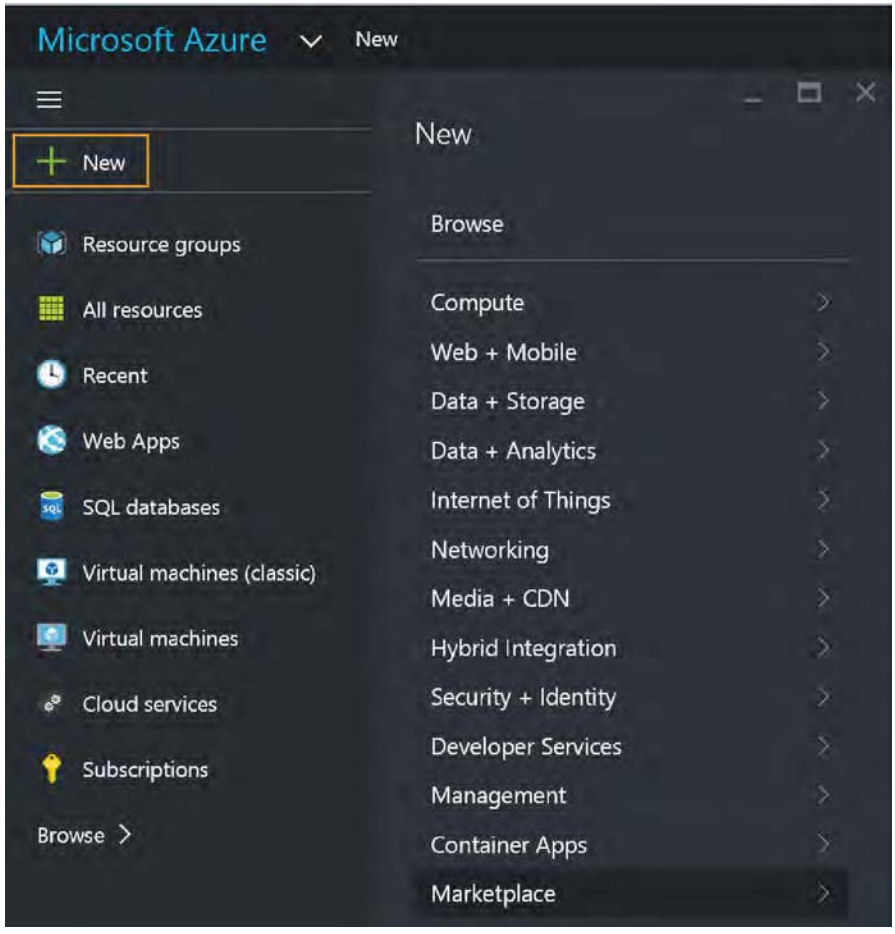


FIGURE 1-3 Create a new resource on Microsoft Azure

3. Under the Everything category, type **service fabric** in the Search box and press Enter. You'll see a Service Fabric Cluster entry, as shown in Figure 1-4. Click the entry to create a new Service Fabric cluster.

