

Zen of Cloud

**Learning Cloud Computing by Examples
on Microsoft Azure**

Haishi Bai



CRC Press
Taylor & Francis Group

Zen of Cloud

**Learning Cloud Computing by Examples
on Microsoft Azure**

Zen of Cloud

**Learning Cloud Computing by Examples
on Microsoft Azure**

Haishi Bai



CRC Press

Taylor & Francis Group

Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2015 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works
Version Date: 20140624

International Standard Book Number-13: 978-1-4822-1581-6 (eBook - PDF)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Visit the Taylor & Francis Web site at
<http://www.taylorandfrancis.com>

and the CRC Press Web site at
<http://www.crcpress.com>

Contents

Foreword	xiii
----------------	------

SECTION I CLOUD COMPUTING FUNDAMENTALS

1 Overview of Cloud Computing	3
1.1 History	3
1.2 Essence of Cloud Computing	5
1.2.1 Elasticity.....	6
1.2.1.1 On and Off Mode	7
1.2.1.2 Rapid-Growth Mode.....	8
1.2.2 Availability	8
1.2.2.1 Fault Domain	9
1.2.2.2 Update Domain.....	10
1.2.3 Scalability.....	10
1.3 Microsoft Azure Overview	11
1.3.1 IaaS (Infrastructure as a Service)	11
1.3.2 PaaS (Platform as a Service).....	12
1.3.3 SaaS (Software as a Service).....	13
1.3.4 Cost Calculation.....	15
1.4 Preparing the Development Environment for Microsoft Azure.....	16
1.4.1 Subscribe to Microsoft Azure.....	16
1.4.2 Install Software Development Kit	16
1.5 Introduction of Microsoft Azure Management Portal.....	16
1.5.1 Sign In.....	17
1.5.2 Page Layout	17
1.6 Summary.....	19
2 Building Websites on the Cloud.....	21
2.1 Microsoft Azure Websites	21
2.2 Website Deployment and Upgrade	25
2.3 Integration with Source Control Systems.....	32
2.4 Scaling of Websites	37

2.4.1	Vertical Scaling.....	37
2.4.2	Horizontal Scaling.....	39
2.4.3	Autoscaling.....	39
2.5	Migrating Existing ASP.NET Websites	40
2.5.1	Azure Websites Runtime Environment.....	40
2.5.2	Data Storage.....	41
2.5.3	Session States.....	41
2.6	Website Gallery	41
2.7	Website Configuration.....	42
2.8	Website Diagnostics and Monitoring.....	49
2.8.1	Website Diagnostics.....	49
2.8.2	Website Monitoring.....	52
2.8.3	Custom Domain Names.....	54
2.9	Summary.....	55
3	Cloud Service Fundamentals.....	57
3.1	Microsoft Azure Cloud Services	57
3.2	Cloud Services and Roles.....	61
3.2.1	Role.....	63
3.2.2	Cloud Service	63
3.3	Basic Steps of Cloud Service Deployment.....	64
3.4	Cloud Service Deployments and Upgrades.....	69
3.4.1	Incremental Updates (Update Domain Walk)	70
3.4.2	Simultaneous Updates	70
3.4.3	Multiple Deployment Environments	70
3.5	Instances and Load Balancing	74
3.5.1	Instances.....	75
3.5.2	Load Balancing.....	76
3.6	Configuration File and Definition File	79
3.6.1	Cloud Service Definition File (.csdef).....	80
3.6.2	Cloud Service Configuration File (.cscfg)	81
3.7	Summary.....	84
4	Advanced Cloud Service.....	85
4.1	Endpoint Types	85
4.1.1	Input Endpoint.....	85
4.1.2	Internal Endpoint.....	85
4.1.3	InstanceInput Endpoint.....	86
4.2	Worker Role	88
4.2.1	Worker Role Application Scenarios.....	90
4.3	Inter-Role Communications	96
4.3.1	Options for Inter-Role Communication	96
4.4	Role Lifecycle	100
4.4.1	Process of Deploying and Launching a Role Instance.....	100
4.4.2	Role Instance Statuses	101
4.5	Startup Tasks.....	102

4.5.1	Defining Startup Tasks.....	102
4.5.2	Startup Task Properties	103
4.6	Diagnostics and Debug	109
4.6.1	Debugging Locally	109
4.6.2	Microsoft Azure Diagnostics	109
4.6.3	IntelliTrace	114
4.6.4	Monitoring Cloud Service	119
4.7	Developer Community.....	123
4.8	Summary.....	125
5	Data Storage: Relational Database.....	127
5.1	Microsoft Azure Data Storage Solutions	127
5.2	SQL Database Overview	129
5.2.1	Differences between an SQL Database and an SQL Server.....	129
5.3	SQL Database Management and Optimization.....	139
5.3.1	SQL Server Management Studio	139
5.3.2	Microsoft SQL Server Data Tools.....	144
5.3.3	Dynamic Management Views.....	145
5.3.4	Query Optimization.....	146
5.4	Data Sync and Migration	149
5.4.1	Data-Tier Application	149
5.4.2	Data Sync	152
5.5	Periodically Backup Your SQL Databases.....	157
5.6	Use MySQL Database	159
5.6.1	Microsoft Azure Store.....	159
5.6.2	Purchasing MySQL Service	160
5.6.3	Other Means to Run MySQL.....	161
5.7	Summary.....	161
6	Data Storage: Storage Services	163
6.1	Local Storage.....	163
6.2	Overview of Microsoft Azure Storage Services	164
6.2.1	Microsoft Azure Storage Account.....	165
6.2.2	Provisioning a Windows Storage Account	167
6.2.3	Storage Account Access Keys.....	168
6.3	Using BLOB Storage	170
6.3.1	BLOB Storage Overview	170
6.3.2	Block BLOB and Page BLOB.....	184
6.3.3	ETag and Snapshots.....	187
6.3.4	REST API	187
6.3.5	Shared Access Signature and Stored Access Policies.....	188
6.3.6	BLOB Update, Copy, and Lease.....	190
6.3.7	Error Handling.....	190
6.4	Using Table Storage.....	190
6.4.1	Table Storage Overview	190
6.4.2	Optimizing Data Partition	191

6.4.3	Query Table Data	204
6.4.4	Other Operations	204
6.4.5	Batch Operations.....	205
6.4.6	Dynamic Table Entities	205
6.4.7	Shared Access Signatures	206
6.5	Use Queue Storage	206
6.5.1	Queue Storage Overview	206
6.5.2	Programmatically Operate Queues	207
6.6	Monitor Storage Accounts	209
6.6.1	Configure Storage Service Monitoring.....	210
6.6.2	Cost of Service Monitoring.....	211
6.7	Summary.....	211
7	Virtual Machines and Virtual Networks.....	213
7.1	Microsoft Azure IaaS.....	213
7.2	Disk Images and Virtual Disks.....	220
7.3	Virtual Machine Communications.....	228
7.3.1	Virtual Machine Endpoints.....	228
7.3.2	Virtual Machines under the Same Cloud Service	230
7.4	Virtual Networks.....	234
7.4.1	Virtual Networks Overview	235
7.4.2	Point-to-Site Virtual Network	237
7.4.3	Site-to-Site Virtual Network.....	243
7.4.4	ExpressRoute.....	243
7.5	Summary.....	243

SECTION II CLOUD SOLUTIONS

8	Cloud Solution Architecture	247
8.1	Client/Server	247
8.1.1	Characteristics of Client/Server Architecture.....	247
8.1.1.1	Benefits.....	247
8.1.1.2	Shortcomings.....	249
8.1.2	Client/Server Architecture on Cloud	249
8.1.3	Multitenant System Design	251
8.1.4	Migrating Client/Server Systems to Cloud	253
8.1.5	Client/Server Systems on Microsoft Azure.....	253
8.1.6	Mobile Clients.....	254
8.2	Browser/Server.....	254
8.2.1	Characteristics of Browser/Server Architecture	255
8.2.2	Browser/Server Architecture on Cloud.....	256
8.2.3	Difficulties of Adapting an Existing Single-Tenant Browser/Server Application for Multitenancy.....	264
8.2.4	Host Single-Tenant Systems on Microsoft Azure for Multiple Tenants	267
8.3	n-Tiered Architecture	269
8.3.1	Characteristics of n-Tiered Architecture	269

8.3.2	n-Tier, MVC, and MVVM	270
8.3.3	Microsoft Azure Service Bus Queue	273
8.3.4	Implementing n-Tiered Services on Microsoft Azure	277
8.4	Distributed System	284
8.4.1	Message-Based Connections	287
8.4.2	Relayed Connections	291
8.5	Summary	299
9	High-Availability Design	301
9.1	Availability	301
9.2	High-Availability Techniques	302
9.2.1	Redundancy	303
9.2.2	Load Balancing	303
9.2.3	Failover	303
9.3	Load Balancing and Health Probe	308
9.4	Competing Consumers	310
9.4.1	Loose Coupling	310
9.4.2	Dynamic Load Balancing	311
9.4.3	Dynamic Scaling	311
9.4.4	Failover	311
9.5	Case Study: High-Availability Service Bus Entities	312
9.5.1	Background	315
9.5.2	Segmented Message Pipelines	316
9.5.3	Paired Namespaces	317
9.5.4	Conclusion	317
9.6	Summary	317
10	High-Reliability Design	319
10.1	Reliability, Availability, and Maintainability	319
10.1.1	Reliability	319
10.1.2	Maintainability	320
10.1.3	Relationships between Availability, Reliability, and Maintainability	320
10.2	Embracing Failures	321
10.2.1	Failures in Operation	321
10.2.2	Failures in State Management	321
10.2.3	Failures in System Design and Implementation	322
10.3	Transient Errors	322
10.3.1	Transient Fault Handling Application Block	323
10.4	Design for Reliability	326
10.4.1	Single Point of Failure	327
10.4.2	Writing Reliable Code	328
10.5	Summary	331
11	High-Performance Design	333
11.1	Microsoft Azure In-Role Cache	333
11.1.1	Overview	334
11.1.2	Deployment Options	334

11.1.3	Cache Features	338
11.1.4	Concurrency Modes	339
11.1.5	Local Cache.....	341
11.1.6	Session State	341
11.2	Microsoft Azure Cache Service.....	345
11.2.1	Overview	346
11.2.2	Cache Service versus In-Role Cache	346
11.2.3	Managing Cache Clusters on Microsoft Azure Management Portal	346
11.2.4	Memcache Support.....	347
11.2.5	Future of Azure Cache.....	348
11.3	Microsoft Azure CDN.....	348
11.4	Asynchronous Operations and Parallel Operations.....	349
11.5	Summary.....	350
12	Claim-Based Architecture	351
12.1	Claim-Based Authentication and Authorization	352
12.1.1	Basic Authentication and Authorization Process.....	353
12.1.2	Authentication and WIF	354
12.1.3	Authentication Broker	354
12.2	Introduction to Microsoft Azure AD.....	356
12.2.1	Managing Microsoft Azure Tenants and Users.....	357
12.2.2	Graph API.....	367
12.3	Microsoft Azure AD New Features	372
12.3.1	Azure Authentication Library	372
12.3.2	Microsoft Azure Active Directory Premium	372
12.4	Summary.....	373
 SECTION III DEVICES AND CLOUD		
13	Mobile Service	377
13.1	Mobile Service Overview.....	377
13.2	Push Notifications	386
13.2.1	Push Notification Overview	386
13.3	Scheduler and API.....	393
13.4	Summary.....	396
14	Internet of Things.....	397
14.1	IoT Overview	397
14.1.1	Radio Frequency Identification.....	398
14.1.2	Artificial Intelligence Equipment.....	398
14.1.3	Wearable Devices	398
14.1.4	Wireless Sensor Network.....	399
14.2	Devices and Cloud	399
14.2.1	Importance of Devices for Cloud.....	399
14.2.2	Importance of Cloud for Devices.....	400
14.3	Challenges of IoT	401
14.4	.NET Micro Framework.....	402

14.4.1	.NET Micro Framework Overview.....	402
14.4.2	.NET Gadgeteer Overview.....	405
14.4.3	Device Integration Sample Scenario	409
14.5	Summary.....	416

SECTION IV SYSTEM INTEGRATION AND PROJECT MANAGEMENT

15	Message-Based System Integration	419
15.1	System Integration.....	419
15.1.1	Integration by Data	420
15.1.2	Shared Business Functions.....	420
15.1.3	Enterprise Service Bus	420
15.2	Message-Based System Integration	422
15.2.1	Content-Based Routing	422
15.2.2	Priority Queue.....	423
15.2.3	Request/Response.....	426
15.2.4	Dead Letter Queue.....	427
15.2.5	Event-Driven Consumer.....	430
15.3	Advanced Message Queuing Protocol.....	433
15.3.1	AMQP Overview	434
15.3.2	AMQP Adoption.....	436
15.4	Advantages of Message-Based Integration	438
15.4.1	Loose Coupling	438
15.4.2	Dynamic Extension	441
15.4.3	Asynchronous Communication	441
15.4.4	Centralized Management	441
15.5	Summary.....	444
16	Source Control and Tests with Visual Studio Online.....	445
16.1	Create a Visual Studio Online Account.....	446
16.2	Source Control with Visual Studio Online.....	446
16.3	Create and Use Unit Tests	452
16.4	Create and Use Load Tests	459
16.5	Summary.....	465
17	Scripting and Automation	467
17.1	Microsoft Azure PowerShell Cmdlets	467
17.1.1	Preparing a Microsoft Azure PowerShell Cmdlets Environment.....	467
17.1.2	Managing Virtual Machines.....	469
17.1.3	Managing Cloud Services.....	472
17.1.4	Managing Microsoft Azure Websites	472
17.1.5	Other Cmdlets	472
17.2	Microsoft Azure Cross-Platform Command Line Tools.....	472
17.2.1	Installing the Command Line Tools.....	474
17.2.2	Getting Started with the Command Line Tools	476
17.3	Microsoft Azure Management API.....	477
17.4	Summary.....	480

18	Azure and DevOps	481
18.1	DevOps Overview.....	481
18.1.1	Everything Is Code.....	481
18.1.2	Everyone Is a Developer.....	482
18.1.3	Every Day Is Release Day	482
18.2	VM Agent and VM Extensions	483
18.2.1	VM Agent	483
18.2.2	VM Extensions.....	483
18.2.3	Custom Script Extension	484
18.2.4	DSC, Puppet, and Chef.....	485
18.3	New Portal	486
18.4	Zen of Cloud.....	487
	Bibliography.....	489

Foreword

This book by Haishi Bai provides a detailed introduction to cloud computing, in general, and to Microsoft Azure, in particular.

Haishi Bai has had an extensive and successful career in information systems, which brought him from the Northeast of China to Beijing and then finally to Silicon Valley and to the Microsoft Head Office in Redmond, Washington as a Microsoft Azure evangelist. His experience in the rise and fall of e-business, boom of social networks, and rise of cloud computing provides him a deep understanding of the pros and cons in cloud computing.

This book explains the various concepts of Azure in a logical and clear manner. It is divided into four sections: cloud computing fundamentals, cloud solutions, devices and cloud, and system integration and project management. Beginners can use this book as a guide in their journey through cloud computing. Experienced cloud developers can benefit from it by studying specific scenarios. The book consists of 69 complete end-to-end examples that provide step-by-step guidance on implementing typical cloud-based scenarios. The examples cover a wide range of application types and technologies with different levels of difficulties. The book also provides practical knowledge, tips and tricks that you can apply to your own work.

This book should provide invaluable help to IT managers who want to stay up to date, developers who want to implement applications using Microsoft Azure, as well as system engineers who are looking to gain in-depth knowledge on cloud computing.

Enjoy the reading as I did and looking forward to your great contributions to this new world!

Pierre Masai
CIO of Toyota Motor Europe

CLOUD COMPUTING FUNDAMENTALS

I

Cloud computing, as its name suggests, is to leverage cloud for computing workloads. These workloads are not new—we have been running applications and services on our workstations and data centers for tens of years. Then why do we need cloud? What benefits are we getting by using it? What are the new scenarios it enables? In this first section of the book, we first look back at the history of cloud computing, and then discuss some unique and exciting capabilities cloud computing brings us. We also study the basics of getting your workloads on cloud using Microsoft developer tools and services. From a developer’s perspective, you can view cloud as a huge resource pool where you can pull out resources to support your application needs, and return the resources when you are done with them. This resource usage model is the foundation of key cloud characteristics such as elasticity, availability, and agility.

Chapter 1

Overview of Cloud Computing

1.1 History

In August 1962, Joseph Carl Robnett Licklider published the famous paper “On-Line Man-Computer Communication,” one of the first conceptions of the future Internet. In the paper, he elaborated on how man would interact with the computer to accomplish various tasks. Not only did he mention the widespread keyboard drag-and-drop operations but also predicted the future of man-computer communication via natural languages. However, the main idea in this paper is how to share the reliable storage and the superb computational capabilities of computers among users in order to accomplish complicated tasks with an online collaboration. The conceptions of resource sharing, dynamic interaction, and remote collaboration laid the foundation for the Internet theory and depicted the preliminary blueprint for cloud computing.

In the past 50 years, although software architecture has evolved through several different stages, from Mainframe to Client-Server, to Browser-Server, to Distributed System, and finally to Cloud Computing (Figure 1.1), providing value-added services with optimized resource utilization remains the theme of software system design.

In the time of Licklider, people concentrated on optimizing the utilization of limited resources on the host systems because terminals had almost no processing power and all computing tasks were carried out on the hosts. With the emerging wide spread use of personal computers, storage capacities and processing power of client machines have been constantly increasing. As a result, more computing and storage requirements can be satisfied directly on client machines. Distributed systems pushed this idea to the extreme and eliminated the need of centralized servers. Many phenomenal distributed systems and applications have emerged and prospered. However, on the flip side, development, maintenance, and management of distributed systems have also shown unprecedented complicity. Then, as browsers and the Internet mature, computing and storage are again pushed to the server side. Browsers replace desktop programs and become the mainstream user front end. Is this a return of mainframe mode? In order to answer this question, we cannot miss mentioning Salesforce.

In 1999, 37-year-old Marc Benioff resigned from his senior vice presidential position at Oracle to found Salesforce. He put forward, for the first time in history, the concept of Software-as-a-Service (SaaS). The SaaS mode shifted the paradigm of software applications in enterprises.

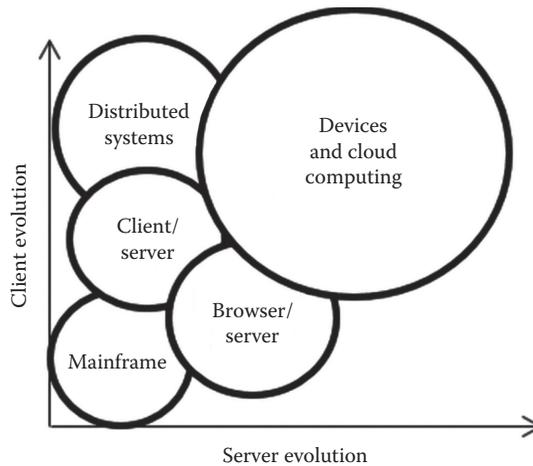


Figure 1.1 Evolution of system application structure.

With SaaS, enterprises do not need to make huge investments to build up and maintain system infrastructures. At the same time, they are no longer obligated to distribute and maintain a huge number of client software. With simple mouse clicks in browsers, users can easily access required functionalities anytime, anywhere, from any connected machines. SaaS helps enterprises to realize that they do not have to put up with the infrastructural costs in order to enjoy the required services. This is a fundamental change in the way software is delivered and consumed. Therefore, SaaS is not a return of mainframe, but a significant milestone in the history of cloud computing.

However, services provided by Salesforce were limited to customer relation management (CRM) and sales. Naturally, people started to reflect on one question: Is it possible to create a general platform, where more services can be hosted and made available to more users? Obviously, it is not an easy job to design, construct, and maintain a universal platform that supports various services and is shared by millions of users. This kind of platform will need to provide not only limitless computing power, efficient and reliable storage, unprecedented network throughputs, and world-class security, but also attractive tariffs to attract more users to complete the transition from using software to consuming service. Facing such a huge hurdle, only few enterprises in the world have the technical and financial power to conquer it.

Amazon became undoubtedly the pioneer in cloud computing. In 2006, Amazon released its elastic computing platform—the EC2. With this release, Amazon claimed the title of the first open cloud platform provider in the world. As a well-experienced tycoon in Internet sales, thanks to years of experience in managing large-scaled data and in dealing with business characterized by large volume with low profit, Amazon had all the necessary ingredients to create such a platform. Soon many enterprises got to know of the platform and began to migrate their existing systems to EC2. Amazon promptly enjoyed its newly gained prosperity and almost became the synonym of cloud computing.

Another company that contributed to the development of cloud computing is Apple of Steve Paul Jobs. This company seems to have less to do with cloud computing but it boosts cloud development in its own way. Almost overnight, Apple swept the electronic market with the iPhone and iPad and forced personal computers to abdicate from the end user market. Mobile devices

have been replacing laptops not only in handling personal matters, but also gradually in handling day-to-day business. The demand for constant access to business functions via various devices is a new challenge to IT departments in enterprises. How to provide secured access? How to keep data synchronized? How to integrate with existing systems? How to handle authentication and authorization? Faced with these challenges, IT departments turned to cloud computing for solutions. Various cloud services began to be adopted by enterprises in order to be able to provide anytime, anywhere accesses from mobile devices. From simple data storing and file sharing to emailing and agenda management, to remote cooperation, to business flow integration, and to internal social network, cloud services are playing increasingly important roles in the daily business with the help of mobile devices.

In 2008, Microsoft announced its participation in the cloud platform business in the PDC event in Los Angeles, and Microsoft Azure entered the landscape of cloud computing. Microsoft's commitment to the cloud platform is comprehensive. The company has committed tremendous resources to provide a world-class cloud platform by developing cloud technologies and constructing world-class large-scale data centers. In addition, Microsoft has also been transferring its own core businesses, such as Office, to the cloud platform. Meanwhile, thanks to its rich experiences in enterprise applications, Microsoft actively pushes the integration of public cloud with on-premise systems. The so-called Hybrid Cloud not only protects customers' existing investment, but also enables them to migrate their existing systems to the cloud platform in a smooth and steady way. In terms of development support, Microsoft remains loyal to its tradition, which is to put the demands of developers first. With Visual Studio, Visual Studio Online, and Microsoft Azure SDKS, Microsoft provides a consistent experience for developers regardless of the types of projects they work on. In addition to providing first-class support to .Net framework on Windows systems, Microsoft Azure is also an open platform that fully supports third-party systems, languages, and tools such as Linux operating systems, PHP, Node.js, Java, Ruby, and Python, as well as various third-party software like MySQL, MongoDB, and GIT.

Cloud computing is only at its starting point in the history of computer and software development. As more enterprises and individual users realize the advantages of cloud computing, the development of cloud computing is accelerating immensely. It is predicated that the cloud computing market will reach a scale of US\$240 billion by 2020. So right now is the perfect moment to join the movement of cloud computing.

1.2 Essence of Cloud Computing

What on earth is the cloud we are talking about? One way to look at the cloud is to think of it as an immense resource pool for storage and computing. Users can access this resource pool from anywhere and at any time to fulfill their needs. The consumers of the cloud do not need to understand any details behind the scenes. As long as they obtain respective endpoint for a service, they can enjoy this service instantly (see Figure 1.2). In other words, service consumers do not need to care about how the service is put together by the provider and how it's made available; they can simply subscribe to the service and use it. This is the fundamental difference in the traditional on-premise solutions, whereas service consumers act as service providers at the same time. They have to purchase and maintain servers and other equipment, and keep the services in a healthy state before they can consume the services. On the cloud platform, service consumers simply acquire new services by subscribing to them, and often follow a pay-as-you-go mode to pay for service consumption without any burdens to maintain the services.

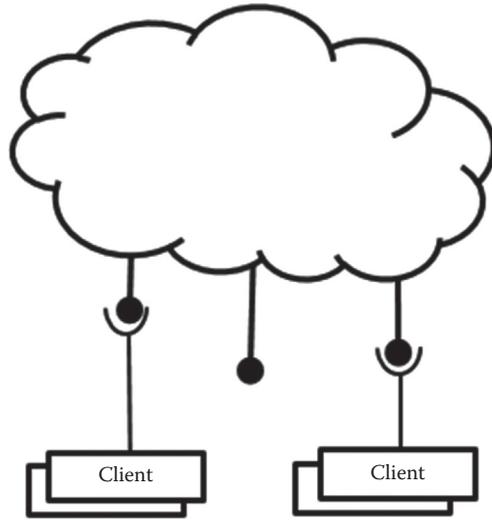


Figure 1.2 Cloud = resource pool + endpoints.

Services running on the cloud are called Cloud Services, which can be categorized into two types: computing and storage, the so-called cloud computing and cloud storage. Of course, if we consider all cloud services as callable functions on the cloud, then all cloud services possess the attribute of “computing.” Therefore, sometimes we do not distinguish the two service types and use “cloud computing” as a generic term to describe both service types.

Note: Service Providers and Service Consumers

Cloud platforms provide necessary services for developers to develop cloud services on them. So within this context, cloud platforms are service providers and service developers are service consumers. However, after service developers have developed and deployed cloud services on the cloud, they become service providers and the end users of the service are service consumers. Readers should pay attention to the different meanings of “service provider” and “service consumer” in different contexts.

In order to fully understand the advantage of cloud computing, we must focus on two main aspects: the agility and the added value. In the following text, we will summarize some of the characteristics of the cloud and see how they help service consumers to reduce cost, increase agility, and gain added values.

1.2.1 Elasticity

A key characteristic of the cloud is elasticity. Elasticity means that service consumers can increase or decrease the subscribed service level at any time per business demand. A user renting storage on the cloud can, for instance, either increase his or her storage from 100G to 100T, or reduce it to 100M at any time. On the other hand, a website provider can switch between one or more

servers to run his or her website as per the site's actual load. Such switches can be done within hours or even minutes.

This kind of flexibility is unreachable for a traditional data center. In a traditional data center, provisioning a server means going through planning, approving, purchasing, installing, testing, tuning, etc.—a long process that usually takes weeks or months. Although server virtualization helps to simplify and speed up the process greatly, enterprises still need complicated processes and close coordination among departments if they want to distribute or reallocate resources to satisfy ever-changing needs. Furthermore, because there are always certain rules to follow when it comes to disposing fixed assets, it is never a small issue to give up a server either. As a result, traditional data centers suffer very often from either too many or too few servers. The ideal allocation of resources to each department is often a very difficult goal to reach. In short, the capacity of a traditional data center is often out of sync with the pace of business development.

The cloud saves us from this situation. Service consumers can get extra storage and processing power from the cloud platform at any time, and they can return the resources that are no longer needed just as easily. Service consumers pay only for what they are actually using and nothing more. The root of elasticity comes from separation of services and underlying infrastructure. With the separation of concerns, a service becomes a pure logical definition that can be replicated and instantiated on any number of available servers (given that the servers satisfy specific constraints imposed by the service definition). Because servers are not permanently bound to particular services, they can be repurposed as needed to provide support to other services.

Elasticity is not only a big help to IT cost reduction, but also a great boost to business agility. Now let us look at two typical scenarios of workload changes and find out how elasticity helps service consumers to control cost efficiently without degrading service levels.

1.2.1.1 On and Off Mode

Under on and off mode, a system stays in either of the two distinctive statuses: active or inactive. In active mode, the system has to support a large number of users, while in inactive mode, the system needs to do almost nothing. A typical example for this kind of system is the online registration system of universities. Before the new semester begins, students are obliged to register for the classes within a given time period. During this period, the system becomes very busy. However, during the semesters, few students would need the system. Such a workload change pattern is depicted in Figure 1.3.

In traditional data centers such systems stay either extremely busy or extremely idle. Users suffer from slow responses or even system crashes during busy periods, while data centers suffer from a big waste of resources during idle periods. Now, with the cloud platform solution, users can rent more servers in busy periods and return the surplus after the peak.

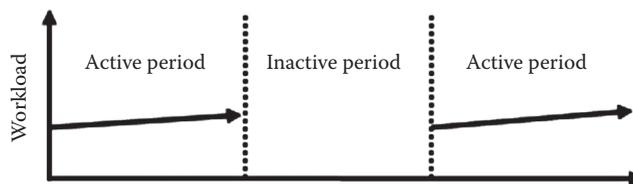


Figure 1.3 Workload changes: on and off mode.

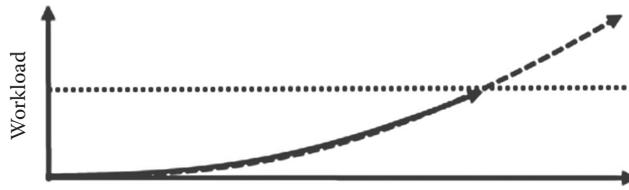


Figure 1.4 Rapid growth of workload.

1.2.1.2 Rapid-Growth Mode

A rapid-growth mode means that the workload of a system increases dramatically during a short period of time. This rapid increase in workload is often not foreseen and thus exceeds the original investment in IT. Therefore, systems cannot meet the demand of the rapidly growing business. Companies generally pursue the rapid growth of their business, especially the start-ups. But if a company staggers due to system restrictions, it would be an unforgivable mistake. Such risks do exist in the traditional data centers. However, the options to mitigate the risk are quite limited: either start-ups are able to make very accurate forecasts or they make large investments in IT in preparation for possible spikes. Obviously, neither can be a viable solution in most cases. Figure 1.4 shows how business cannot reach its deserved level due to system restriction toward workload.

Cloud platform is a good solution for start-ups. They can rent minimum resources to jump start at the beginning and rent more resources when business increases. The pool of resources on the cloud is so vast that it seems limitless for these start-ups. No matter how fast the business grows, cloud platforms can easily satisfy the increasing needs. Utilizing cloud platforms as a powerful backbone, start-ups can start small and stretch out their business to the utmost. This is definitely a smart and effective way of starting new businesses.

There are of course other modes of workload changes. Some workloads vary with seasons, such as those for online retailers; some others have sudden spikes, such as web traffic generated by breaking news. Being able to quickly and efficiently handle fluctuations in workloads is the biggest advantage of using cloud platforms.

Note: In this book, we focus on the technical aspect of the elasticity and the scalability. From the business perspective, the elasticity of the cloud reflects the agility of the company adapting itself to the market changes.

1.2.2 Availability

Simply speaking, the availability of the cloud means that users can access the services hosted on the cloud at anytime, anywhere. In other words, cloud services must be “usable” *almost* at any time and from anywhere. Availability seems simple, but it is actually decided by many related aspects of design, development, and operation of cloud platform and cloud service.

When we say that a platform or a service is available, it implies that this platform or service functions properly. Obviously, if a cloud platform or cloud service cannot maintain a healthy running status, its availability cannot be guaranteed. There are no shortcuts for availability. Generally

speaking, system availability is realized by redundancy and backups. Any cloud platform providers would, for the sake of saving cost, avoid choosing servers with very high configurations but take commodity hardware. Therefore, it is unavoidable to encounter hardware failures. In order to guarantee system availability, large quantities of redundant and automatic backups are configured inside these cloud platforms. For example, a user who subscribes to a Microsoft Azure SQL Database automatically gets two hot backups for each of the servers he acquires. When the main server fails, one of the backup servers will replace it automatically to guarantee system availability. Microsoft Azure thus can guarantee 99.9% availability in the user's databases. The different levels of availability are defined and enforced by Service Level Agreements (SLAs).

Note: About Service Level Agreement

Microsoft Azure provides different SLAs for its services. Availability is generally expressed as a percentage. For example, the availability level of your subscribed service from Microsoft Azure can be shown by the following formula:

$$\text{Availability} = \frac{\text{Total subscribed time} - \text{unavailable time}}{\text{Total subscribed time}}$$

Note that this formula is only a schematic expression. Different services have different methods of calculation and may have certain restrictions. Further information is available at <http://www.windowsazure.com/en-us/support/legal/sla/>.

It might be safe to claim that no software is 100% bug-free. Services on the cloud platform may encounter various problems. Therefore, cloud platform providers need not only to monitor and handle hardware problems, but also to check and tackle software defects. Microsoft Azure provides comprehensive support for telemetry, as well as the ability to automatically recover failed services. Autorecovery is an excellent example of a value-added service provided by Microsoft Azure. Without cloud service developers writing any extra code or performing any additional administrative tasks, cloud services hosted on Microsoft Azure automatically gain the autorecovery capability. In Section II of this book, we further discuss the mechanism of autorecovery.

Before wrapping up this section, we still need to clarify two concepts related to availability: fault domain and upgrade domain.

1.2.2.1 Fault Domain

Fault domain refers to a group of resources that could fail at the same time. For example, a personal computer can be regarded as a fault domain, because all its components—CPU, memory, and hard disk—depend on electricity to work. If power supply fails, all resources stop working at the same time. In a data center, a group of servers on the same rack is a fault domain because they share the same power supply or cooling system. Fault domain is a very important concept in containing errors and providing high availability. Obviously, if you switch off your PC, you would expect other household electronics to continue to work. In other words, failure in one fault domain should not affect other fault domains. Therefore, Microsoft Azure allocates instances of your hosted services into different fault domains to improve service availability. Because it is rare to have two fault domains to stop functioning at the same time in a world-class data center such

as that provided by Microsoft Azure, Microsoft Azure can guarantee high availability when you have multiple instances of your services.

1.2.2.2 *Update Domain*

Update domain is a logical concept. It refers to a group of resources that can be updated simultaneously during system upgrades. When Microsoft Azure updates a service (no matter whether this service belongs to Microsoft Azure or is published by a user), it will not update all resources needed by this service at the same time, but rather by group. This is to guarantee that at least one group of resources remains available to handle users' requests during update. This approach is called rolling upgrades or zero-down time upgrades. Of course, system throughput will be influenced briefly during upgrade, but in general the service remains in the state of "available."

Note: In Microsoft Azure, rolling upgrades are sometimes also called Upgrade Domain Walk because upgrade domains are updated one by one. Microsoft Azure also supports other upgrade approaches, such as switching between different deployments. We will discuss this topic further in Section II.

From the earlier discussion, we can conclude that Microsoft Azure fully supports availability at business level, technical level, and operational level. In the later part of this book, we will also talk about techniques and tools for high-availability design.

1.2.3 *Scalability*

Scalability is an indicator of how well a system can support its users in terms of both quantity and quality.

Quantity indicates the capacity of a system to deal with workload changes. In traditional data centers, this kind of scalability is often achieved by upgrading system hardware. For example, when a database becomes a bottleneck, we can add more memory or CPUs to improve the capacity of the server. The throughput of the system is thus increased to serve more users. This is called scaling up or vertical scaling. The advantage of vertical scaling lies in the easy and low-risk execution because we do not have to modify the code in most of the cases, but change the hardware only. Still, there are two restrictions. First, vertical scaling is not limitless. Any server obviously has limited physical space for memory and storage expansions. It is not possible to improve the capacity of a server without a limit. Virtual servers have more restrictions in this aspect. Generally, cloud providers have virtual machines in several sizes for users to choose. If the highest configuration still cannot meet the demand, we have to apply other scaling methods, such as horizontal scaling. Furthermore, because vertical scaling reconfigures the existing servers, the servers often need to be shut down during upgrades, causing service interruptions.

A common scaling method on the cloud is scaling out, also referred to as horizontal scaling. Unlike vertical scaling, horizontal scaling does not modify the existing configurations of the servers, but adjusts system capacity by increasing or reducing the number of servers. Take a database server as an example, where higher data-processing power can be achieved by adding more servers. For instance, if a database server can handle 10,000 transactions per second, theoretically two servers can handle 20,000 businesses per second. When several servers share the workload of one system, it is called load balancing, which will be further discussed in Section II.

Horizontal scaling provides some advantages over vertical scaling. First, it is not restricted by the hard limit of a server or a virtual machine. When you want to increase the throughput of a system, all you need to do is add more servers. The workload is then distributed equally to all the servers by means of load balancing. Second, it is also easy to reduce the throughput of a system by simply returning the obsolete servers to the cloud platform provider. Users thus avoid unnecessary rental cost. Last but not the least, the process of adding or reducing servers does not affect the running of other servers, avoiding unnecessary service interruptions. However, not all systems can be easily scaled in this way because horizontal scaling has some extra requirements in system structure. We shall further discuss this point in Section II.

In terms of quality, scalability means that a service can maintain an acceptable performance level when dealing with a large number of concurrent users. Users will not tolerate a slow service, especially when there are many alternatives. Therefore, the scalability of a system is closely linked to its performance. Concepts, tips, and tools of system performance are discussed in Section II.

1.3 Microsoft Azure Overview

Microsoft Azure is a flexible, reliable, and open environment for developing and hosting SaaS solutions. We can understand Microsoft Azure from three levels: IaaS, PaaS, and SaaS.

1.3.1 IaaS (Infrastructure as a Service)

IaaS is the hardware provided by a cloud platform provider to run users' applications, including infrastructure (power supply, cooling system, ventilation, etc.), hardware devices (network, rack, storage, servers, etc.), and server virtualization. Users can rent these virtual servers directly from Microsoft Azure. Developers are free from the trifles of managing and maintaining hardware and can concentrate on the design and development of their applications and services.

Renting servers is much cheaper and quicker than setting up servers by yourself. Users can obtain high computing power at a small cost. On the other hand, users can return the obsolete capacity to the provider at any time to keep the cost to a minimum.

Microsoft Azure provides a number of virtual server configurations, as shown in Table 1.1.

Note: The data in Table 1.1 are taken from <http://msdn.microsoft.com/en-us/library/windowsazure/jj156003.aspx>. Available virtual machine choices may be different in your region, and are subject to change over time.

Microsoft Azure also provides various prebuilt images for easy virtual machine creation. At the time of writing this work, available images include the following:

- Windows Server 2012 Datacenter
- Windows Server 2012 R2 Preview
- Windows Server 2008 R2 SP1
- SharePoint Server 2013 Trail
- SQL Server 2014 CTP1 Evaluation On WS 2012
- SQL Server 2014 CTP1 Evaluation On WS 2012 R2

Table 1.1 Microsoft Azure Virtual Machine Choices

<i>Size</i>	<i>Number of CPU Cores</i>	<i>Memory</i>	<i>Temporary Storage</i>	<i>Bandwidth (Mbps)</i>	<i>Maximum Numbers of Data Disks (1 TB Each)</i>
Extra small	Shared	768M	20G	5	1
Small	1	1.75G	70G	100	2
Medium	2	3.5G	135G	200	4
Large	4	7G	285G	400	8
Extra large	8	14G	605G	800	16
A6	4	28G	285G	1000	8
A7	8	56G	605G	2000	16

- SQL Server 2012 SP1 Enterprise On Win2012
- SQL Server 2012 SP1 Enterprise On Win2K8R2
- SQL Server 2012 SP1 Standard On Win2012
- SQL Server 2012 SP1 Standard On Win2K8R2
- SQL Server 2008 R2 SP2 Enterprise On Win2K8R2
- SQL Server 2008 R2 SP2 Standard On Win2K8R2
- SQL Server 2008 R2 SP2 Web On Win2k8R2
- BizTalk Server 2013 Enterprise
- BizTalk Server 2013 Evaluation
- BizTalk Server 2013 Standard
- Visual Studio Ultimate 2013 Preview
- openSUSE 12.3
- SUSE Linux Enterprise Server 11 SP2
- SUSE Linux Enterprise Server 11 SP3
- Ubuntu Server 12.04 LTS
- Ubuntu Server 12.10
- Ubuntu Server 13.04
- OpenLogic CentOS 6.3

From this list, we see that Microsoft Azure provides virtual machine images based not only on Windows systems but also on popular Linux variations. For Linux images, you have the same level of technical support as you get for Windows-based systems. This shows that Microsoft Azure is a real open system.

1.3.2 PaaS (Platform as a Service)

Built on IaaS, PaaS provides a software environment for service developers to develop and run their applications on the cloud. PaaS hides infrastructural details from the application developers, so that they can better concentrate on the development of their business logics. IaaS users still have to manage their virtual servers, such as upgrade operation systems and install patches. On the contrary, PaaS users do not directly work on virtual servers. They work at a higher, abstract

service level. They define their services and specify the kind of environment the services need, but they do not actually manage the environment, which is taken care of by the cloud platform. For example, a service developer can specify that the service needs two small Windows Server 2008 R2 servers to run. When this service is deployed, Microsoft Azure will build and maintain two virtual servers according to this abstract definition. Microsoft Azure manages every aspect of these virtual machines, including the following:

- Installing an operating system and building up the necessary environment to host the service
- Updating the operating system and installing security patches
- Managing network configurations such as mapping ports and setting up firewall rules
- Monitoring the health of virtual servers and performing autorecovery when necessary

PaaS separates cloud services from underlying infrastructure. This is a very important abstraction layer that enables cloud services to be efficiently managed by Microsoft Azure. Service developers do not need to care about how and where these virtual machines are provisioned and managed. Coming back to the previous example, this application needs two virtual servers. Let us suppose that the physical server on which one virtual server is hosted breaks down; Microsoft Azure will search for another healthy physical server and build up a new virtual server on it, and then transfer the application to the new server. This process is totally transparent to the service developer.

Microsoft Azure provides two platform services for cloud service developers: Microsoft Azure Websites and Microsoft Azure Cloud Service. Users can create and run websites very quickly by using Microsoft Azure Websites, while Microsoft Azure Cloud Service supports the development and operation of all kinds of cloud services. These two services will be explained in detail in Sections II and III. Why does Microsoft provide these two different platform services? From the cloud computing viewpoint, a website is nothing but a special kind of cloud service, which exposes an endpoint based on HTTP/HTTPS to its clients (usually browsers) to access its functionalities (to get HTML responses). Microsoft Azure Websites is an optimized platform for this special kind of cloud service. On the other hand, Microsoft Azure Cloud Service is the general platform for developing and operating cloud services. In other words, both Microsoft Azure Websites and Microsoft Azure Cloud Service are PaaS offerings, as they both hide infrastructural details from developers. It is just that Microsoft Azure Websites is optimized for websites, which is one type of cloud service.

Note: Website is a kind of cloud service.

1.3.3 SaaS (Software as a Service)

SaaS means that software is provided to end users as a hosted service. End users do not have to install or maintain any hardware or software environment to support the service. Instead, they simply access required functionalities via endpoints provided by service providers. Among these services, some are provided by Microsoft Azure, such as storage service, SQL Database, caching service, and access control service. Other services are provided by third parties such as MongoDB, MySQL, and SendGrid. More importantly, cloud service developers can also host their services

on Microsoft Azure and provide these services to end users in the form of SaaS. To a great extent, the ultimate goal of Microsoft Azure is to facilitate cloud service developers to develop and host their cloud services on it. Design, development, and operation of SaaS are the main focus of this book.

Microsoft Azure provides cloud service developers with many building block services. These building blocks are provided to developers and users in the form of SaaS. These services can be accessed not only by using client libraries, but also by using REST-styled calls. We will explain many of these services in detail later in this book. Here we provide a quick overview of each of these services:

■ SQL Database

SQL Database can be simply considered as “SQL Server on the Cloud.” Users can create and use SQL Database instances on Microsoft Azure at any time without needing to acquire and manage any physical database servers. Microsoft Azure SQL Database is highly compatible with Microsoft SQL Server as they both use the same Tabular Data Stream (TDS) protocol. Many existing SQL Server databases can be directly migrated to SQL Database. In addition, SQL Server data-accessing techniques such as ADO.Net and Entity Framework are applicable to Microsoft Azure SQL Database as well. Of course, there are still some differences between SQL Server and SQL Database. For example, SQL Database supports data sharing, which reflects the common scaling-out philosophy of the cloud.