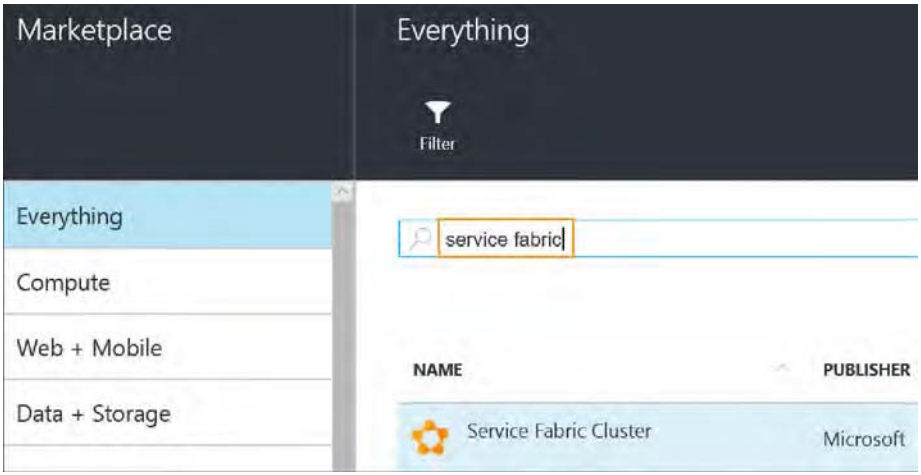


FIGURE 1-4 Service Fabric in Marketplace

4. On the Service Fabric Cluster blade, click the Create button to continue, as shown in Figure 1-5.

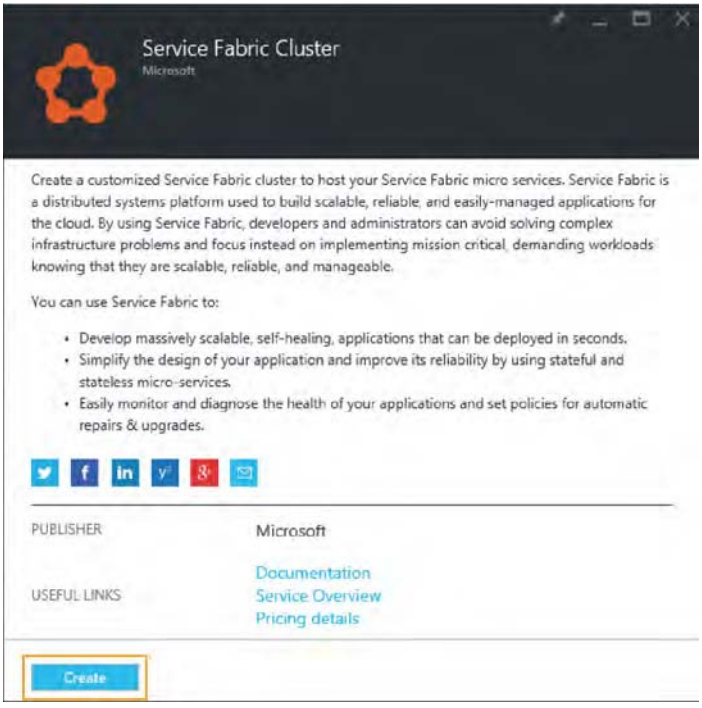


FIGURE 1-5 Service Fabric template blade

5. On the Basics blade, enter a Cluster Name. Enter the user credentials for VM. Select the Azure Subscription you want to use, and type a name for the new Resource Group. Then, pick an Azure Location where you want the cluster to be hosted, and click OK to continue, as shown in Figure 1-6.

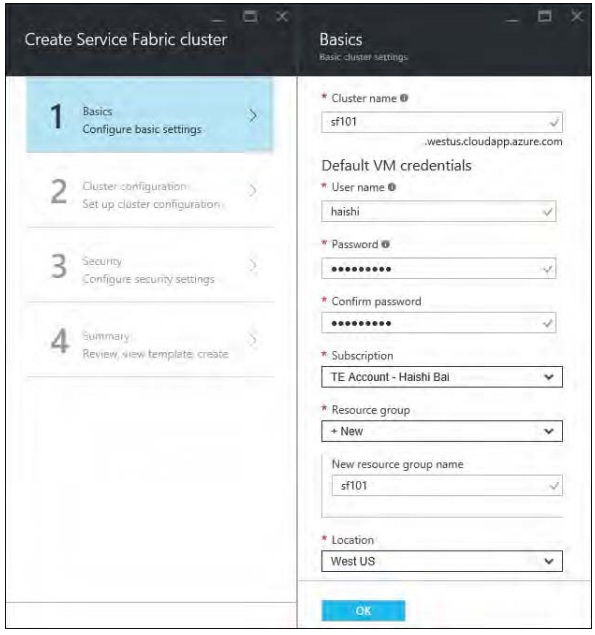


FIGURE 1-6 Service Fabric Cluster creation blade

Note Azure resource groups

A *resource group* is a collection of resources on Azure. On Azure, every resource, such as a virtual machine or a virtual network, belongs to a resource group. A resource group defines a management boundary and a security boundary. You can provision and deprovision all resources in a resource group as a logical unit. And you can apply group-level Role-Based Access Control (RBAC) policies, which are inherited by all members in the group.

6. Click Node Type and create a new node type configuration. (You'll find more information about types of nodes later in this book.) In the Node Type Configuration blade, enter a name for the node type and pick a virtual machine size. Type **80** for the Custom Endpoints value, and then click OK , as shown in Figure 1-7.