SQL Database will be further discussed in Chapter 5.

■ Storage Services

Microsoft Azure provides a rich set of NoSQL storage, including table storage, BLOB storage, queue storage, and virtual drives. A table is used to store large amounts of unstructured data; BLOB is used to store large amounts of text or binary data such as video, audio, and images; Queue Storage provides message queue service; and virtual drives simulate NTFS disk volumes for cloud services.

Storage functions are further discussed in Chapter 6.

■ Caching Service

Microsoft Azure has two flavors of Cache Services: Role-Based Caching, and Microsoft Azure Caching. Role-Based Caching clusters are self-hosted by cloud services. Because the cache clusters are collocated with cloud service roles, they can provide optimum performance with minimum overheads. Microsoft Azure Caching is a hosted service, so you do not need to host the cache clusters yourself. At the same time, You subscribed Microsoft Azure Caching clusters are dedicated to your services; hence, you have more control over cluster behavior and throughput.

We discuss the concept of roles in Chapter 3, and introduce caching service in Chapter 6.

■ Service Bus

Service Bus provides various message-based services for system integration, loose coupling, hybrid cloud, and push notifications, including message queues, topics and subscriptions, notification hubs, and relayed connections.

Different functionalities of Service Bus are covered in Chapters 4, 6, 9, 10, 11, and 15.

■ Mobile Service

Mobile Service provides a series of common services required by mobile applications, such as data storage, scheduled tasks, and push notifications. It provides a turnkey solution for

mobile application developers so that they can focus on building up unique values of their applications without needing to dig into details of backend services.

We discuss Mobile Service in Chapter 8.

- **Media Service**

Media Service provides comprehensive support for applications to ingest, encode, manage, and share media. Users can upload media documents to Media Service, which encodes, encrypts, and broadcasts media streams to end users. Due to the space restriction of this book, no further discussions will be made on this topic.

- **Workflow Service**

Built on Windows Workflow Foundation (WF), this service allows cloud service developers to create, execute, and manage workflows on Microsoft Azure. No further explanation will be made on this topic due to the space restriction of this book.

- **HDInsight**

HDInsight provides the capability of managing dynamic Apache Hadoop clusters to handle Big Data scenarios on Microsoft Azure. Big Data is currently a hot topic and is pursued and admired widely. However, it is too big a topic to be fully covered by this book.

- **Active Directory Access Control**

Active Directory Access Control Service provides strong support for authentication and authorization. Not only can you project users and groups in local Active Directory trees to directory tenants on cloud, but also integrate with other identity providers such as Windows Live, Google, and Yahoo via standard protocols like ws-Federation and OAuth. In addition, because Microsoft Office 365 also uses Microsoft Azure Active Directory for authentication, if your services use Microsoft Azure Active Directory as well, you can potentially tap into the huge Office 365 user base via Single Sign-On.

Active Directory Access control is discussed in Chapter 12.

- **BizTalk Service**

Microsoft's BizTalk is now available on Microsoft Azure as a service. BizTalk provides a middleware for system integration. It can meet the requirements of complex system integration scenarios. No further explanation will be made on this topic due to the space restriction of this book.

Note: Some documentation and web articles categorize all these services as PaaS because they are part of the Microsoft Azure “platform.” This kind of categorization mixes up the general meaning of “platform” and the special meaning of “platform” in PaaS. According to the NIST definition of cloud computing, only services related to cloud service delivery and deployment can be categorized as PaaS. So, services such as BizTalk Service and Media Service should be categorized as SaaS.

On a different note, all listed services are available at the time of writing this book. Obviously, this list will grow steadily as time passes by.

1.3.4 Cost Calculation

Costs of using services on Microsoft Azure are rather low and some of the services are free of charge. For cost estimation and calculation, the best place to start is by using the Microsoft Azure online cost calculator at <http://www.windowsazure.com/en-us/pricing/calculator/?scenario=full>.

1.4 Preparing the Development Environment for Microsoft Azure

Microsoft provides a comprehensive, efficient, and consistent development environment for cloud service development on Microsoft Azure. For readers who are familiar with Visual Studio and .Net Framework, it is quite easy to work with Microsoft Azure. For developers using other languages, Microsoft has also prepared corresponding SDKs. You will need three things to start your application development on Microsoft Azure:

- A Microsoft Azure subscription
- A Microsoft Azure SDK
- An integrated development environment (IDE) such as Visual Studio

Note: Microsoft Azure SDK supports Visual Studio 2010 SP1, Visual Studio 2012, Visual Studio 2013, and higher, including the free express editions. If you have not purchased Visual Studio, you can download the free Visual Studio 2012 Express for Web from the official Microsoft site to complete exercises in this book.

1.4.1 *Subscribe to Microsoft Azure*

Before you can deploy your cloud services to Microsoft Azure and use various Microsoft Azure services, you need to create a Microsoft Azure subscription. Microsoft Azure offers free subscriptions for you to try out all services. In addition, if you are an MSDN Professional, Premium, or Ultimate subscriber, you get up to \$150 credits per month to be used on Microsoft Azure services. To create a new Microsoft Azure subscription, go to <http://www.windowsazure.com> and click on the free trial link at the upper-right corner of the home page. You'll need a free Microsoft account (previously known as Windows Live ID), which will become your account administrator.

1.4.2 *Install Software Development Kit*

Microsoft has SDKs for the following programming languages:

- .Net
- Node.js
- PHP
- Java
- Ruby
- Python

In addition, Microsoft also provides development kits for Mobile Devices and Media Services. Readers can download them for free at <http://www.windowsazure.com/en-us/downloads>.

1.5 Introduction of Microsoft Azure Management Portal

Before closing this chapter, let us spend some time to get familiar with Microsoft Azure Management Portal. The interface of this portal is built using standard HTML, which can be

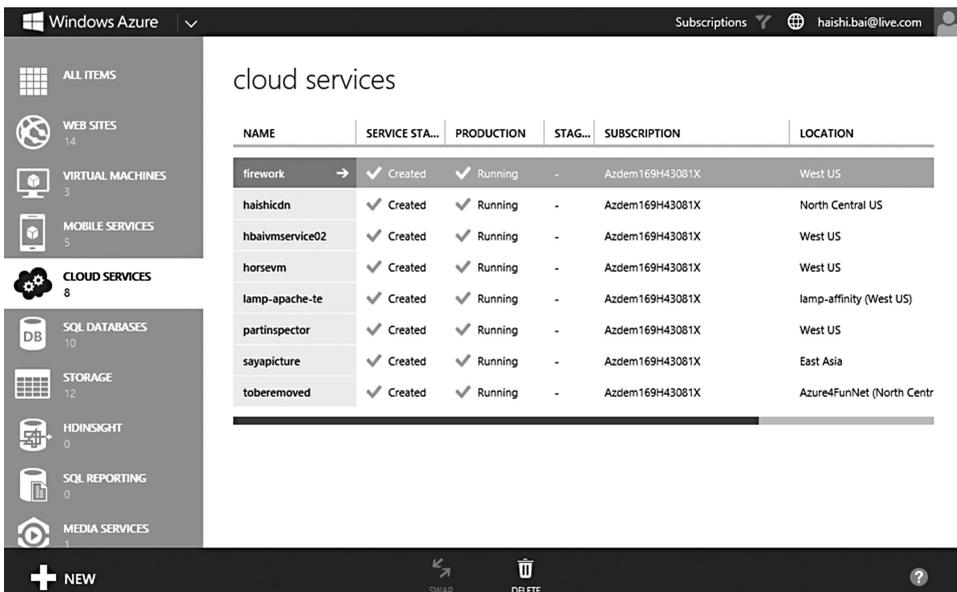


Figure 1.5 Microsoft Azure Management Portal.

viewed on all major explorers and devices. The UI is simple and clear and is quite easy to work with (Figure 1.5).

1.5.1 Sign In

Before using the management portal, you need to sign in to Microsoft Azure with your subscriber account or a co-admin account. At the time of writing this book, either a Microsoft account (original Windows Live ID) or a Microsoft Azure Active Directory account could be used to sign in. For individual users, it is most likely you are using a Microsoft account. Enterprise subscribers may contact their subscription administrator for sign in information. The sign in process is very simple—navigate to <https://manage.windowsazure.com/> and sign in when prompted.

1.5.2 Page Layout

The portal's pages consist of the following components:

- **Navigation pane**
The navigation pane is on the left side of the screen. Except for the “All items” icon on the top, each icon represents a type of asset on Microsoft Azure. Users can navigate to different assets using the navigation pane.
- **Command bar**
The command bar is at the bottom of the screen. The “new” icon at the lower-left corner provides a unified way of creating a new item. If you want to create anything, just click on the “new” icon. The middle part of the command bar is a list of icons that changes with context, representing the operations supported by currently selected items. Finally, the question mark icon on the lower-right corner is for online help.

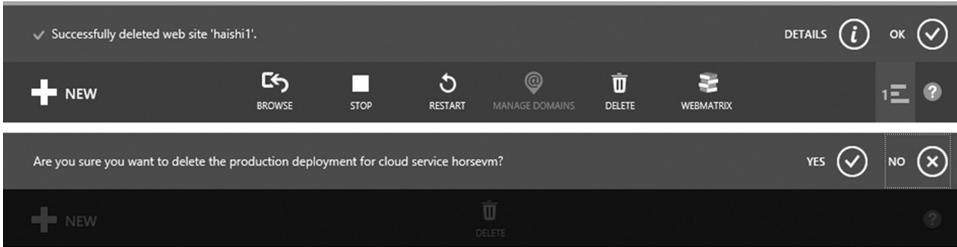


Figure 1.6 Status bar examples.

■ Status bar

The status bar is at the bottom of the screen, above the command bar. The status bar shows dynamic information such as reminders, confirmations, operation progress, and results. Please see the two examples in Figure 1.6.

Tip: Microsoft Azure Management Portal allows multiple operations to run at the same time; therefore, there could be multiple messages appearing in the status bar. For such cases, there will be a number shown at the lower-right corner of the screen to indicate the number of messages. As shown in Figure 1.7, there are two messages.

Users can dismiss a message by clicking on the OK icon on the right side of the message. Messages for completed operations can be dismissed together by clicking on the DISMISS COMPLETED link on the top of the status bar. Note that this menu does not dismiss error messages.

■ Title bar

The title bar is displayed on top of the screen and shows subscription information and current user information. Users can change languages by using the globe icon. When users have several Microsoft Azure subscriptions, they can use the *Subscriptions* menu on the title bar to filter user assets by subscriptions. Asset management will be explained in Chapter 4.

■ Main display area

The rest of the page is the main display area. This area has two common display modes: list and details. Figure 1.8 is an example of the list view. Clicking on any item in this list takes you to the detailed view of the item.

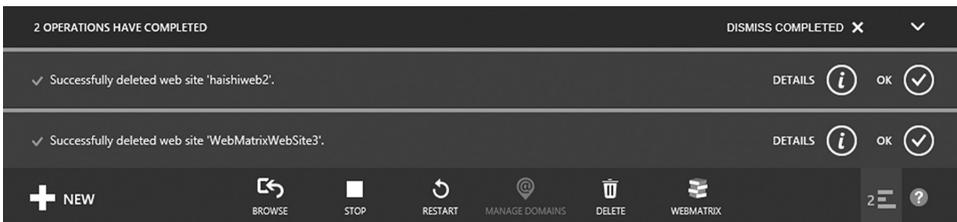


Figure 1.7 Managing multiple status messages.

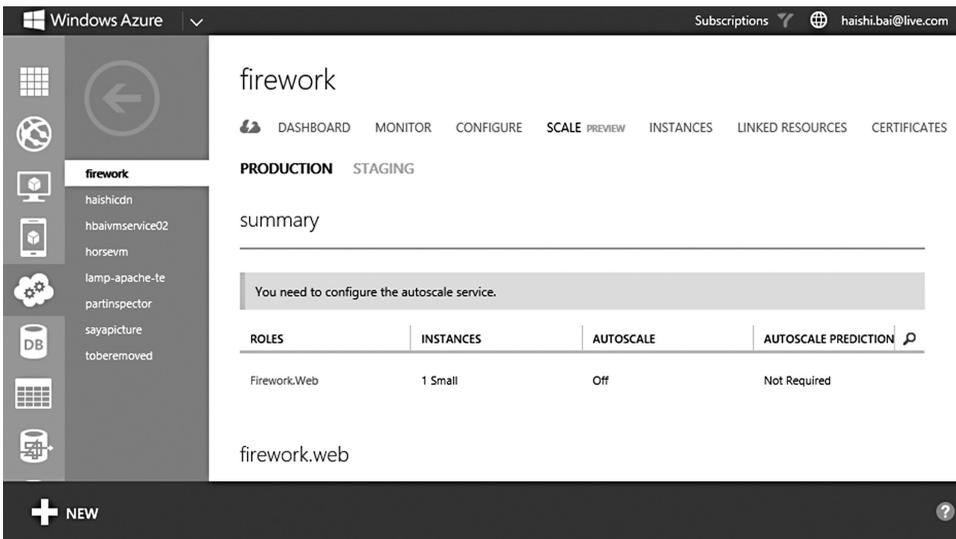


Figure 1.8 Details view.

Two things happen when you click on an item. First, the navigation pane collapses into a list of icons. Second, items are displayed as a vertical name list to the right of the collapsed navigation pane. In the main display area, from top to bottom are item title, tabs, and different display segments. Because most of the pages share the same layout, it is very easy to get familiar with other pages once you've learned how to use one page.

1.6 Summary

In this chapter, we reflected on the history of cloud computing and explained some of its basic concepts. We hope that by reading this chapter, you will get an overview of the essential characteristics of cloud computing. You do not have to fully understand the concepts mentioned in this chapter immediately, as long as you get a general understanding of them. In the following chapters, we will further discuss these concepts in more detail. We will also learn how to prepare for Microsoft Azure development and how to use Microsoft Azure Management Portal. Starting from the next chapter, we will get in touch with the developing process of the cloud services on Microsoft Azure. We propose you complete the exercises as you read through the texts to gain first-hand experience in cloud service development.

Chapter 2

Building Websites on the Cloud

2.1 Microsoft Azure Websites

In this part, we build a simple website on Microsoft Azure as the first example of using Microsoft Azure. There are several options to build and host websites on Microsoft Azure. Here we will take the most direct one, which is to use Microsoft Azure Websites. Azure Websites allows users to build highly scalable websites with minimum initial investments. For example, when you start to build a website on Microsoft Azure, you can choose the free hosting mode to begin with and then increase its capacity as site traffic grows. Websites on Microsoft Azure run in one of the most advanced data centers in the world, which give you unprecedented advantages in reliability, availability, efficiency, economy, ease of use, and rich functionalities. Now let us build up a simple website: Hello, Microsoft Azure.

Example 2.1: Website—Hello, Microsoft Azure

Difficulty: *

1. Log in to Microsoft Azure Management Portal.
2. Click **WEBSITES** on navigation pane.
3. Click **NEW** icon on the command bar.
4. Select **COMPUTE**→**WEBSITE**→**QUICK CREATE**. Then, enter the URL prefix for the site (all websites belong to *azurewebsites.net* domain) (Figure 2.1).

Note: The URL prefix must be globally unique and contain between 2 and 60 characters. It can only contain letters, numbers, and hyphens. The first and the last characters must be a letter or number. The first and the last characters cannot be hyphens.

5. Click **CREATE WEBSITE** to complete the operation.

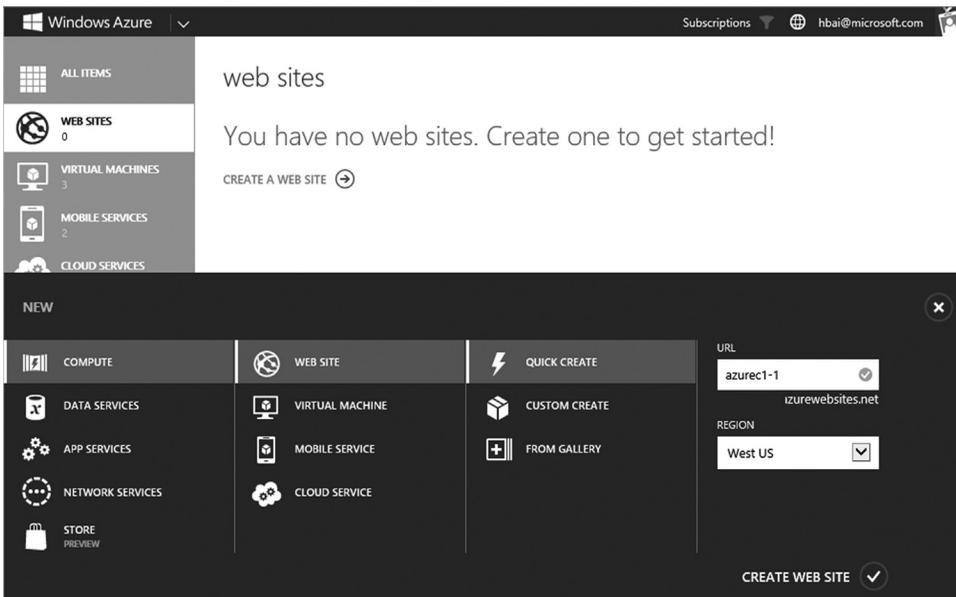


Figure 2.1 Creating a new website.

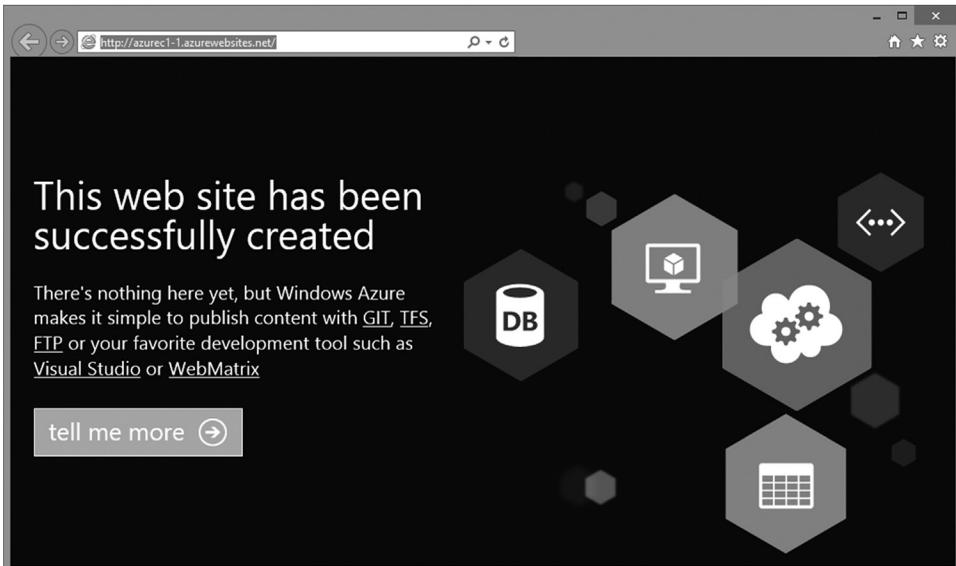


Figure 2.2 The default page of your first website.

6. The website will be built within seconds. You can now use a browser to browse the URL. Congratulations! You've successfully created and deployed your first website on Microsoft Azure. Isn't it magical! Now your website is live to users around the world (Figure 2.2).
7. Next we will use Visual Studio to create a new ASP.NET website and replace the default website. But before we launch Visual Studio, let us first download the publish profile of the website.

Note: About the publish profile

The publish profile is an automatically generated XML file that contains varied information necessary for publishing a website, such as user credential and website address. You can import the publish profile to Visual Studio or WebMatrix so that you do not need to repeatedly enter this information while publishing your sites.

8. In the Microsoft Azure Management Portal, click on the name of the website (as indicated by the arrow in Figure 2.3) to open its details page.
9. Click on **Download the publish profile** link to download the profile. Save it to your local disk (Figure 2.4).
10. Launch Visual Studio. Create a new ASP.NET Empty Web Application (Figure 2.5).

Note: In this example, we have chosen to create a new web application. If you want to publish the existing ASP.NET web application on Azure Websites, you can choose the existing website project and skip the next step (step 11). Be aware that not all ASP.NET websites can directly run problem-free on Microsoft Azure (see Section 2.4 for more details).

11. Add a new default.html file to the website and enter the following code (Code List 2.1):
12. In Solution Explorer, right-click on the solution and select the **Publish** menu (Figure 2.6).
13. In the Publish Web dialog, click the Import button, then select the publish profile you downloaded in step 9 (Figure 2.7).

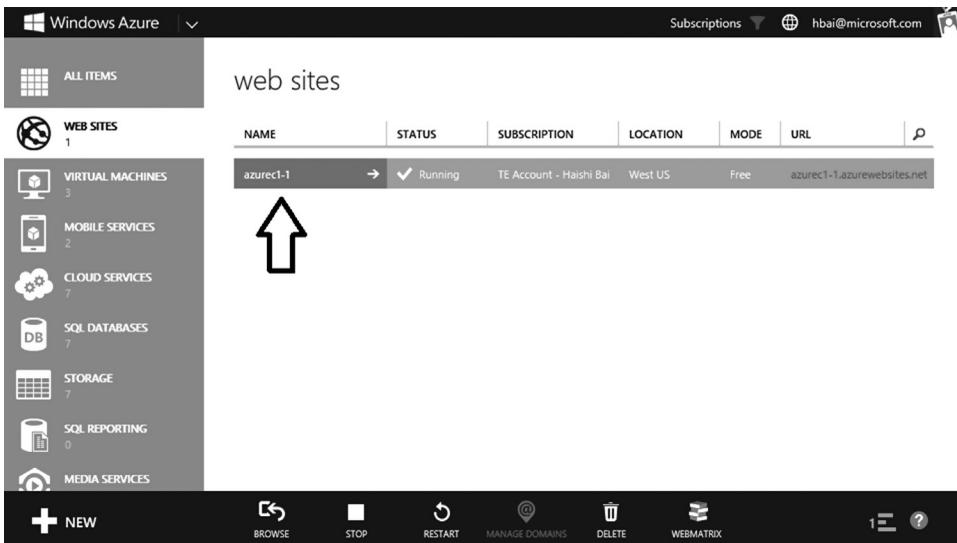


Figure 2.3 Website list on management portal.

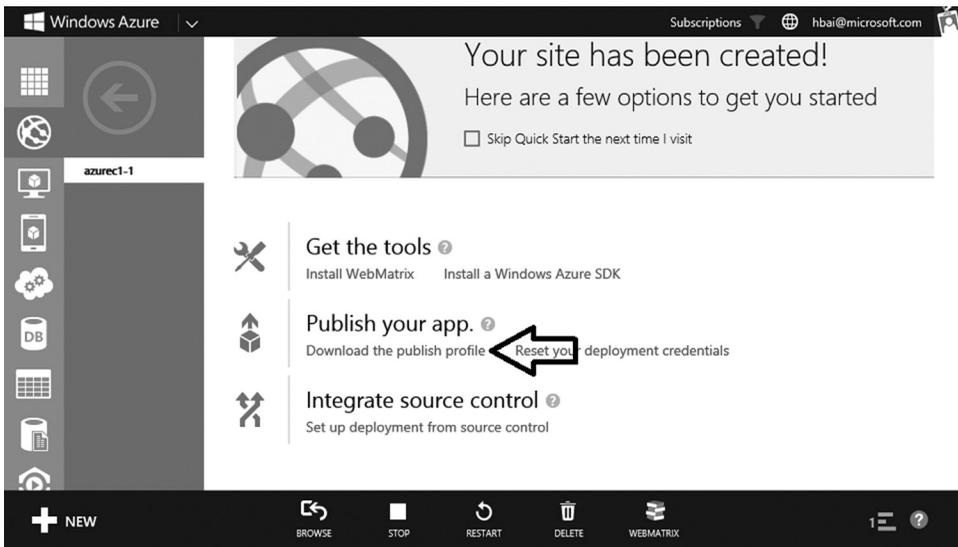


Figure 2.4 Download publish profile.

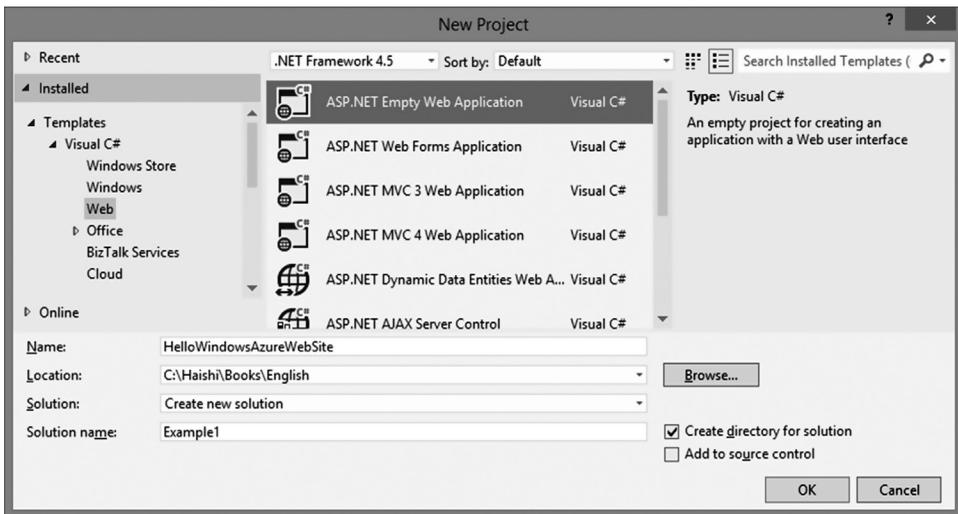


Figure 2.5 Creating a new ASP.NET website project.

14. After the publish profile is successfully imported, you will find all the information needed for publishing automatically uploaded. All you need to do now is to click on the **Publish** button to publish the website (Figure 2.8).
15. The publishing process takes only seconds. Once publishing is complete, your new website will be successfully deployed (Figure 2.9).

Note: Due to browser caching, you may need to refresh your browser to see the new page.

CODE LIST 2.1 HELLO, MICROSOFT AZURE WEBSITES

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title></title>
</head>
<body>
  <h1>Hello, Microsoft Azure Web Sites!</h1>
</body>
</html>

```

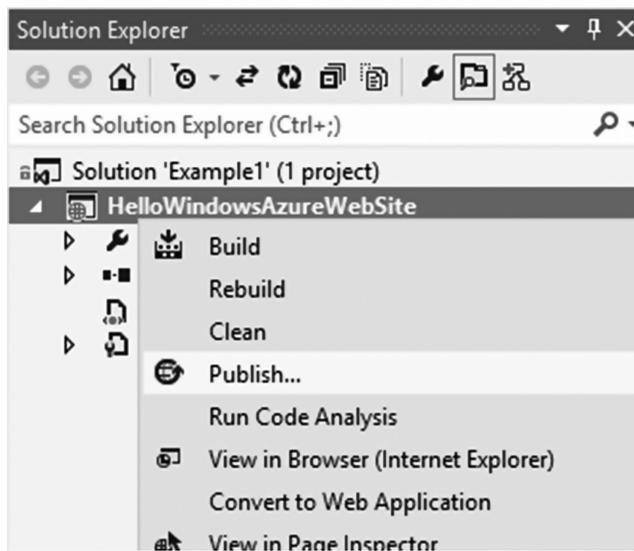


Figure 2.6 Publish menu in Visual Studio.

Note: With Microsoft Azure SDK 2.3, you can directly publish to an existing website from Visual Studio without manually importing the publish profile first. You can also publish your web site to a new or existing virtual machine.

2.2 Website Deployment and Upgrade

We have just created and deployed our first website on Microsoft Azure. Besides deploying via Visual Studio, Azure Websites also supports other tools and approaches to deploy websites, such as WebMatrix, FTP, and integration with source control systems. Next, let us explore these different approaches.

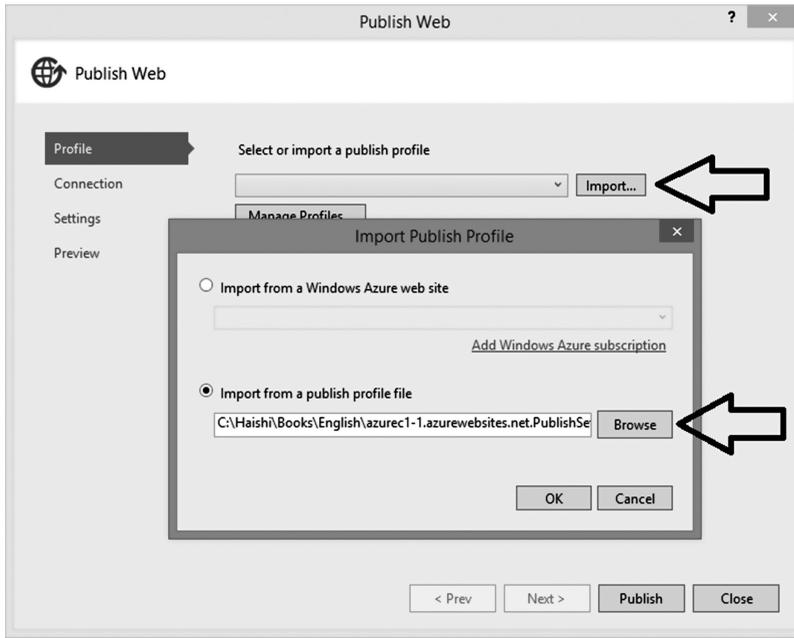


Figure 2.7 Import publish profile.

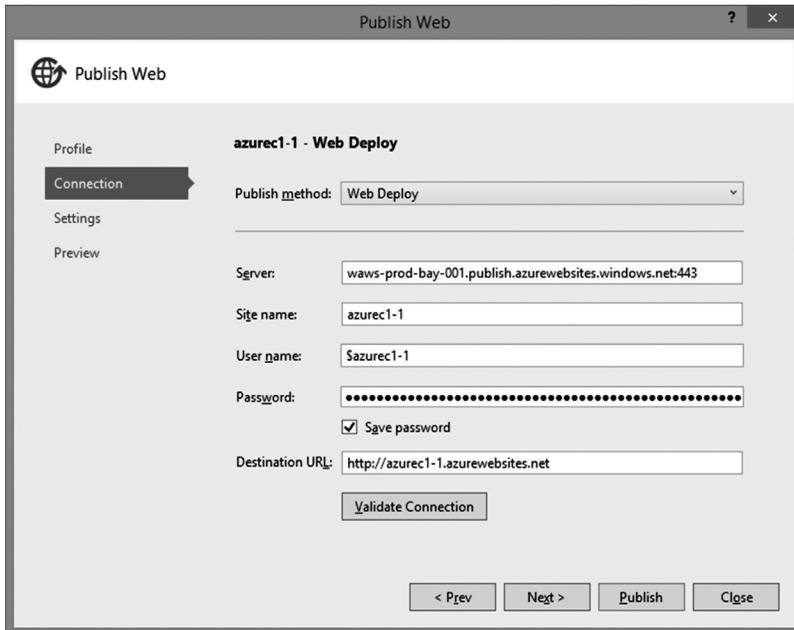


Figure 2.8 Publish website.



Figure 2.9 Hello, Microsoft Azure Websites!

Example 2.2: Use WebMatrix to Update Websites

Difficulty: *

WebMatrix is a free and handy website-developing environment provided by Microsoft. Azure Websites provides built-in support of WebMatrix. We cannot provide a thorough introduction of Web Matrix here due to restriction of space. Instead, we will use a simple example to demonstrate the tool. If you have no plan of using WebMatrix, you can skip this example.

1. Log in to Microsoft Azure Management Portal.
2. Select **Websites** on the navigation pane.
3. From the website list, select the website we created in Example 2.1. Then click the **WEBMATRIX** icon on the command bar (Figure 2.10).
4. If you do not have WebMatrix installed, the system will guide you through the installation process to get it installed. Then WebMatrix will start with your web project loaded. Click on the **Edit live site directly** link to edit the live site (Figure 2.11).
5. Double-click default.html to open the file (arrow 1 in Figure 2.12). Then, change some texts on the page—"Version 2" is appended to the original text (arrow 2). Finally, click the save button to update the live site (arrow 3).
6. Refresh the browser; you can see that the website has already been updated (Figure 2.13).

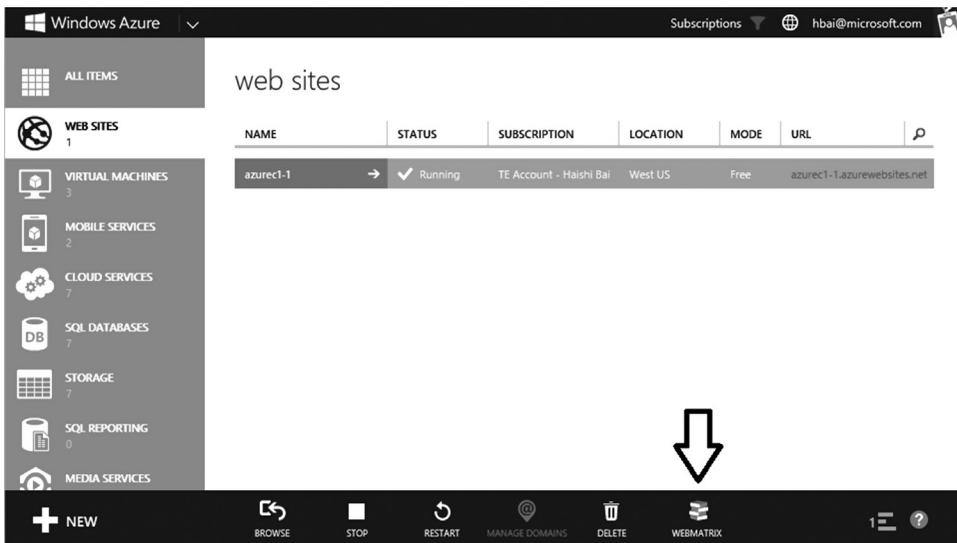


Figure 2.10 WEBMATRIX icon on command bar.

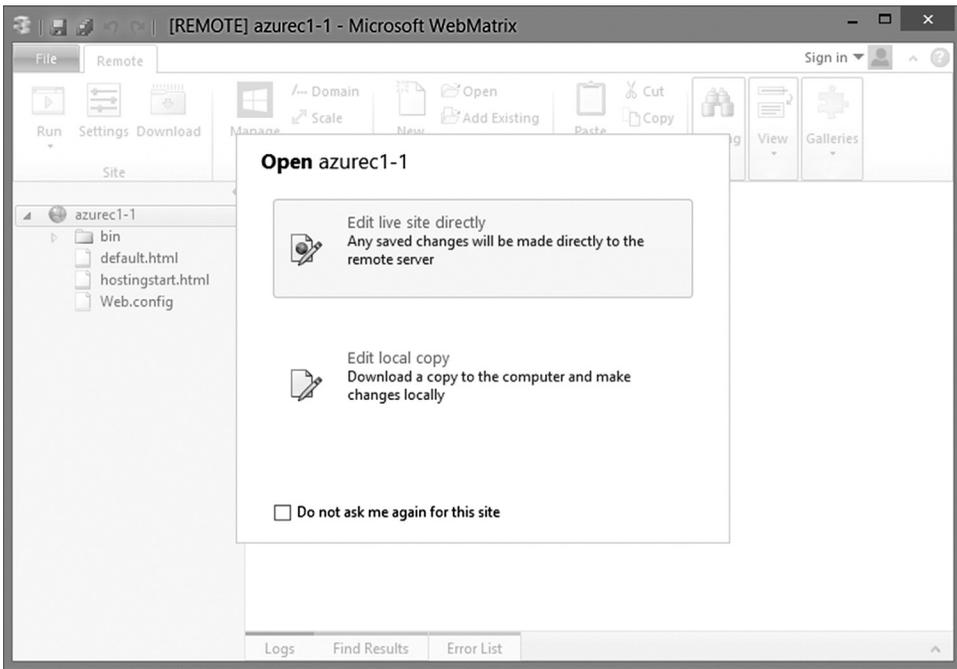


Figure 2.11 WebMatrix with website loaded.

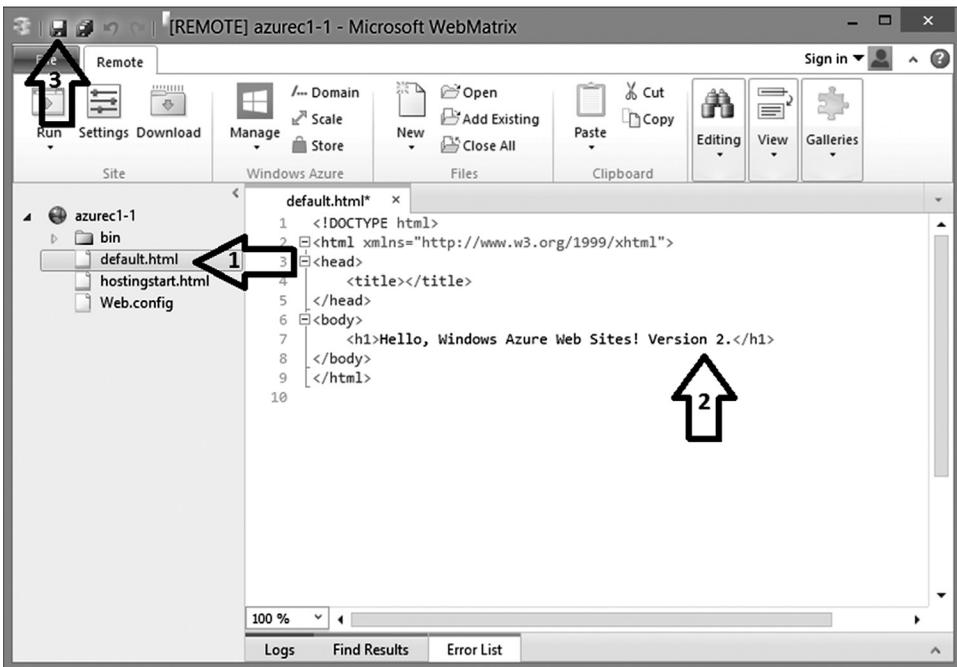


Figure 2.12 Edit website in WebMatrix.



Figure 2.13 Second version of the website.

Of course, this only gives a quick glance to WebMatrix. WebMatrix has many powerful functions, such as CSS editor, data management, template management. The reader may consult WebMatrix documentation for more information if interested.

Example 2.3: Use FTP to Deploy and Update a PHP Website

Difficulty: *

Azure Websites also supports the deployment via FTP. In this example, we will use PHP to create a new website and deploy it to Microsoft Azure by using FTP.

1. Create a new Microsoft Azure website.

Note: For steps to create a website, please see Example 2.1, steps 1–6.

2. Open the website's details page. Then click on the **Reset your deployment credentials** link (Figure 2.14).

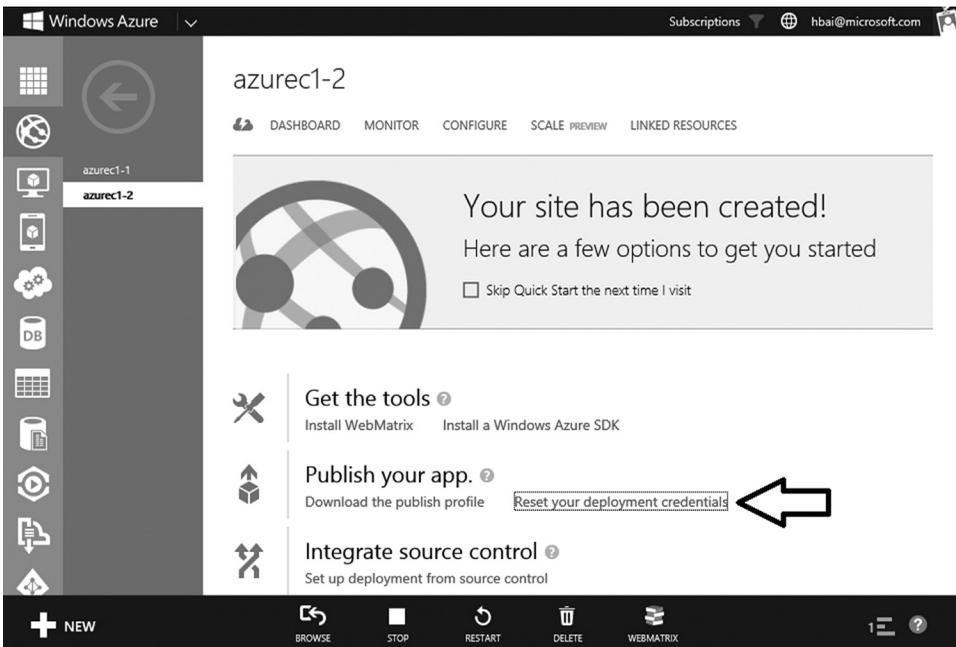


Figure 2.14 Reset deployment credentials link.

New user name and password

Git and FTP cannot use your Windows account to authenticate, so this dialog lets you specify a user name and password that can be used when using those technologies.

This user name and password can be used to deploy to any web site in your subscription. You do not need to set credentials for every web site that you create.

USER NAME

NEW PASSWORD

CONFIRM PASSWORD

Figure 2.15 Set up FTP user name and password.

3. Git cannot use your Windows account to authenticate, so you need to specify a user name and password for FTP deployment (Figure 2.15).
4. Click on the **DASHBOARD** link to return to the website's dashboard (Figure 2.16).
5. Scroll down and you will see the website's FTP host name and user name (Figure 2.17).
6. Create a new folder, and add a new **index.php** page to the folder.