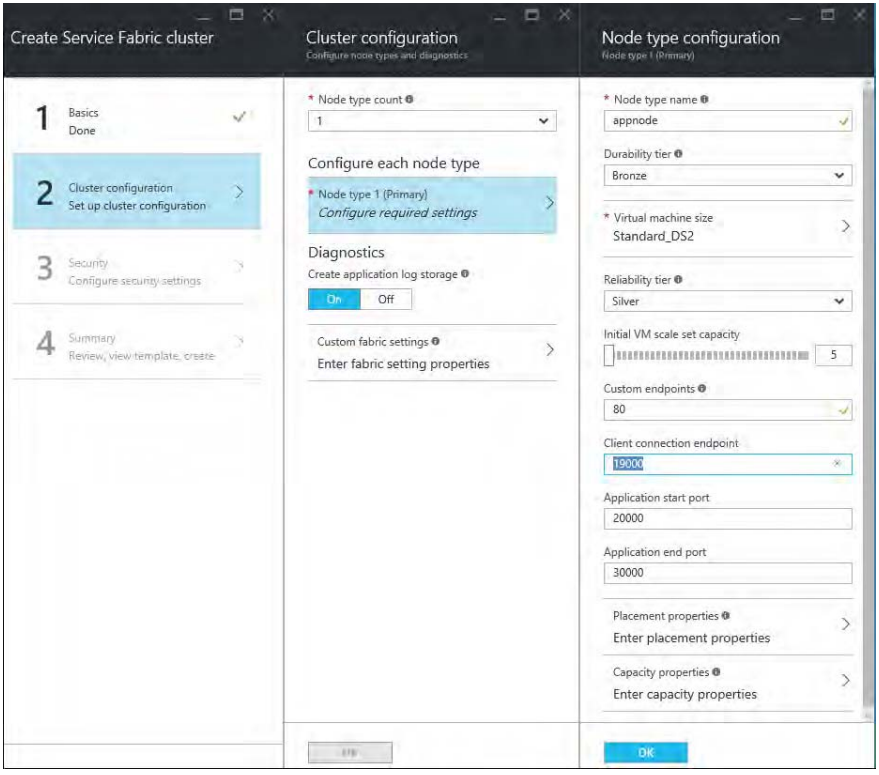


FIGURE 1-7 Service Fabric Cluster settings blade

7. Change the Security mode to Unsecure, and follow the creation wizard to complete provisioning the cluster.

8. The provisioning process takes a few minutes. Once that is done, you'll have a new tile on your dashboard to access the cluster. Figure 1-8 shows the cluster blade, on which you can find the cluster public address (in the format of <cluster name>.<region>.cloudapp.azure.com) and the port number (the default is 19000). You'll need this information to connect to the cluster later.

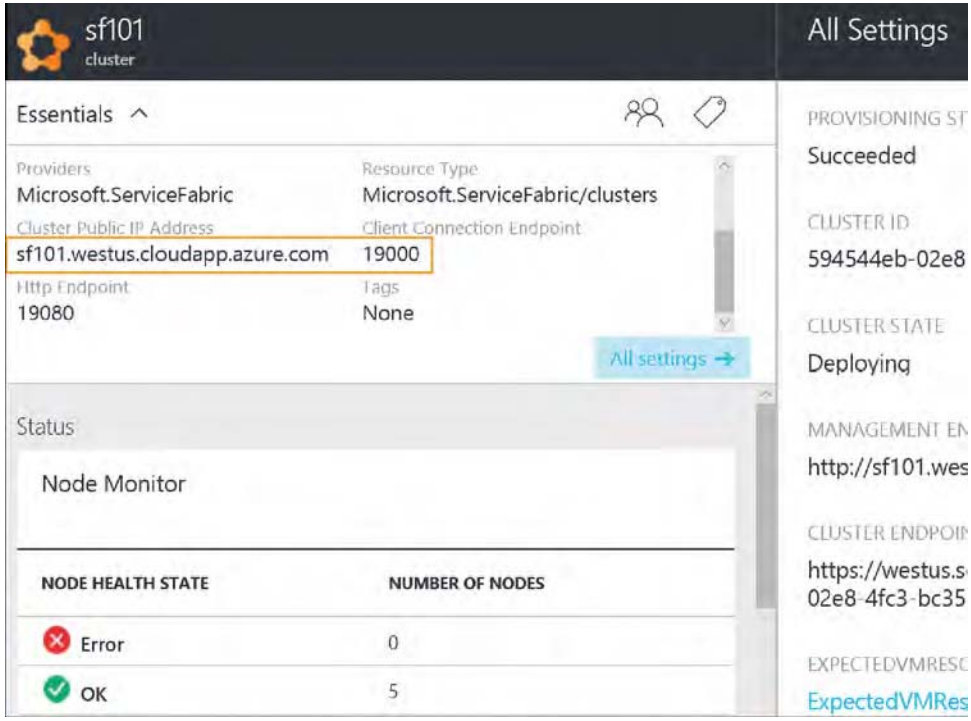


FIGURE 1-8 Service Fabric cluster blade

Hello, World

This is the moment you've been waiting for—a chance to implement the beloved "Hello, World" in a new way.

A tribute to Hello, World

The following code was my first "Hello, World" program, which was written in BASIC about 27 years ago:

```
10 PRINT "Hello, World"
20 END
```

What's yours? I'm glad to see that such simplicity and elegance is being carried over throughout the years collectively by the community to get developers started with new languages and platforms. Of course, because Service Fabric is designed to tackle complex problems, you need to inherit some established frameworks and structures from the platform. However, as you'll see in a moment, the "Hello, World" program still calls for only a couple lines of changes.

Now you are ready to create your first Service Fabric application. This application contains a stateless service that generates a time stamped "Hello World" string every five seconds.

1. Launch Visual Studio 2015 as an administrator.

> **Note** Launching Visual Studio as an administrator
>
> You need to launch Visual Studio as an administrator in this case because you are going to test the application with a local test cluster, which needs administrative rights to be launched.