```
<?php
    echo "Hello, PHP web site!";
?>
```

7. Use an FTP client to upload **index.php** to **site\wwwroot** directory of the FTP site.
8. Now you can use `http://[website name].azurewebsites.net` to access the page (Figure 2.18).

Note: PHP Support on Microsoft Azure

Microsoft Azure websites support both PHP versions 5.3 and 5.4, which you can specify on a website's **CONFIGURE** page (Figure 2.19).

In addition, Microsoft Azure also provides a PHP SDK for PHP developers. You can download the SDK from Microsoft Azure PHP Developer Center: <http://www.windowsazure.com/en-us/develop/php/>.

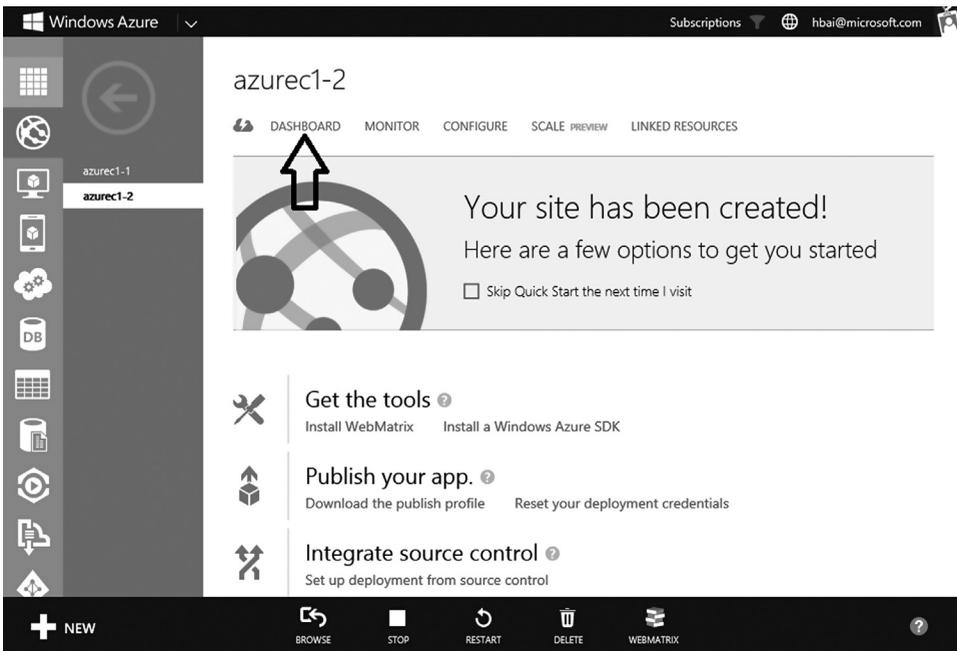


Figure 2.16 DASHBOARD link.

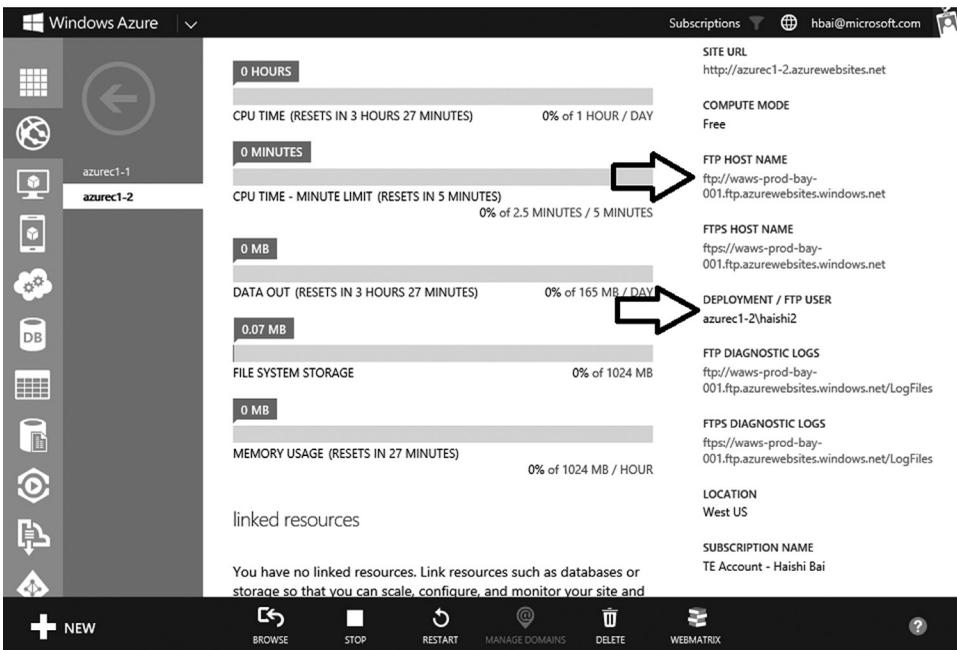


Figure 2.17 FTP host name and user name.

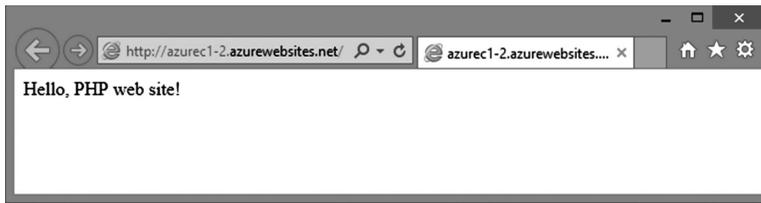


Figure 2.18 PHP website.

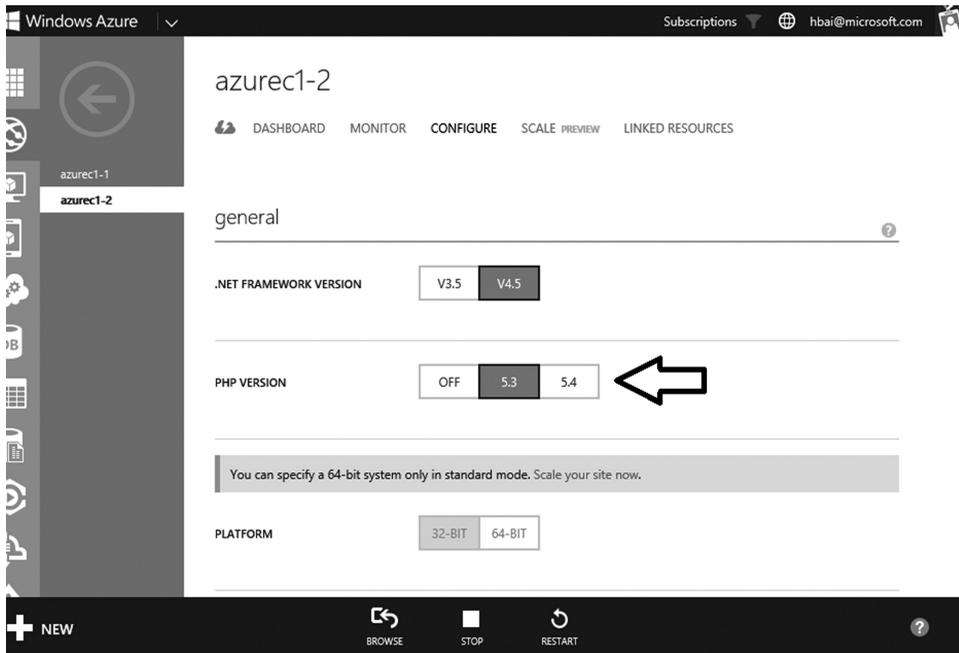


Figure 2.19 Select PHP version.

2.3 Integration with Source Control Systems

WAWS supports integration with many source control systems:

- **Team Foundation Service**
Microsoft's Team Foundation Service is the SaaS version of Microsoft's Team Foundation Server. It provides a comprehensive solution for project management, source control, problem tracking, automated testing, etc.
- **CodePlex**
CodePlex is Microsoft's free site for hosting open-source projects. You can create new open-source projects on CodePlex, or join thousands of existing projects.
- **Git and GitHub**
Git is a distributed source code management system that has become popular in recent years. Many products and tools of Microsoft fully support Git.

- Dropbox

Dropbox is a service that enables you to carry all your photos, documents, and videos with you and enjoy them whenever and wherever you want to. Microsoft Azure supports efficient synchronization and deployment of the source code in Dropbox folders.

- BitBuckets

BitBucket is a hosting site for distributed version control systems (DVCS) such as Git and Mercurial. It provides services such as problem tracking, wiki, and integration with other popular services such as Basecamp, Flowdock, and Twitter.

We discuss project management in Section IV of this book. Here we demonstrate Git integration with a simple example.

Example 2.4: Use Git to Deploy and Update a Website

Difficulty: **

This example is a continuation of Example 2.3; therefore, you will need to complete Example 2.3 before starting with this exercise.

1. Open the details page of the website (if you do not see the page in Figure 2.20, click on the icon of the blue cloud with a flash under your website name). Then click the **Set up deployment from source control** link.
2. In the **SET UP DEPLOYMENT** dialog, select the source control service you want to use. Here select **Local Git repository** (Figure 2.21).
3. It only takes seconds to provision the Git repository. Then, you will see a confirmation page, where you can get the URL to your new Git repository (Figure 2.22).
4. Open Git's command prompt and navigate to the directory containing `index.php`; then submit the `index.php` to the source code repository by using the following Git commands:

```
git init
git add.
git commit -m "initial commit"
git remote add azure https://haishi2@arec1-2.scm.azurewebsites.net/
aurec1-2.git
git push azure master
```

Note: The URL in these commands (highlighted) needs to be replaced by the URL you obtain in the preceding step. You will notice some changes in the details page (see Figure 2.23). As soon as you push your changes to the remote repository, the code is deployed to your production environment. Because of the simplicity and agility of the process, direct modification of the production environment is widely used by many service developers, especially in small start-ups. However, you do run the risk of interrupting your service by accidentally committing a bad version. We will discuss how to ensure check-in qualities in Section IV of this book.

5. Now, let us open the `index.php` page and add one line to invoke the `phpinfo()` method:

```
<?php
    echo "Hello, PHP web site!";
    phpinfo();
?>
```

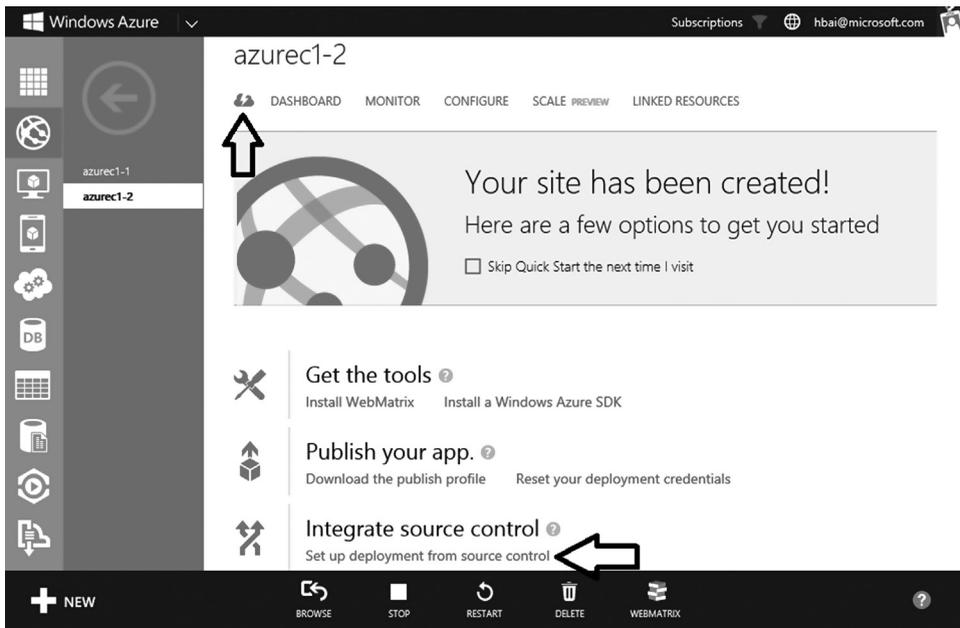


Figure 2.20 Website's details page.



Figure 2.21 Select Git repository service.

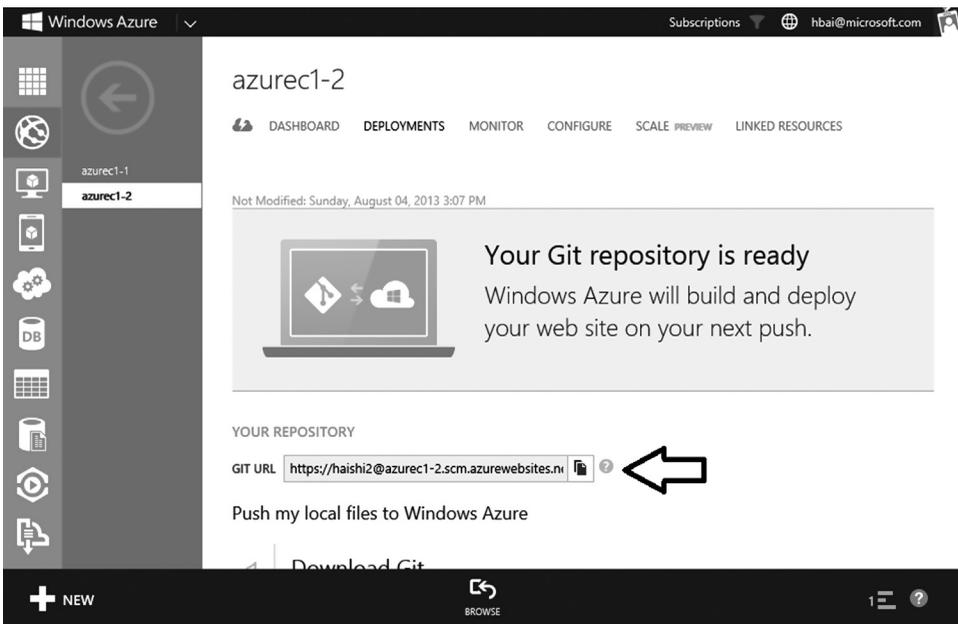


Figure 2.22 Git repository URL.

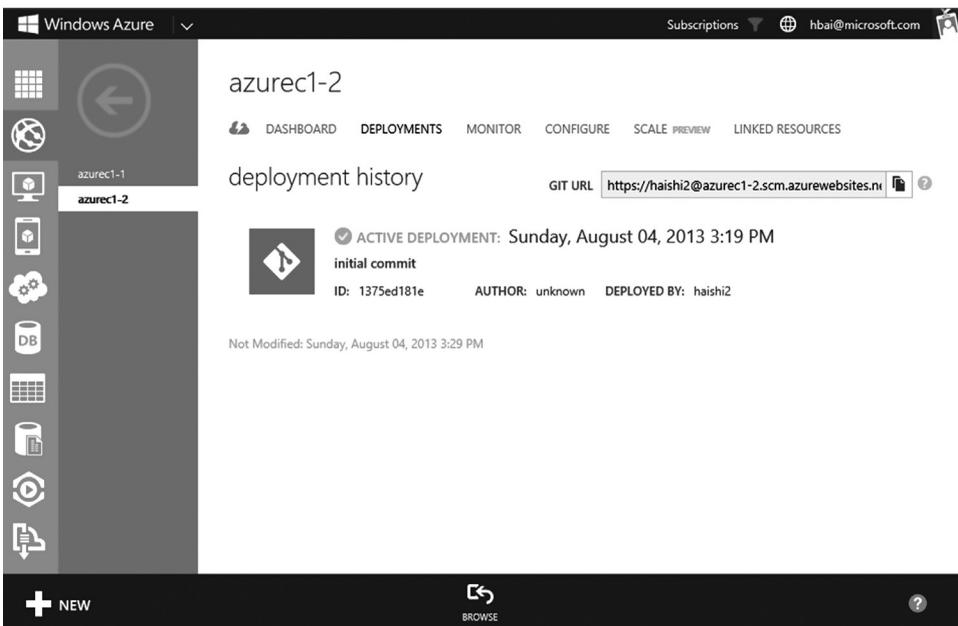


Figure 2.23 Website deployment via Git.

6. Save the file. Then use Git to commit the new version:

```
git add.
git commit -m "version 2"
git push azure master
```

7. After the second version is deployed, you will see two deployments recorded in the history (Figure 2.24).
8. Now, click the BROWSE icon on the command bar to browse the updated website (Figure 2.25).

Note: When you have multiple developments, you can switch between different versions at any time. You can select any inactive deployment, and then select the REDEPLOY icon to promote it as the active deployment. This allows you to quickly switch to a previous deployment if you find that your new deployment is broken.

In addition, Azure Websites also supports a multiple deployment environment. You can create additional deployment slots by clicking on the “Add a new deployment slot” link on the website’s DASHBOARD page. Then, you can deploy your site to any of the slots and swap among these slots at any time.

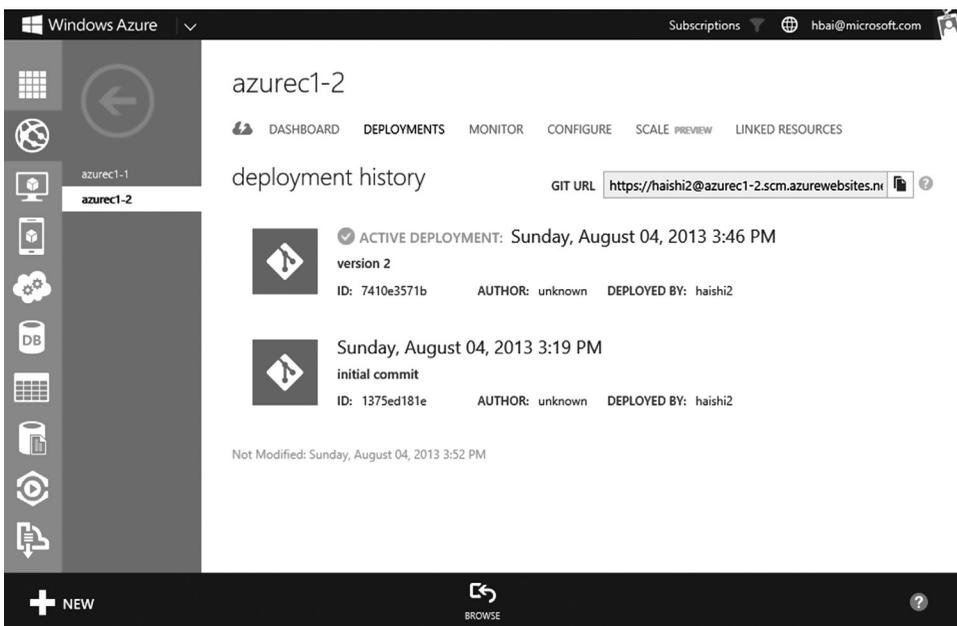


Figure 2.24 Two deployments of the website.



Figure 2.25 Updated PHP site.

2.4 Scaling of Websites

As mentioned earlier, one of the major benefits of hosting websites on Azure Websites is scalability. You can adjust the throughputs of your websites by simply modifying their configurations, without needing to care about the details of managing underlying infrastructures. All you need to do is to click the mouse several times, and your websites will gain the required capabilities within seconds.

On Azure Websites, you can scale your websites vertically or horizontally. Vertical scaling is done by choosing different hosting modes, and horizontal scaling is done by changing the number of website instances.

2.4.1 Vertical Scaling

■ Free mode

When you deploy a new website on Azure Websites, by default it is put under the **FREE** hosting mode. In this mode, your websites share the available resources on Microsoft Azure with other websites. Moreover, Microsoft Azure may allocate your websites to different virtual machines to balance resource usage.

To ensure that websites share the resources fairly, Microsoft Azure puts limits on the amount of CPU cycles, egress data, and storage space that your websites can use. Such limits, or quotas, are calculated per Microsoft Azure subscription. For instance, Microsoft Azure allows a maximum of 156M of egress data from websites hosted in free mode. When you deploy multiple websites, the 156M quota is shared among all these websites. In addition, different quotas have different sampling intervals. For example, egress data quotas are calculated per day, and storage quotas are calculated per hour (Figure 2.26).

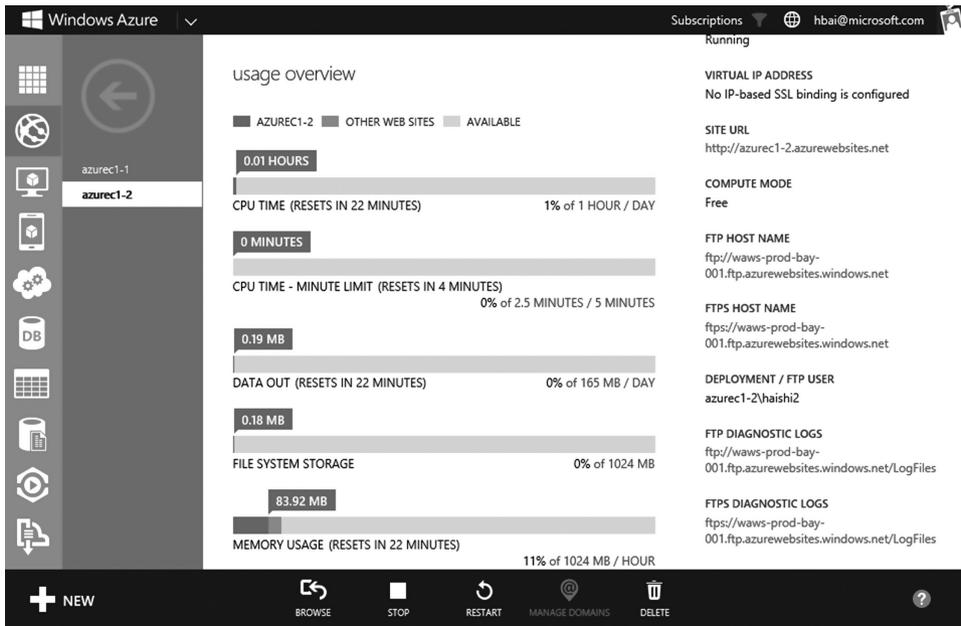


Figure 2.26 Usage overview section.

Note: When one of your websites exceeds your quota, **ALL** websites under the same subscription will be stopped, till the next sampling interval starts. You can monitor your resource usage on the website's dashboard page. In the usage overview section, you may not only monitor resource usages, but also observe when quotas will be reset.

- Shared mode

Websites hosted in the shared mode have the same running environment as in the free mode. The difference between the two modes is that the websites in the shared mode are not constrained by data egress quotas. In the shared mode, the first 5G of egress data is free, and more traffic is charged in a pay-as-you-go mode.

- Standard mode

Websites hosted in the standard mode run on dedicated virtual machines. Because they have exclusive accesses to resources, they can achieve higher, more stable performance without constraints of quotas. The resource consumption under standard mode is charged in pay-as-you-go mode. In addition, in the standard mode, you can specify the sizes of virtual machines, such as small (1 core, 1.75 GB memory), medium (2 cores, 3.5 GB memory), and large (4 cores, 7 GB memory). You should note that if you have multiple websites in the standard mode, they may share the same virtual machine. In other words, the virtual machines are dedicated to your subscription, but that does not necessarily mean each website has a dedicated virtual machine. However, you can select the websites you want to be hosted in the standard mode. If you want to ensure that one or several websites have dedicated resources, you can select to put only these sites in the standard mode (see Figure 2.28).

2.4.2 Horizontal Scaling

In shared and standard modes, you can horizontally scale your websites. In the shared mode, you can scale out to as many as 6 instances, while in the standard mode, you can scale out up to 10 instances. In addition, in the standard mode, you can choose whether the scaling is to be applied to all your websites or selected websites only.

Note: The available scaling ranges may change.

You can change scaling settings on the Microsoft Azure Management Portal. On a website's **SCALE** page, you can choose different hosting modes, and you can drag the **INSTANCE COUNT** slider to change the number of instances, as shown in Figure 2.27.

Tip: When you have unsaved changes, corresponding fields will be highlighted with purple background, reminding you to save the changes by using the SAVE icon on the command bar.

2.4.3 Autoscaling

In the standard mode, you can also enable autoscaling based on CPU usage. You can specify that when CPU usage goes above or below certain thresholds, the number of instances should be

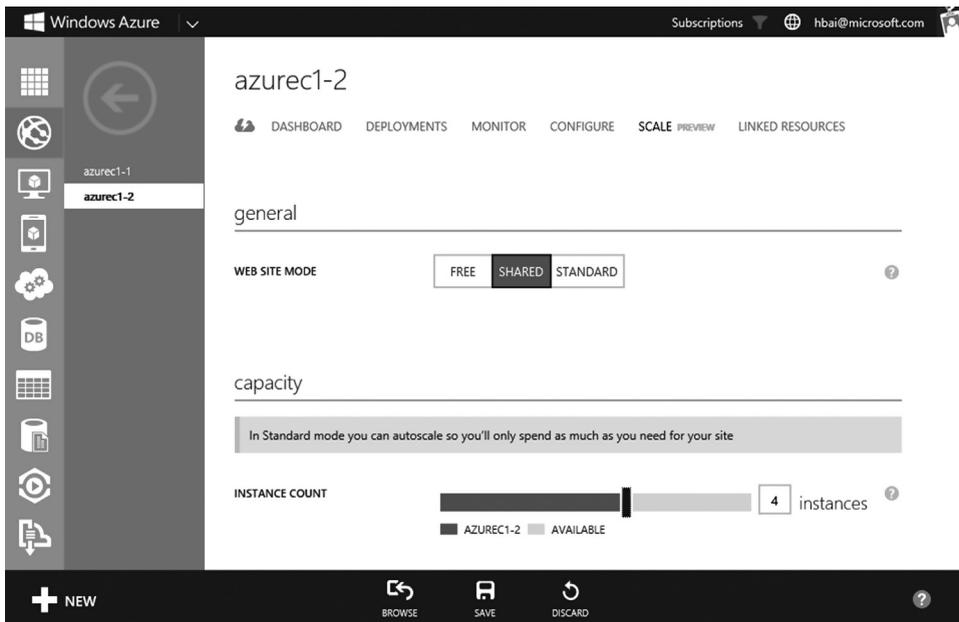


Figure 2.27 Scaling settings on Management Portal.

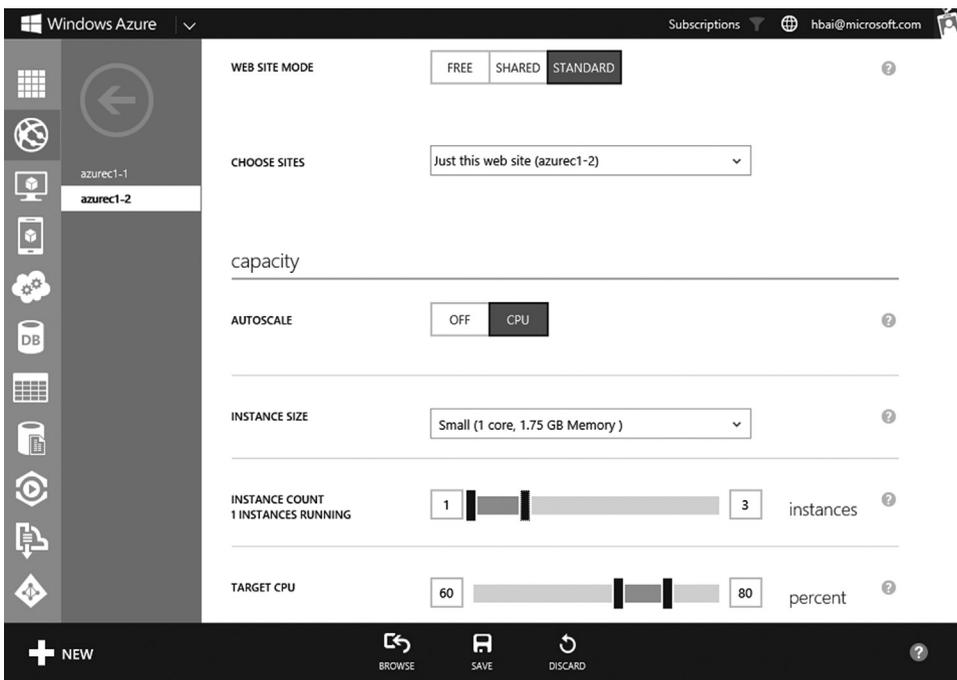


Figure 2.28 Autoscaling settings.

increased or decreased to keep the CPU usage within the given range. You can also specify how many instances you would allow your websites to be scaled to. Microsoft Azure will never go beyond the range you choose on this page (Figure 2.28).

2.5 Migrating Existing ASP.NET Websites

In previous examples, we have used new websites only. In real-life jobs, you may often need to migrate existing ASP.NET websites to Microsoft Azure. Obviously, because Microsoft Azure provides first-class support to ASP.NET, many ASP.NET websites can be directly deployed to Microsoft Azure (see the deployment steps in Example 2.1). However, not all ASP.NET websites can directly run on Microsoft Azure problem-free. Some sites may fail to launch; others may seem okay but expose some problems later on. In the second section of this book, we discuss various aspects in designing cloud services. Here we go through several pitfalls you might face.

2.5.1 Azure Websites Runtime Environment

Azure Websites runtime environment is based on Internet Information Services (IIS) running on the Windows operating system. However, because this is a multitenant environment, your websites share resources with other websites. Because of this, Azure Websites does not allow you to deeply customize IIS or virtual machines, such as installing third-party software. If you need more control over the running environment, you will have to consider Microsoft Azure Cloud Services (MACS) or Microsoft Azure Virtual Machines. We introduce MACS in Chapters 3 and 4.

2.5.2 Data Storage

Azure Websites does not allow you to access the entire file system on virtual machines. But you can read and write files and folders under the root folder of your sites. In addition, if you use databases, you will need to migrate your databases to either SQL Database or MySQL service (MySQL is provided by ClearDB). On the other hand, if you need high-performance key-value pair storage, or large-volume file storage, you should consider Microsoft Azure Storage service. We discuss various data storage options in Chapters 5 and 6.

2.5.3 Session States

Although Azure Websites supports cookie-based sticky sessions, it is advisable to save session states to external storages (such as databases). When the virtual machine hosting your website fails, the Azure Websites failover mechanism will reallocate your website to another healthy virtual machine. If you save your session states in memory, the states will be lost during such migrations. In addition, if a website is inactive for a long time, it will be removed from IIS to maintain website density and put into hibernation, which will cause websites to lose in-memory states as well. In general, you should use stateless design when designing cloud services. We discuss stateless design in Section II of this book.

2.6 Website Gallery

In addition to supports for ASP.NET, PHP, and Node.js, Azure Websites also provides a website gallery based on open-source projects. While this book is written, the gallery contains many different types of open-source projects such as Drupal, Wordpress, Joomla!, and MediaWiki. The companion website of this book is made using the Drupal template. Now let us learn how to use the gallery through an example.

Note: Refer to Example 7.5 for a Node.js example.

Example 2.5: Creating a Drupal Website

Difficulty: **

1. Sign in to Microsoft Azure Management Portal.
2. In command bar, click on the **NEW** icon, and then select **WEBSITE→FROM GALLERY** (Figure 2.29).
3. In **ADD WEB APP** dialog, select the **Acquia Drupal 7** template, and then click the next arrow to continue (Figure 2.30).
4. On the next page, enter the URL of your website. In the **DATABASE** field, choose an existing database, or create a new database for the site. Here choose the free 20 MB SQL database option. Enter your user name and password for the database, and click the next button to continue (Figure 2.31).
5. On the next page, create a new SQL database server. Enter the database name, user name, and password; then click on the check icon to complete the process (Figure 2.32).
6. Provisioning the website and related database resources only takes a couple of minutes. Once the site is created, click on the URL to open the website (as shown in Figure 2.33).

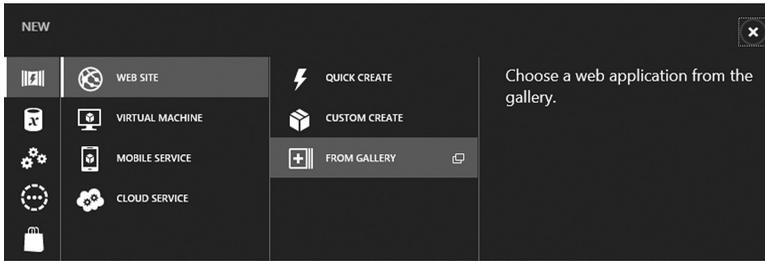


Figure 2.29 Create website from gallery.

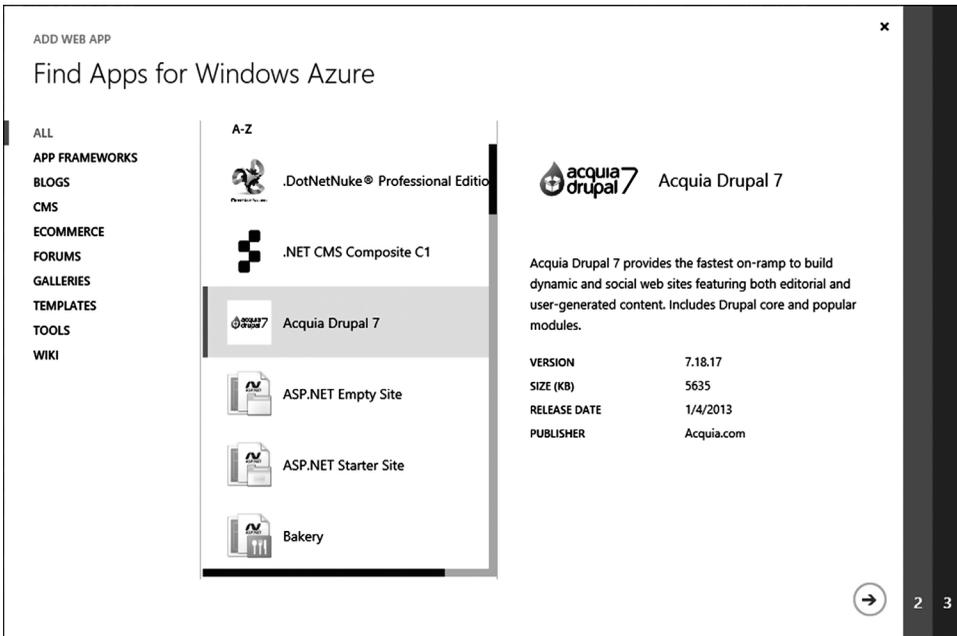


Figure 2.30 Drupal template.

7. On the Drupal welcome page, **Click here to continue** to install Drupal (Figure 2.34).
8. After installation, enter the email address, user id, and password for the administrator, and then click the **Save and Continue** button to continue (Figure 2.35).
9. Now your Drupal site is ready. Optionally, you can create an Aquia network subscription and install latest patches for your site (Figure 2.36).

2.7 Website Configuration

Azure Websites allows you to manage application settings through the Management Portal. In the following example, we will create a simple website that plots functions. The site allows the system administrator to change the colors of the axis and plot lines. Although this is not a typical site you might build, it would not hurt to have a little fun once in a while.

ADD WEB APP
✕

Configure Your App

URL
 ✔

.azurewebsites.net

DATABASE
 ▼

REGION
 ▼

Deployment Settings

DBUSERNAME

DBUSERPASSWORD

ENTER PASSWORD **CONFIRM PASSWORD**

LEGAL TERMS
By clicking the Next button, I acknowledge that I am getting this software from Acquia.com and that Acquia.com's legal terms apply to it. Microsoft does not provide rights for third-party software.



Acquia Drupal 7

Acquia Drupal 7 provides the fastest on-ramp to build dynamic and social web sites featuring both editorial and user-generated content. Includes Drupal core and popular modules.

VERSION	7.18.17
SIZE (KB)	5635
RELEASE DATE	1/4/2013
PUBLISHER	Acquia.com

1
← →
3

Figure 2.31 Configuring a website.

ADD WEB APP
✕

Specify database settings

NAME

SERVER
 ▼

SERVER LOGIN NAME

SERVER LOGIN PASSWORD **CONFIRM PASSWORD**

REGION
 ▼

CONFIGURE ADVANCED DATABASE SETTINGS

1 2
← →

Figure 2.32 Database settings.

web sites

NAME	STATUS	SUBSCRIPTION	LOCATION	MODE	URL	
azurecl-1	✓ Running	TE Account - Haishi Bai	West US	Free	azurecl-1.azurewebsites.net	
azurecl-2	✓ Running	TE Account - Haishi Bai	West US	Free	azurecl-2.azurewebsites.net	
zenofcloud	✓ Running	TE Account - Haishi Bai	West US	Free	zenofcloud.azurewebsites.net	



Figure 2.33 Created website.

Welcome



Welcome to Acquia Drupal installation

[Click here to continue](#)

- ✓ Choose profile
- ✓ Choose language
- ▶ **Start install**
 - Verify requirements
 - Set up database
 - Install profile
 - Configure site
 - Finished

Figure 2.34 Drupal welcome page.

Configure site



- ✓ Choose profile
- ✓ Choose language
- ✓ Start install
- ✓ Verify requirements
- ✓ Set up database
- ✓ Install profile
- ▶ **Configure site**
- Finished

SITE INFORMATION

Site name *

Site e-mail address *

Automated e-mails, such as registration information, will be sent from this address. Use an address ending in your site's domain to help prevent these e-mails from being flagged as spam.

Acquia subscription identifier

If you have an Acquia Network subscription, please enter the subscription identifier. You can also provide it later at Administer > Configuration > Acquia Network settings.

Figure 2.35 Configuring Drupal site.

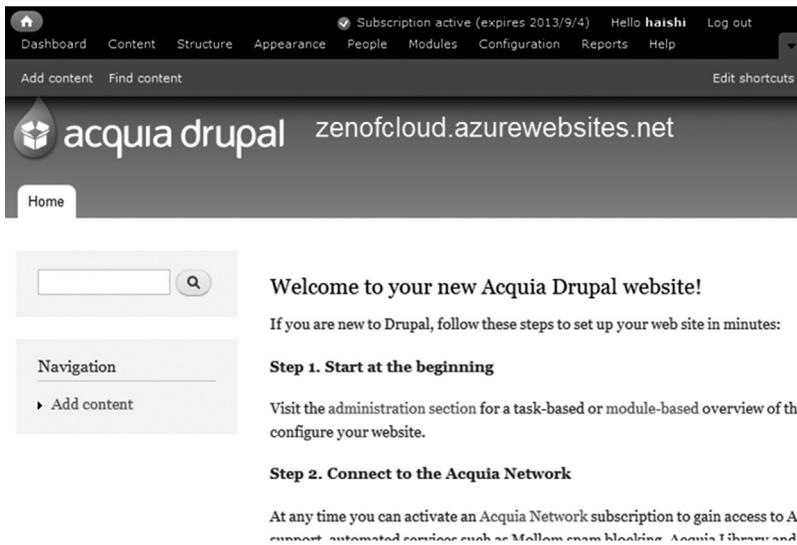


Figure 2.36 Drupal website.

Example 2.6: Website Configuration—Plotting User Functions

Difficulty: ***

1. Follow steps 1 through 10 in Example 2.1 to create a new ASP.NET website.
2. Add a default.aspx Web Form to the site. The main part of the code is in JavaScript, which takes an x expression and a y expression and plots the function in a `<canvas>` tag. The application takes two settings to control the axis color (`axis-color`) and the plot color (`plot-color`). The application uses `System.Configuration.ConfigurationManager.AppSettings` to read these settings. The complete source code of the form is as follows (Code List 2.2):
3. In the web.config file, add the corresponding application settings (in bold):

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
  </system.web>
  <appSettings>
    <add key="axis-color" value="gray" />
    <add key="plot-color" value="black" />
  </appSettings>
</configuration>
```

4. Press F5 to run the application locally. Click on the **render** button to plot the default function. You can enter other expressions with t as argument to further test the application (Figure 2.37).

Note: In order to simplify the code, we skip the input validation. So the code is subject to code injection attack. You should add validations in your version.

CODE LIST 2.2 WEB FORM THAT PLOTS FUNCTIONS

```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="default.
  aspx.cs" Inherits="Example2.6._default" %>

<%@ Import Namespace="System.Configuration" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8" />
  <title></title>
  <script>
    function plotExpression() {
      //get size of canvas object
      var canvas = document.getElementById('plot');
      var height = canvas.height;
      var width = canvas.width;
      //adjust origin and scale based on canvas size
      var xOrigin = width / 2;
      var yOrigin = height / 2;
      var xScale = width / 10;
      var yScale = height * xScale / width;
      //initialize canvas
      var context = canvas.getContext('2d');
      context.clearRect(0, 0, width, height);
      //read axis color from application settings
      context.strokeStyle =
        '<%=ConfigurationManager.AppSettings["axis-color"]%>';
      //draw axis
      context.beginPath();
      context.moveTo(0, yOrigin);
      context.lineTo(width, yOrigin);
      context.moveTo(xOrigin, 0);
      context.lineTo(xOrigin, height);
      context.stroke();
      //read plot color from application settings
      context.strokeStyle =
        '<%=ConfigurationManager.AppSettings["plot-color"]%>';
      //plot the function
      context.beginPath();
      for (var t = -10; t <= 10; t += 0.1) {
        var x = eval(xExp.value);
        var y = eval(yExp.value);
        context.lineTo(xOrigin + x * xScale, yOrigin - y *
          yScale);
        context.stroke();
      }
    }
  </script>
</head>

```

```

<body>
x expression:<input type="text" id="xExp"
  value="Math.sin(2*t-2*Math.PI/3) * 3"
  style="width: 300px" /><br />
y expression:<input type="text" id="yExp" value="Math.cos(3*t) * 3"
  style="width: 300px" /><br />
<input type="button" id="render" value="render"
  onclick="javascript: plotExpression();" />
<br />
<canvas width="500" height="500" id="plot"></canvas>
</body>
</html>

```

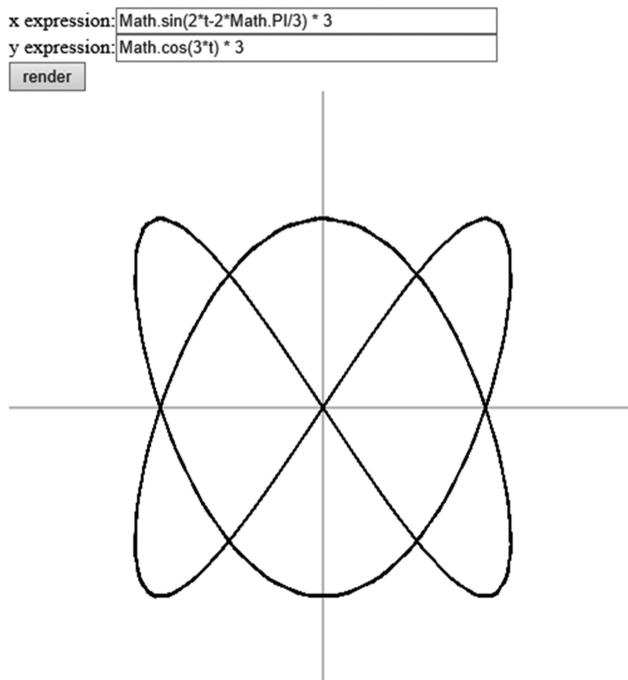


Figure 2.37 Sample plot.

5. Follow steps 12 through 15 in Example 2.1 to deploy the website on Azure Websites.
6. On the details page, click on the **CONFIGURE** link to open the configuration page, on which you will find the **app settings** section (you need to scroll down to see the section) (Figure 2.38).
7. Although the *axis-color* and *plot-color* settings are not automatically populated from your web.config file, you can enter new values in this section to replace the values in web.config. After you have added the two settings, click the **SAVE** button on the command bar to save changes (Figure 2.39).

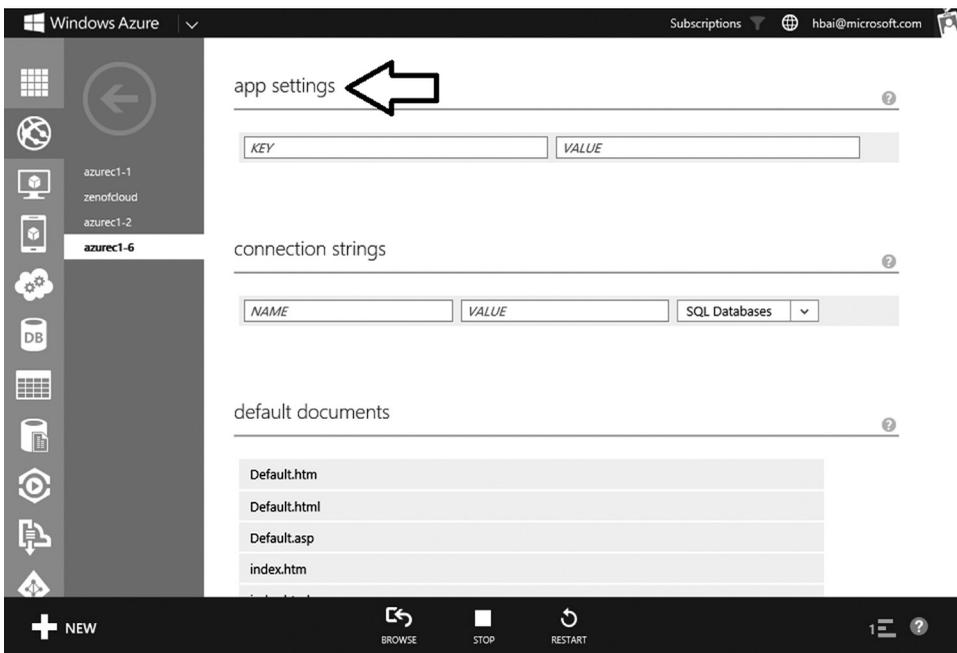


Figure 2.38 App settings section.