2. Create a new project named HelloWorldApplication using the Cloud\Service Fabric Application template, as shown in Figure 1-9.

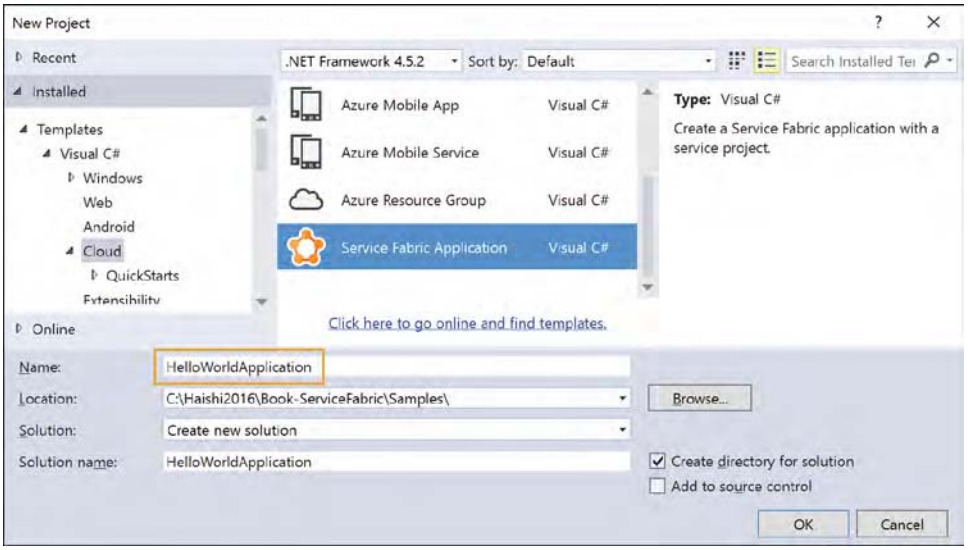


FIGURE 1-9 New Project dialog box

3. In the New Service Fabric Service dialog box, select the Stateless Service template, enter **HelloWorldService** as the Service Project Name, and then click OK to create the Hello World service, as shown in Figure 1-10.

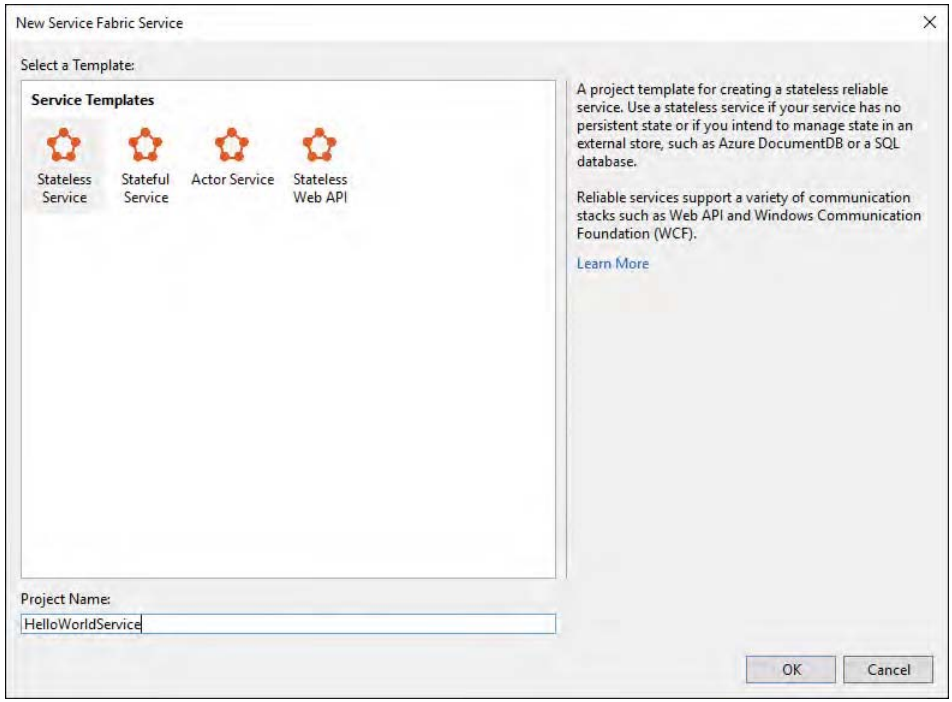


FIGURE 1-10 New Service Fabric Service dialog box

Now, in the solution you have two projects: A HelloWorldApplication project for the Service Fabric application and a HelloWorldService project for the stateless service. You'll go through the rest of the solutions in a moment. For now, focus on the *HelloWorldService* class in the service project:

```
internal sealed class HelloWorldService : StatelessService
{
    public HelloWorldService(StatelessServiceContext context)
        : base(context)
    { }
    protected override IEnumerable<ServiceInstanceListener>
CreateServiceInstanceListeners()
    {
        return new ServiceInstanceListener[0];
    }
    protected override async Task RunAsync(CancellationToken cancellationToken)
    {
        long iterations = 0;
        while (true)
        {
            cancellationToken.ThrowIfCancellationRequested();
            ServiceEventSource.Current.ServiceMessage(this, "Working-{0}", ++iterations);
            await Task.Delay(TimeSpan.FromSeconds(1), cancellationToken);
        }
    }
}
```