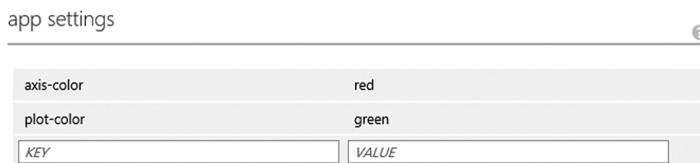


Figure 2.39 Updated application settings.

Note: Azure Websites saves setting values in a central database so that multiple instances of a website can share the same set of settings.

- In the browser, refresh the page and plot the function again. You will find the diagram colors have changed.

Tip: In PHP, you can use the `getenv()` method to read settings. For example, `getenv("aix-color")`

In Node.js, you can use `process.env` to read settings. For example, `process.env.plot-color`

2.8 Website Diagnostics and Monitoring

2.8.1 Website Diagnostics

First of all, because you can debug your ASP.NET websites in your development environments, many problems can be discovered and resolved locally before your websites are deployed to Microsoft Azure. Of course, the most annoying bugs always jump out on you in production environments. Azure Websites diagnostics comes to the rescue in this case. You can turn on and off the collection of various diagnostic logs on a website's configuration page (Figure 2.40).

- Application diagnostics

Turning on application diagnostics allows you to save tracing information from your source code to either the local file system or a storage account. When you save trace files to the file system, you can download them via an FTP within 12 hours. When you save trace information to a storage account, you can view them via any clients that support Microsoft Azure Storage service (we introduce the service in Chapter 6) (Figure 2.41).

You can get the FTP address to download the diagnostics logs on the website's dashboard page (Figure 2.42).

Just as in any other .NET applications, you can instrument your ASP.NET applications using System.Diagnostics namespace. For example, you can modify the default.aspx.cs in Example 2.6 to add some tracing information (Code List 2.3):

After the website is deployed, you can download the tracing files from the mentioned FTP address. For instance, this code generates the following file content:

```
2013-04-14T01:16:12 PID[4516] Information This is a test message
```



Figure 2.40 Diagnostics settings.

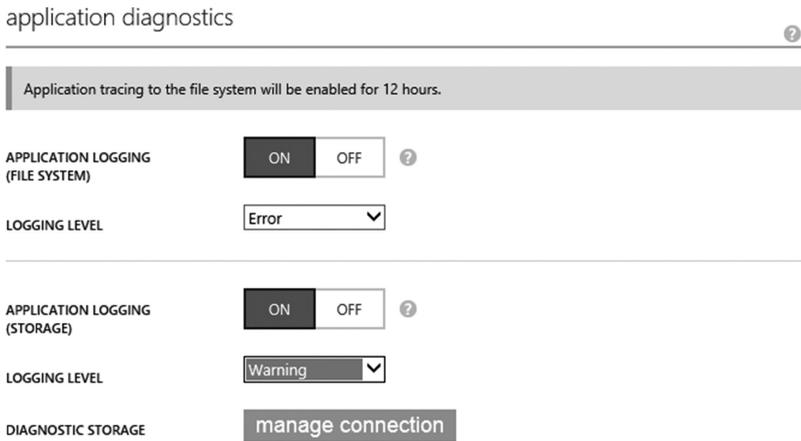


Figure 2.41 Application diagnostics settings.

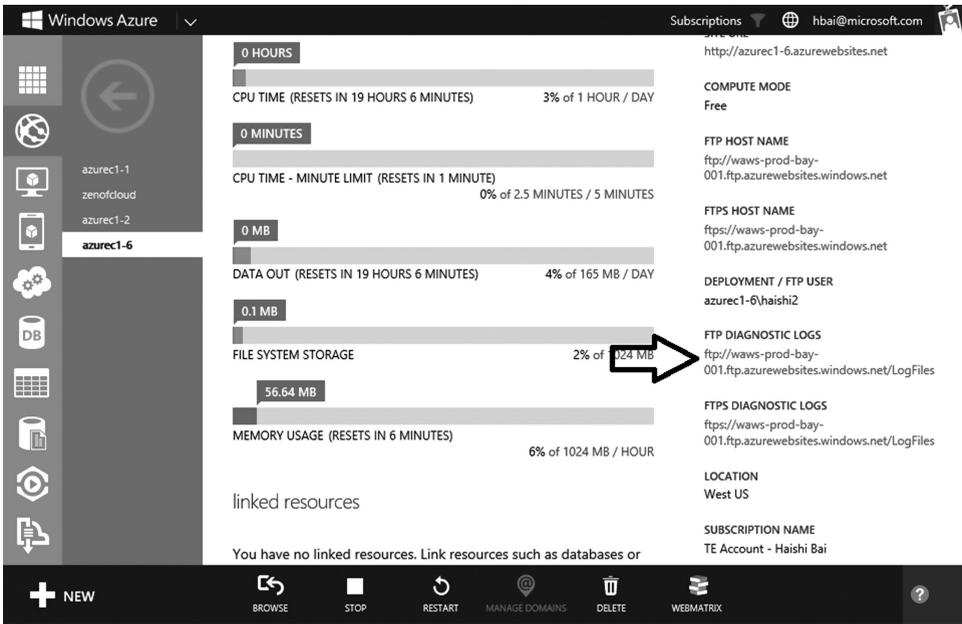


Figure 2.42 FTP address to download diagnostics logs.

In addition to collecting diagnostics logs, you can turn on or off web server tracing (see Figure 2.40). Then, you can download the tracing files via an FTP. Figure 2.43 shows that after web tracing is turned on, there is an additional http directory, which contains website tracing files.

■ Log streaming

You can also stream logs directly to Visual Studio. To start streaming logs, right-click on the website in Visual Studio Server Explorer, and select View Streaming Logs in the output window menu (Figure 2.44).

Then, you will see the logs streamed to the output window (Figure 2.45).

CODE LIST 2.3 INSTRUMENT ASP.NET CODE

```
public partial class _default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        System.Diagnostics.Trace.TraceInformation("This is a
            test message");
    }
}
```

FTP directory /LogFiles at waws-prod-hk1-001.ftp.azurewebsites.windows.net

To view this FTP site in File Explorer: press Alt, click View, and then click Open FTP Site in File Explorer.

[Up to higher level directory](#)

04/14/2013 01:34AM	Directory Application
04/13/2013 05:38PM	Directory Git
04/14/2013 01:40AM	Directory http



Figure 2.43 Web tracing files.

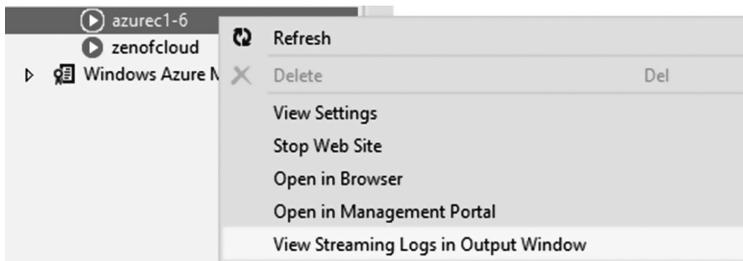


Figure 2.44 View Streaming Logs menu.

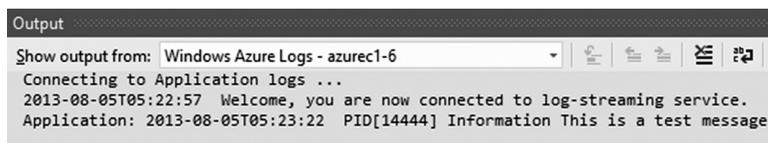


Figure 2.45 Streamed logs.

2.8.2 Website Monitoring

You can configure performance counters collected by Azure Websites on a website’s **MONITOR** page. On this page, you can click on the **ADD METRICS** icon on the command bar to choose performance counters; you can also turn on or off plot lines by clicking on the colored checks (Figure 2.46).

Furthermore, in the standard mode, you can turn on website monitoring, which tests the performance of your website from distributed test locations around the globe. As shown in Figure 2.47, two test locations have been configured—one in Chicago and one in Dublin—to test the performance of the home page.

After several hours, you will see the test results on the dashboard (Figure 2.48).

Clicking on the endpoint name brings up the test details (Figure 2.49).

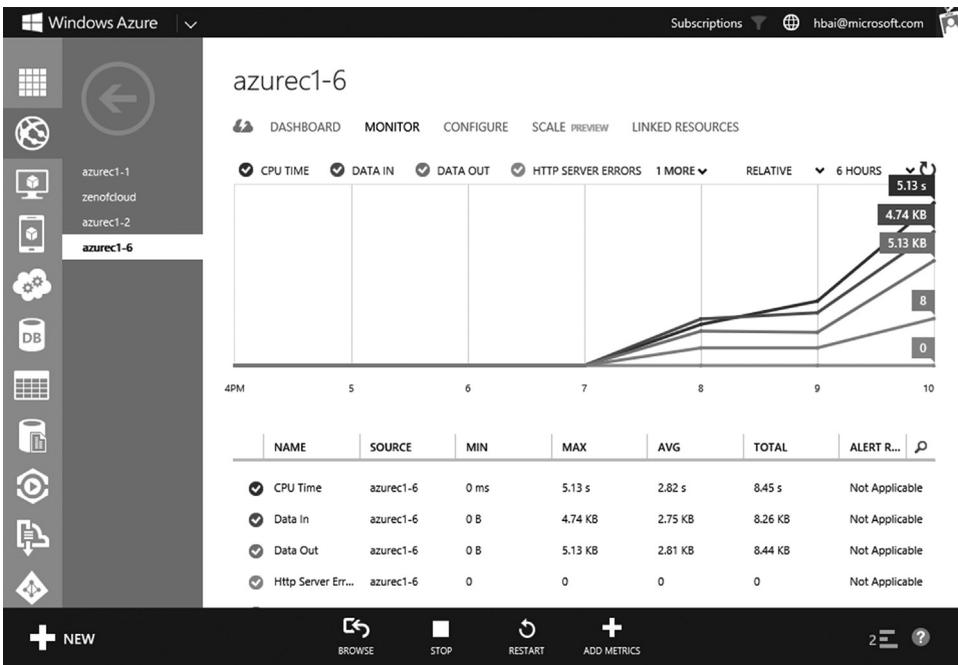


Figure 2.46 Monitor page.



Figure 2.47 Distributed test locations.

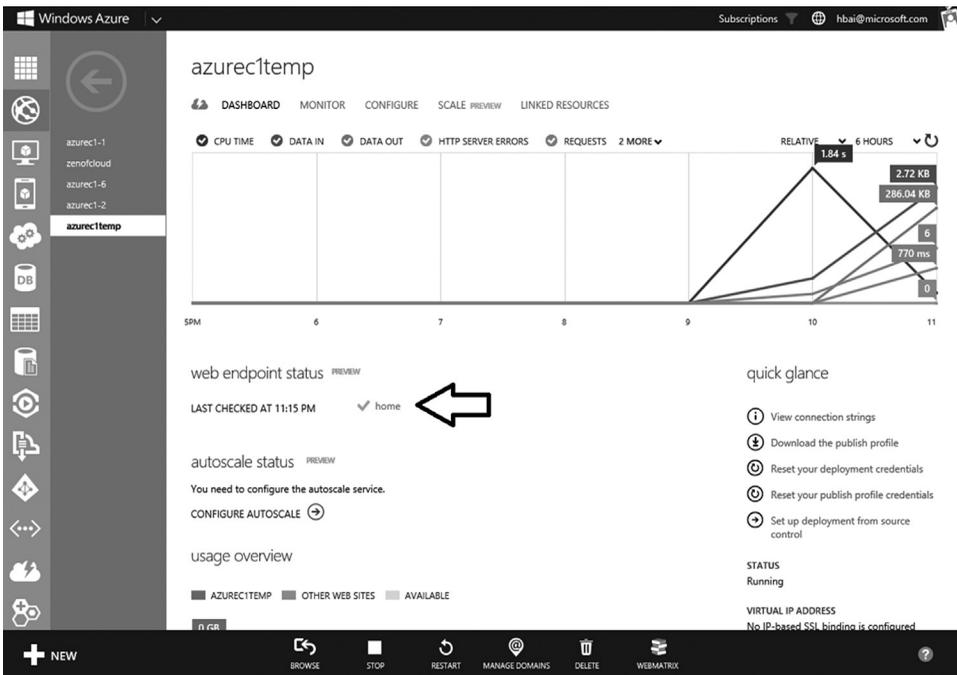


Figure 2.48 Web endpoint status.

WEB ENDPOINT STATUS ✕

URL <http://azurec1temp.azurewebsites.net/>

Last test result ▲ IE: Dublin ▲ US: IL - Chicago

Last 5 test results from each location

TIME STAMP	LOCATION	RESPONSE TIME	STATUS
8/4/2013 11:20:00 PM	US: IL - Chicago	346 ms	The request completed successfully.
8/4/2013 11:15:00 PM	US: IL - Chicago	537 ms	The request completed successfully.
8/4/2013 11:10:00 PM	US: IL - Chicago	612 ms	The request completed successfully.
8/4/2013 11:05:00 PM	US: IL - Chicago	Not Available	Not Available
8/4/2013 11:00:00 PM	US: IL - Chicago	Not Available	Not Available
8/4/2013 11:20:00 PM	IE: Dublin	874 ms	The request completed successfully.
8/4/2013 11:15:00 PM	IE: Dublin	899 ms	The request completed successfully.

Figure 2.49 Test result details.

2.8.3 Custom Domain Names

If you wish to add custom domain names on your websites, you need to create a CNAME record with your DNS provider to map your domain name to *[website].azurewebsites.net*. For example, when *azurebyexamples.haishibai.com* was mapped to *azurebyexamples.azurewebsites.net*, a CNAME record was created with my DNS provider (Figure 2.50).

Then, you can manage your custom domain names (applicable only to standard mode). Click on the **manage domains** button (Figure 2.51).

Finally, in the **Manage custom domains** dialog, add the custom domain names you want to use (Figure 2.52).

Add DNS Record

Record type:

Enter an Alias Name: ?

Points to Host Name: ?

TTL: ?

OK [Cancel](#)

Figure 2.50 CNAME record.

domain names ?

azurec1temp.azurewebsites.net

manage domains

Figure 2.51 Manage custom domains.

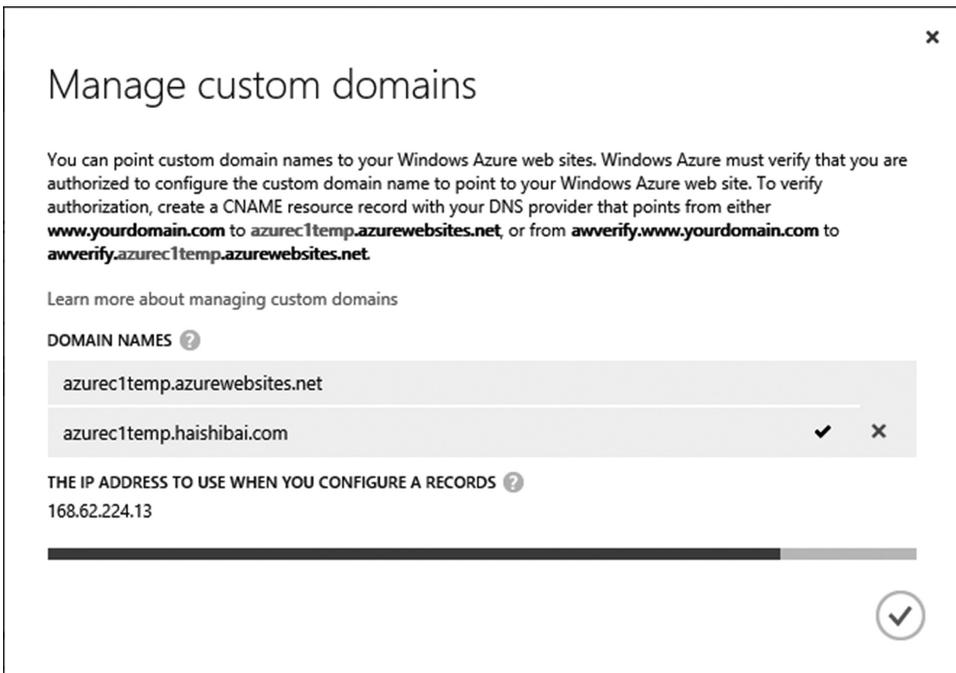


Figure 2.52 Manage custom domains dialog.

2.9 Summary

Websites are the most common type of cloud services. In this chapter, we introduced Microsoft Azure Websites in detail. We studied how to develop an ASP.NET MVC website using Visual Studio; we created a PHP website; we deployed a Drupal website in minutes; and we also studied how to use WebMatrix, FTP, and Git to deploy websites. In addition, we learned different concepts and tools for website managing, monitoring, and tracing. We also examined different scaling options in detail. Finally, we learned the basic steps for configuring custom domains.

Chapter 3

Cloud Service Fundamentals

3.1 Microsoft Azure Cloud Services

As we learned in the previous chapter, Azure Websites provides a quick and easy way to create and host websites on Microsoft Azure. However, the architecture of a common cloud service is often much more complex than a simple website. For example, in a multitier cloud service, the presentation layer, business layer, and data layer are separate. Such a complex architecture is hard to implement with Azure Websites. In addition, under Service-Oriented Architecture (SOA), many cloud services are not presented as websites. They may not have user interfaces, but only provide integration endpoints based on different protocols such as HTTP, TCP, or UDP. Furthermore, many cloud services need to integrate with legacy systems, which can be hosted on Microsoft Azure Virtual Machines, on-premise data centers, or even local servers in users' offices. Microsoft Azure Cloud Services (MACS) provides support for all these scenarios.

Because MACS is a common platform for all kinds of cloud services, there is no problem in using it for websites (as we mentioned in Chapter 1, websites are just one type of cloud services). This is exactly what we are going to do in the following example. Then, we will go over some basic concepts.

Example 3.1: Hello, Microsoft Azure Cloud Service!

Difficulty: *

1. Launch Visual Studio as an administrator.

Note: We need to launch Visual Studio with elevated privileges because of the requirements of the Computer Emulator in Microsoft Azure SDK. Starting with Microsoft Azure SDK 2.3, you don't need to launch Visual Studio as an administrator anymore, as the new lean emulator doesn't require elevated privileges. The lean emulator doesn't support multi-instance emulation, which you can still get using the full emulator.

2. Select **FILE**→**New**→**Project** menu.
3. On **New Project** dialog, select the **Microsoft Azure Cloud Service** template under the **Cloud** category. Enter a project name, and then click the **OK** button to continue (Figure 3.1).

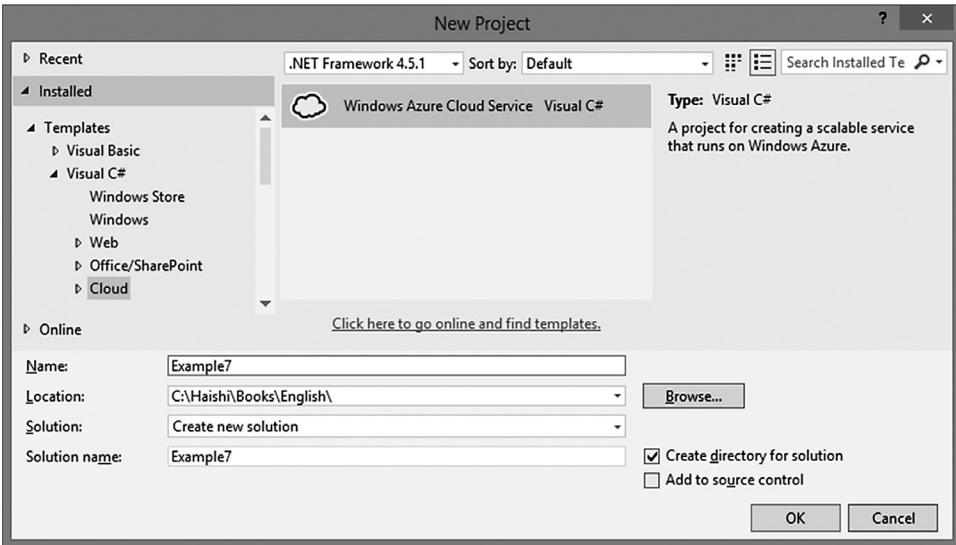


Figure 3.1 Create a new cloud service.

4. On **New Microsoft Azure Cloud Service** dialog, add an **ASP.NET Web Role** to the cloud service, and then click the **OK** button to continue (Figure 3.2).
5. On **New ASP.NET Project** dialog, select the **MVC** template, and click **OK** to continue (Figure 3.3).
6. Congratulations! You’ve just created your first cloud service! And you haven’t even written a single line of code yet. Isn’t that cool? What’s even cooler is that you can now test your cloud services locally by pressing F5.