4. To implement a stateless service, your service class needs to inherit from the *StatelessService* base class. Service Fabric doesn't mandate a communication protocol. Instead, you can plug in different communication stacks by providing an ICommunicationListener implementation. You'll see a number of communication stack implementations throughout this book. For this example, you'll skip the communication stack, which means your service is a background service that doesn't take any client requests.

5. Modify the *RunAsync* method to implement your service logic.

```
protected override async Task RunAsync(CancellationToken cancellationToken)
{
  while (!cancellationToken.IsCancellationRequested)
  {
ServiceEventSource.Current.ServiceMessage(this, "Hello World at " +
DateTime.Now.ToLongTimeString());
    await Task.Delay(TimeSpan.FromSeconds(5), cancellationToken);
  }
}
```

As you can see in this code snippet, to implement a background service, all you need to do is override the *RunAsync* method and construct your processing loop.

Note Cancellation token

In the .NET asynchronous programming pattern, the CancellationToken structure is used to propagate notification that operations should be canceled. Because a cancellation token can be passed to multiple threads, thread pool work items, and Task objects, it can be used to coordinate cancellation across these boundaries.

When the token's IsCancellationRequested property is set to True, the owning object has requested cancellation of associated tasks. You should stop processing when receiving the notification. If you are calling downstream tasks in your code, you should pass the cancellation token along so that the downstream tasks can be cancelled, like the Task.Delay call in the previous code sample.

6. Service Fabric SDK automatically generates an Event Tracing for Windows (ETW) event source implementation for you to generate trace events. Examine the generated *ServiceEventSource* class if you are interested.

Note Use ETW for tracing and logging

It's recommended to use ETW for tracing and logging because ETW is fast and has minimum impact on your code performance. Furthermore, because Service Fabric also uses ETW for internal tracing, you can view your application logs interleaved with Service Fabric traces, making it easier to understand the relationships between your applications and Service Fabric. Last but not least, because ETW tracing works both in local environment and in the cloud, you can use the same tracing mechanism across different environments.

7. Now, you can press F5 to run the application. Visual Studio will launch a test cluster, deploy the application, and start the services. Once the service is launched, you can see the "Hello World" output in the Diagnostic Events window, as shown in Figure 1-11.

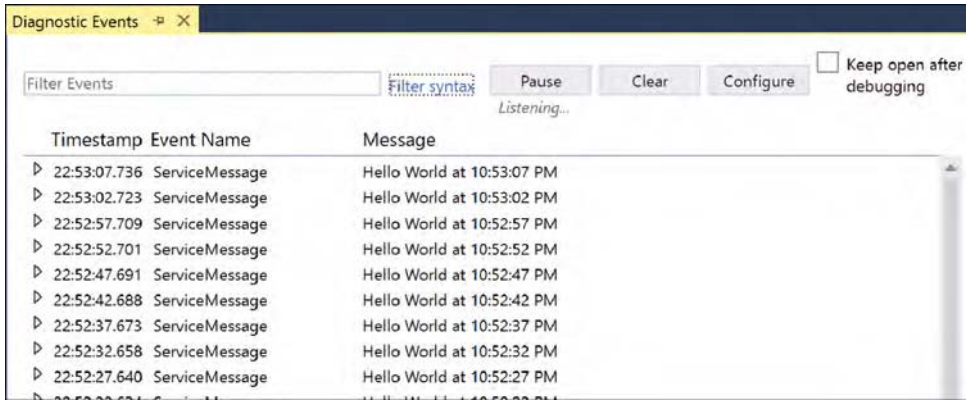


FIGURE 1-11 Diagnostic Events window

Note Bring up the Diagnostic Events view

If you don't see the Diagnostic Events view, you can bring it up by using the View\Other Windows\Diagnostic Event Viewer menu.

8. Click Stop Debugging on the Visual Studio toolbar or press Shift+F5 to stop the debugging session. If you bring back the Diagnostic Event view, you can see the "Hello World" strings are still coming in. This is because the Stop Debugging button only stops your debugging session; it doesn't stop the service.

Congratulations! You've implemented, deployed, and tested your first Service Fabric application. Next, you'll take a peek into the local cluster.

Managing your local cluster

There are multiple ways to manage your local cluster. You can use Visual Studio Server Explorer, Cloud Explorer, Service Fabric Local Cluster Manager, or PowerShell. You'll walk through all the options in this section.

Visual Studio Server Explorer

You can bring up Server Explorer from the Visual Studio View\Server Explorer menu. Under the Azure node, you'll see a Service Fabric Cluster (Local) node, along with other Azure resource types if you have Azure SDK installed. Figure 1-12 shows that on my cluster I have five nodes, and I have a single Hello World application deployed. You should note that applications and cluster nodes are presented separately, showing the clear separation between workloads and cluster resources that was introduced earlier in this chapter.

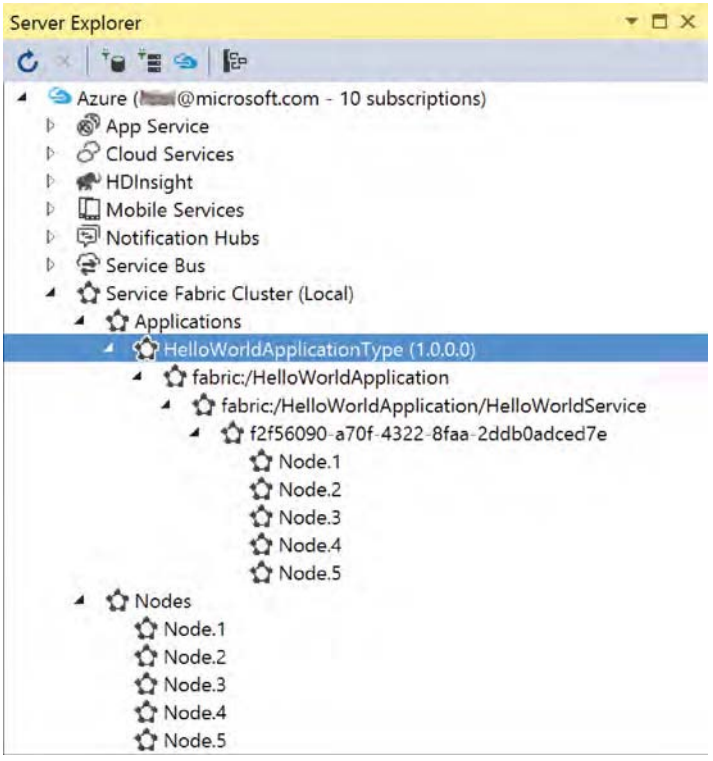


FIGURE 1-12 Server Explorer

The Nodes view in Figure 1-12 seems straightforward—it shows a cluster that consists of five nodes. The application view, however, needs some explanation. What do those levels of nodes mean? Don't worry, you'll go through each of them next.

Application Type node

At the top level, there is an Application Type node that represents an application type, which in this case is HelloWorldApplicationType. When you write your application code in Visual Studio, you define an application type. When the application is deployed, you get an application instance on the cluster. This relationship is similar to the relationship between a class and an object in Object Oriented Programming (OOP).

Application Instance node

Below the Application Type node is the Application Instance node, which is identified by the application's name (fabric:/HelloWorldApplication in this case). By default, an application is named after the corresponding application type name. But you can change to any other name in the format of fabric:/<string>; for example, fabric:/MyApplication.

Service Type node

Under the Application Instance node, you have Service Type nodes. Each node represents a registered service type, such as the HelloServiceType in this example. Each service in a Service Fabric application is named as fabric:<application name>/<service name>. This name also is the address of the service. Service Fabric built-in naming service resolves this name to the actual service instance address at run time.

Partition node

A partition is identified by a GUID, suggesting it's not something that a client would use to address a partition directly. Instead, the Service Fabric naming service will figure out the correct partition to which a service request should be sent.

Replica node

Figure 1-12 shows that by default, Service Fabric maintains multiple replicas for a partition for high availability. In this case, the Hello World service has a single partition with five replicas.

Note Replica and service instance

A replica is a service instance. The term *service instance* often is used in logical descriptions of deployment topologies, and the term *replica* often is used when explicit replica behaviors or concepts are being discussed.

Visual Studio Cloud Explorer

Microsoft Azure SDK comes with a Visual Studio extension named Cloud Explorer, which you can access by the View\Cloud Explorer menu. Cloud Explorer is similar to Server Explorer in terms of Service Fabric cluster management functionalities, as shown in Figure 1-13.

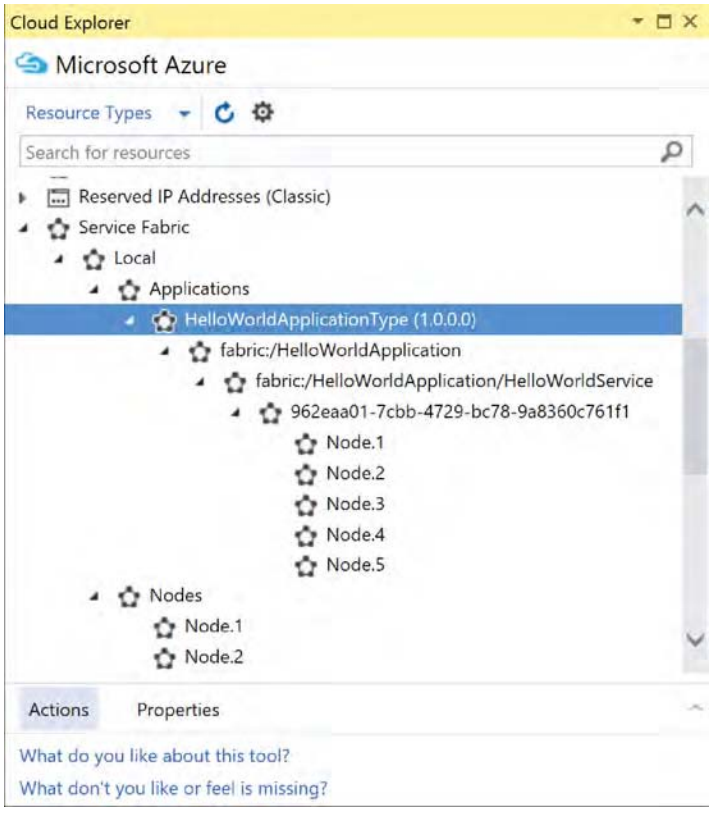


FIGURE 1-13 Cloud Explorer

Service Fabric Explorer

Neither Server Explorer nor Cloud Explorer is designed as a full-fledged management tool. They are designed for you to navigate and view your cloud resources and server resources easily and with limited management functionality.

Service Fabric SDK provides a powerful tool named Service Fabric Explorer. To bring up the management UI, you can use any browser and navigate to *http://localhost:19080/Explorer*.

The left panel of the Service Fabric Explorer looks much like Server Explorer or Cloud Explorer. However, the tool also has a details panel to the right providing very detailed information on the currently selected item, as shown in Figure 1-14.

FIGURE 1-14 Local Cluster Manager

The management tool is loaded with features. You'll use this tool frequently throughout this book for different scenarios. For now, perform a little exercise to familiarize yourself with the tool. In this exercise, you'll delete the Hello World application from the cluster.

1. In the Service Fabric Explorer, select the fabric:/HelloWorldApplicationnode in the left pane. Then, in the right pane, click the Actions button and then click the Delete Application menu, as shown in Figure 1-15.

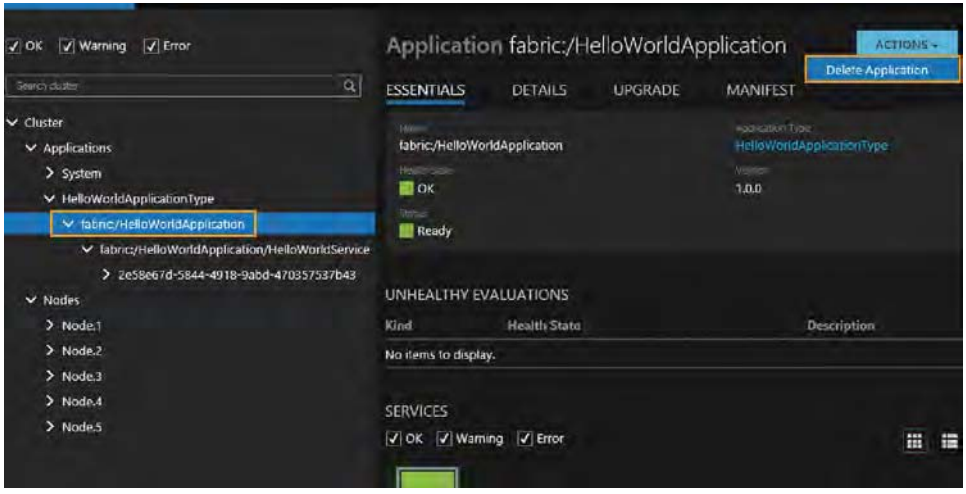


FIGURE 1-15 Deleting a new application instance

2. In the Confirm Application Deletion dialog box, shown in Figure 1-16, click the Delete Application button to continue.

FIGURE 1-16 Confirm Application Deletion dialog box

3. The UI refreshes itself every few seconds. Once the UI is refreshed, you can see the application instance has been removed, as shown in Figure 1-17.

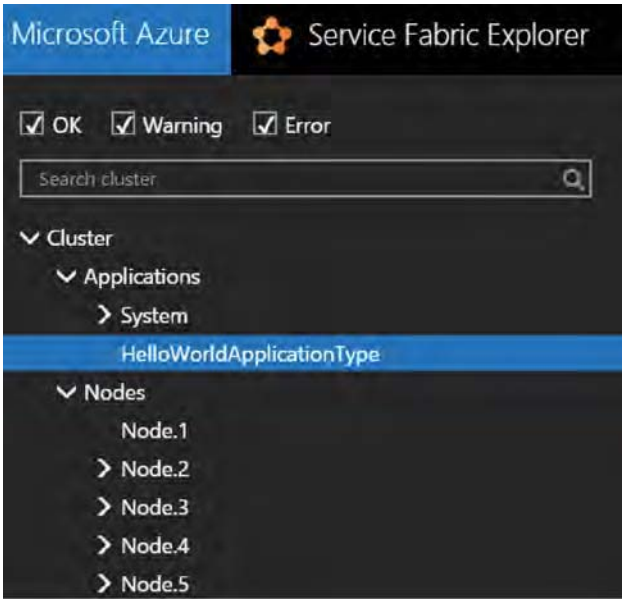


FIGURE 1-17 Two application instances

Note Resetting the cluster

Service Fabric SDK installs a Service Fabric icon on your taskbar, which provides a couple of shortcut menus for some cluster-level operations. This comes in handy when you want to reset everything and make a fresh start. To reset the cluster, right-click the Service Fabric icon on the taskbar and select the Reset Cluster menu.

Windows PowerShell

Windows PowerShell is a powerful automation and configuration management framework. Service Fabric SDK installs a number of PowerShell cmdlets for you to manage your applications and clusters using command lines or automation scripts.

If you look under the Scripts folder of the HelloWorldApplication, you can find a Deploy-FabricApplication.ps1, which in turn calls a number of prebuilt scripts under your Service Fabric SDK folder (C:\Program Files\Microsoft SDKs\Service Fabric\Tools\Scripts by default) to deploy, upgrade, and remove your applications.

To get started, open a new Windows PowerShell window and use the Connect-ServiceFabricCluster cmdlet to connect to the local cluster, as shown in Figure 1-18.

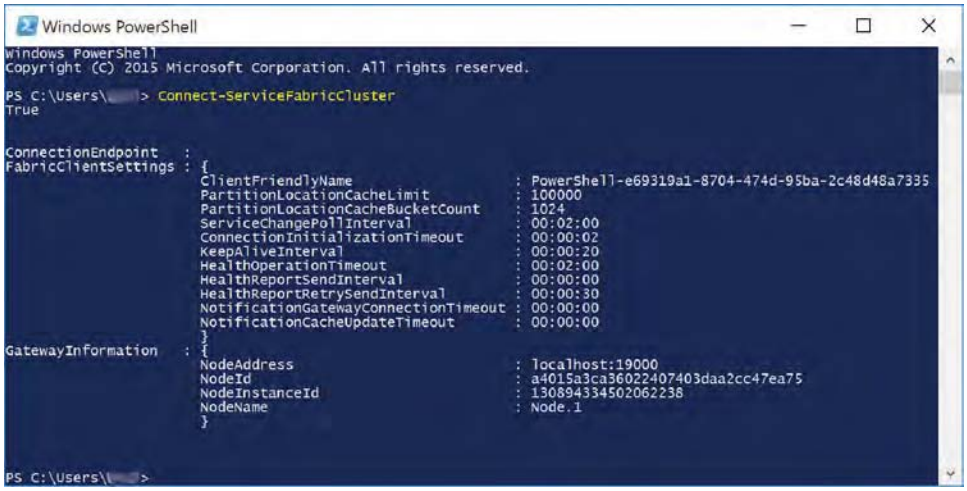


FIGURE 1-18 Connect-ServiceFabricCluster cmdlet

Next, try a few commands. You'll be introduced to more cmdlets as you go through the chapters.

- List application instances.

  To list application instances on the cluster, type the following cmdlet:

  ```
  Get-ServiceFabricApplication
  ```

  The above cmdlet generates the following output on my environment (before the application instance is removed. To restore the instance, press F5 in Visual Studio to redeploy application):

  ```
  ApplicationName        : fabric:/HelloWorldApplication
  ApplicationTypeName    : HelloWorldApplicationType
  ApplicationTypeVersion : 1.0.0.0

           ApplicationStatus      : Ready
           HealthState             : Ok
           ApplicationParameters  : { "HelloWorldService_InstanceCount" = "-1" }
  ```

- List cluster node names and statuses.

  To list all the nodes and their current statuses, use the following cmdlet:

  ```
  Get-ServiceFabricNode | Format-Table NodeName, NodeStatus
  ```

  A healthy cluster would look like this:

  ```
  NodeName NodeStatus
  -------- ----------
  Node.1           Up
  Node.2           Up
  Node.3           Up
  Node.4           Up
  Node.5           Up
  ```

Additional information

At the time of this writing, Service Fabric is still in preview. It's likely that service APIs, tooling experiences, and management UIs will change over time. However, key concepts covered in Part I and Part II of this book and the patterns and applicable scenarios in Part III and Part IV of this book should remain valid in future releases.

For up-to-date Service Fabric documentations, please visit *https://azure.microsoft.com/documentation/services/service-fabric/*.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z