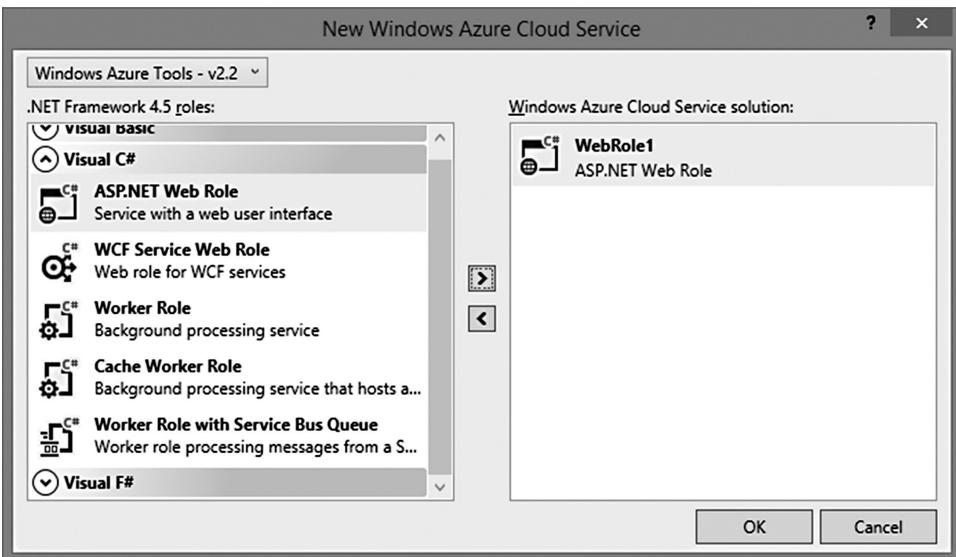


Figure 3.2 Adding a web role to a cloud service.

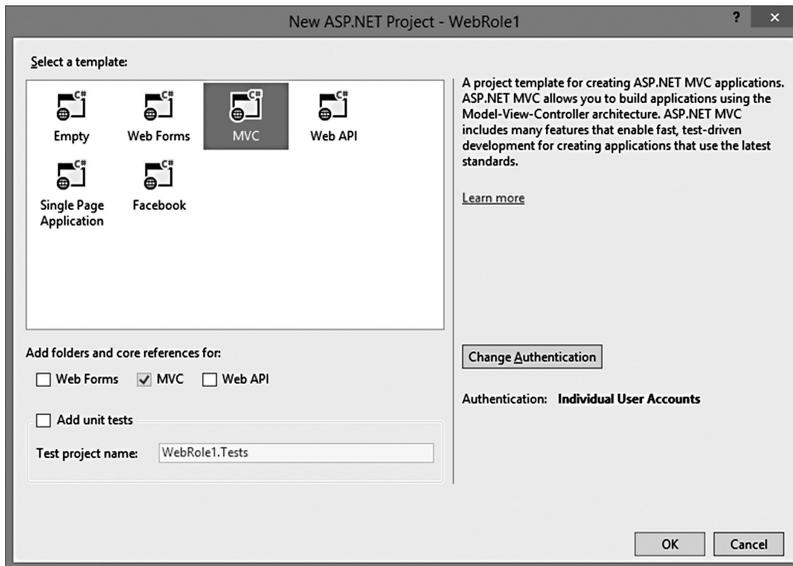


Figure 3.3 New ASP.NET MVC 4 Project dialog.

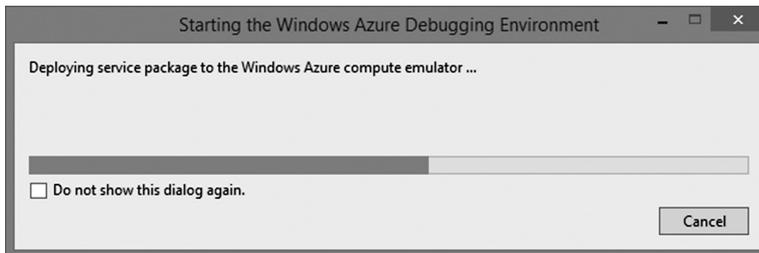


Figure 3.4 Starting the Microsoft Azure Debugging Environment.

- When you press F5, you will notice a dialog popping up (see Figure 3.4). This dialog represents something very unique provided by Microsoft Azure for cloud service developers—a local debugging environment. With this emulated environment providing simulated MACS as well as Microsoft Azure Storage services (more on this in later chapters), you can test and debug your cloud services locally. You can set up break points and step through code execution just as if you were debugging a local application. All these advantages improve the work efficiency of developers and prove yet again the philosophy of Microsoft, which is developer first.

Note: If you see a security alert when the debugging environment is launched, click Allow access to continue (Figure 3.5).

- Shortly after the debugging environment is launched, you will see the website running on it (Figure 3.6).



Figure 3.5 Windows security alert.

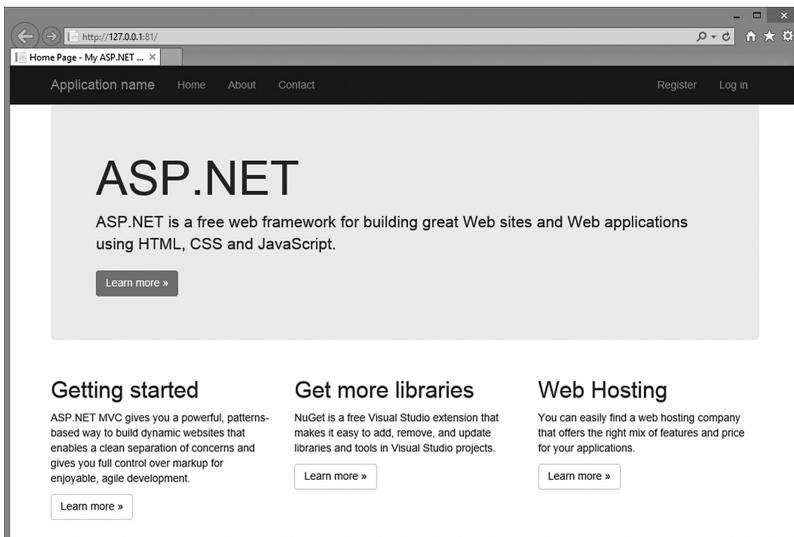


Figure 3.6 Website running on the debugging environment.



Figure 3.7 Debugging environment icon on Windows task bar.

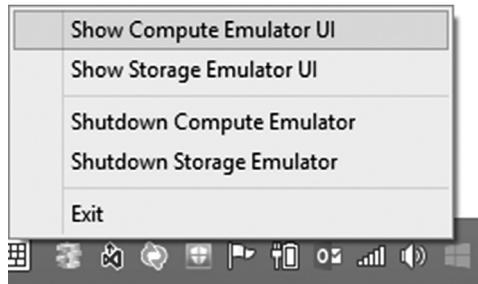


Figure 3.8 Shortcut menu to bring up emulator UI.

Note: About the Microsoft Azure Debugging Environment

The Microsoft Azure Debugging Environment is part of Microsoft Azure SDK. After the environment is launched, you can see its icon on the Windows task bar (Figure 3.7).

Right-click on the icon, and select Show Compute Emulator UI to bring up the emulator UI (Figure 3.8).

You can observe the status of our web role and its tracing outputs. We will make more use of this UI in Section 3.1 (Figure 3.9).

3.2 Cloud Services and Roles

In Example 3.1, we created a new cloud service. Before we move on to new contents, let us first examine the source code structure of Example 3.1. As shown in Figure 3.10, the solution contains two projects. The first is the cloud service project, which contains a *Roles* folder. Under the Roles folder is the definition of our web role (WebRole1). The definition corresponds to the other project in the solution, which is an ASP.NET project with the same name. If you are familiar with ASP.NET MVC, you can easily see this project is an ordinary ASP.NET MVC project. The only exception is the extra *WebRole.cs* file. For now you can ignore this file. Actually, even if you delete

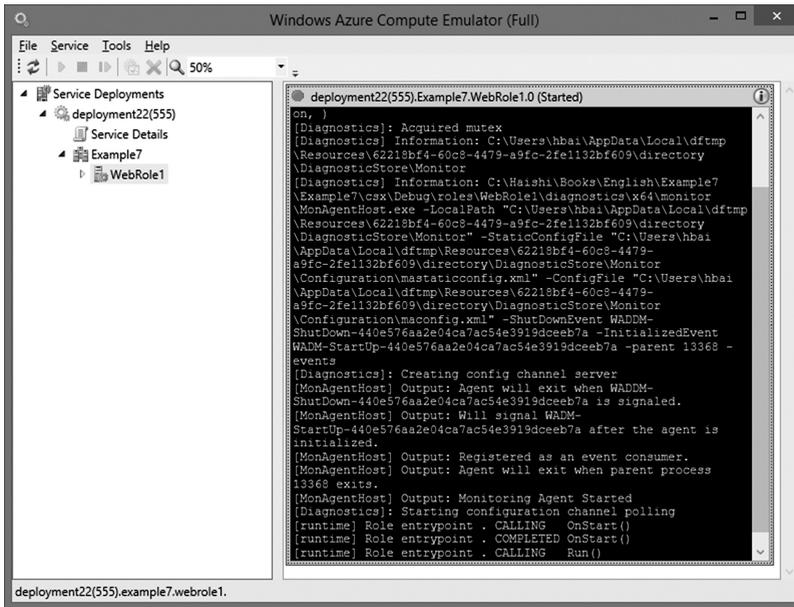


Figure 3.9 Compute emulator UI.

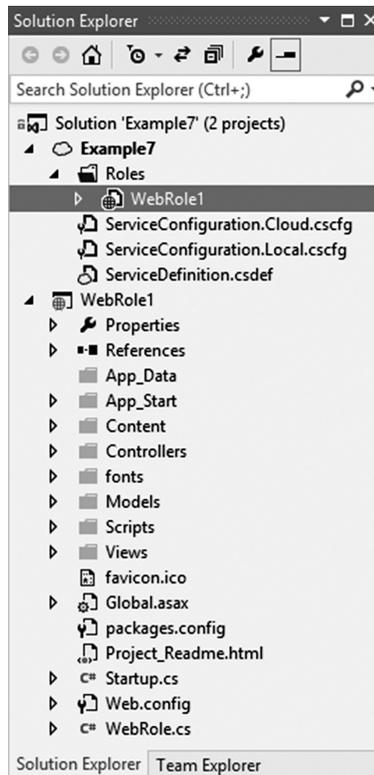


Figure 3.10 Cloud service solution code structure.

this file, it will not affect compilation or execution of your web role. In summary, a cloud service solution has the following structure:

- A cloud service project, which contains definitions and configurations of all roles.
- Each role in a cloud service has its own corresponding project. For example, for a web role, its corresponding project is an ASP.NET project.

We have mentioned “role” several times but have not given a formal definition yet. Now it is time for us to do that and go over some other basic concepts related to cloud services.

3.2.1 Role

Simply speaking, a role is a unit that provides some kind of service. End users can access these services via endpoints exposed by the role. For example, a web role is a functional unit that provides a website, which is accessible by HTTP, port 80. The left side of Figure 3.11 depicts such a web role. End users do not need to understand any implementation details of the role. As long as they can access the endpoint (the URL of the website), they can utilize the service. On Microsoft Azure, a role can have multiple endpoints, and these endpoints can use different protocols, such as HTTP, TCP, and UDP, as shown in the right part of Figure 3.11.

3.2.2 Cloud Service

Generally speaking, a Cloud Service on Microsoft Azure is a container that can hold a number of different roles. Figure 3.12 shows an example of a cloud service. As a container, it holds two roles. The first role defines one endpoint, and the second role defines three endpoints.

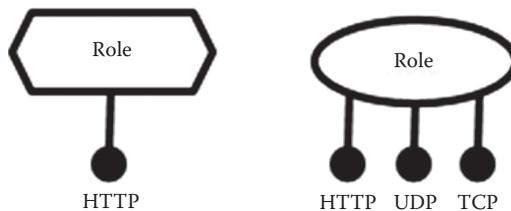


Figure 3.11 Role example.

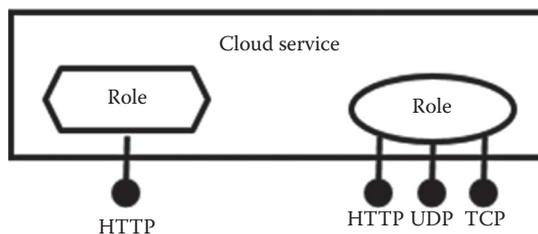


Figure 3.12 Cloud service example.

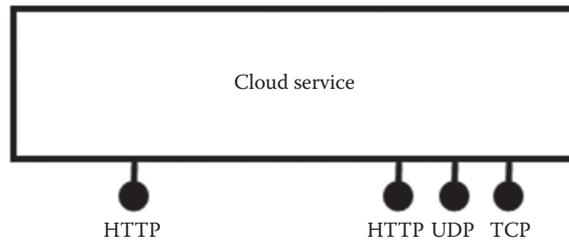


Figure 3.13 Consumer's view of a cloud service.

Under most circumstances, consumers of cloud services do not care exactly which roles are providing the required services. From their perspective, they can simply consider that all endpoints are provided by the cloud service, as shown in Figure 3.13.

Of course, a cloud service is merely a container of roles, but it is also a development, management, communication, and security boundary. First of all, on Microsoft Azure, the development unit is a cloud service. Even if you only need one role, you need to put it in a cloud service, as we have seen in Example 3.1. Second, when you deploy to Microsoft Azure, the deployment unit is a cloud service as well. Third, a cloud service defines a private communication space for the roles it contains. Roles in the same cloud service can communicate with each other via private endpoints, which are invisible from outside the cloud service. As our discussion goes deeper, you will understand more characteristics of cloud services as containers.

3.3 Basic Steps of Cloud Service Deployment

In Example 3.1, we only tested our cloud service in an emulated environment. Now let us deploy the service to Microsoft Azure. The deployment process consists of the following steps:

1. Create a new Cloud Service on Microsoft Azure.
2. If you have not downloaded the publish profile, you need to download it first.

Note: The publish profile is valid for all services in all your Microsoft Azure subscriptions associated with the same Microsoft Account (or Microsoft Azure Active Directory account), so you only need to download it once. As a matter of fact, because Microsoft Azure generates a new digital certificate for authentication each time you download the profile, and there is a limit on the number of certificates you can have, it is not recommended to download this file repeatedly.

The publish profile of Azure Websites cannot be used as a MACS publish profile.

3. Compile and package the service. The packing process creates two files: a cloud service package file (.cspkg) and a cloud service configuration file (.cscfg).
4. The package file and the configuration file are uploaded to Microsoft Azure.

Note: To be exact, the cloud service package and the configuration file are uploaded to a storage account of your Microsoft Azure subscription. We will introduce Microsoft Azure Storage services in Chapter 6.

5. Microsoft Azure provisions required virtual machines based on the configuration file.
6. Cloud service package is deployed to the virtual machines.
7. Service roles are launched to handle user requests.

Now let us go through this process with an example.

Example 3.2: Deploy Cloud Service

Difficulty: *

1. Launch Visual Studio as an administrator. Open the solution in Example 3.1.
2. Right-click on the cloud project in the solution, and then select the **Publish** menu (Figure 3.14).
3. If you have not deployed any cloud services before, you need to download a publish profile. When the **Publish Microsoft Azure Application** dialog is launched, an additional dialog box shows up to allow you to sign in to your Microsoft Azure subscription. Enter your credentials and click the **Sign in** button to sign in (Figure 3.15).
4. Once signed in, you can select the subscription you want to use from the **Choose your subscription** dropdown box. Then, click the **Next** button to continue (Figure 3.16).
5. On the next screen, if you have not created any cloud services yet, **Create Microsoft Azure Services** dialog will automatically pop up. Continue with step 6. Otherwise, pull down the **Cloud service** dropdown box, and select **<Create New...>** (Figure 3.17).
6. On **Create Microsoft Azure Services** dialog, enter a name for your cloud service and choose a location where you want the service to be deployed. The cloud service name has to be globally unique, and the final address of your cloud service will be *[name].cloudapp.net*. Click on the **OK** button to continue (Figure 3.18).

Note: If you have not created a storage account yet, the tool will automatically create a new storage account with the same name as your cloud service for package uploads. We introduce Microsoft Azure Storage services in Chapter 6.



Figure 3.14 Publish menu.

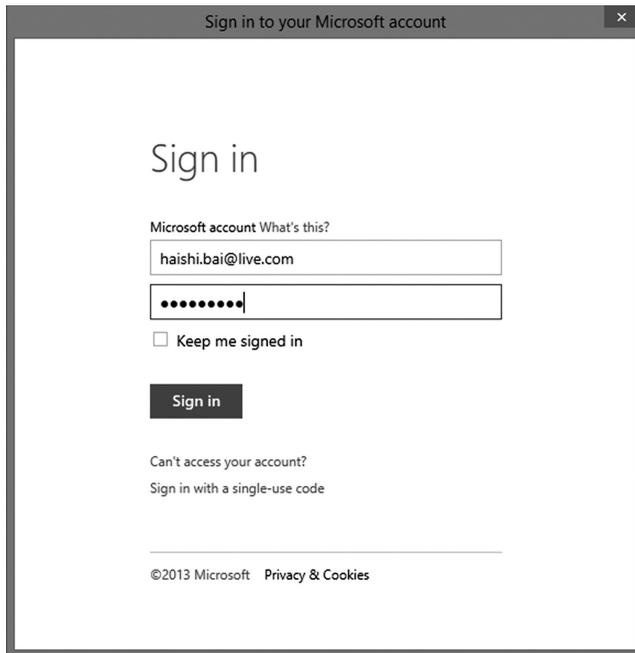


Figure 3.15 Sign in to Microsoft Azure subscription.

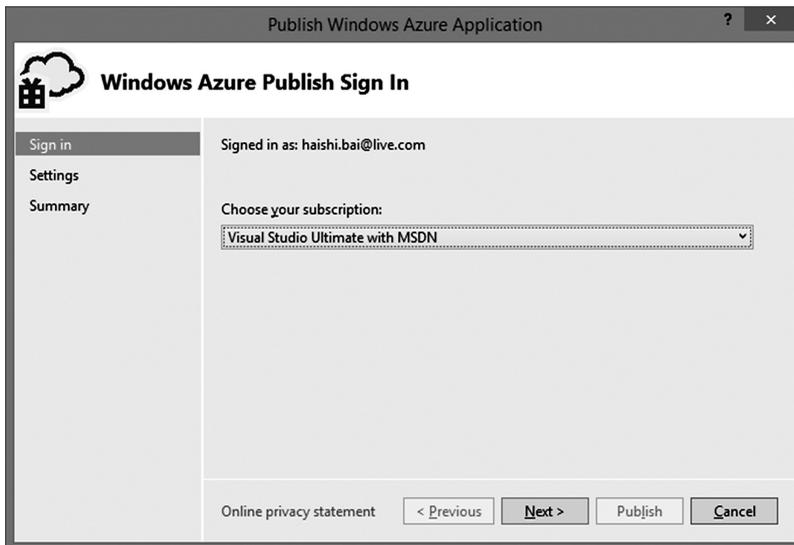


Figure 3.16 Select Microsoft Azure subscription.

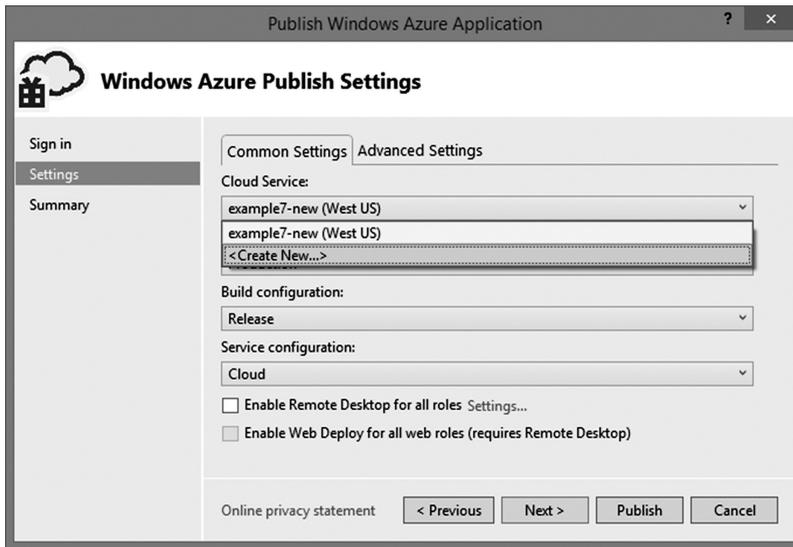


Figure 3.17 Create a new Cloud Service from the publish wizard.

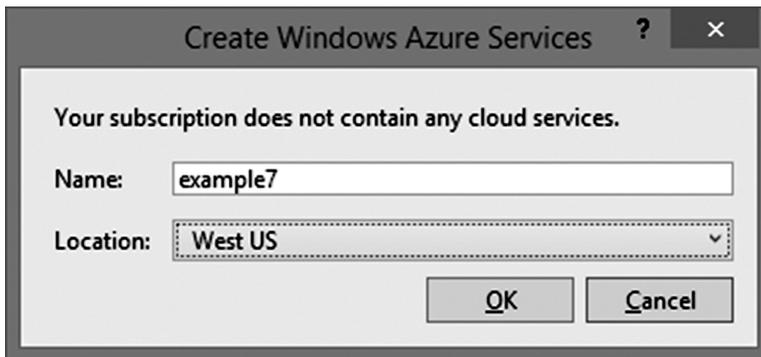


Figure 3.18 Create Microsoft Azure Services dialog.

7. [Optional] Before we click the **Next** button, check the **Enable Remote Desktop for all roles** checkbox.
8. [Optional] Step 7 pops up a **Remote Desktop Configuration** dialog, where you can enter your user name and password for remote desktop access to virtual machines hosting your cloud service (Figure 3.19).
9. Click the **Next** button to continue.
10. Click the **Publish** button. The deployment process takes several minutes. You can observe the deployment progress in the **Microsoft Azure Activity Log** window. After deployment succeeds, you can click on the **Website URL** link to bring up the website (Figure 3.20).

Now we can also manage our cloud service on Microsoft Azure Management Portal. You will find that the user interface for cloud service management is very similar to website management UI (Figure 3.21).

Open the **INSTANCE** page of the cloud service. You will see that the cloud service has one running instance. If you have enabled remote desktop access in steps 7 and 8, you can click on

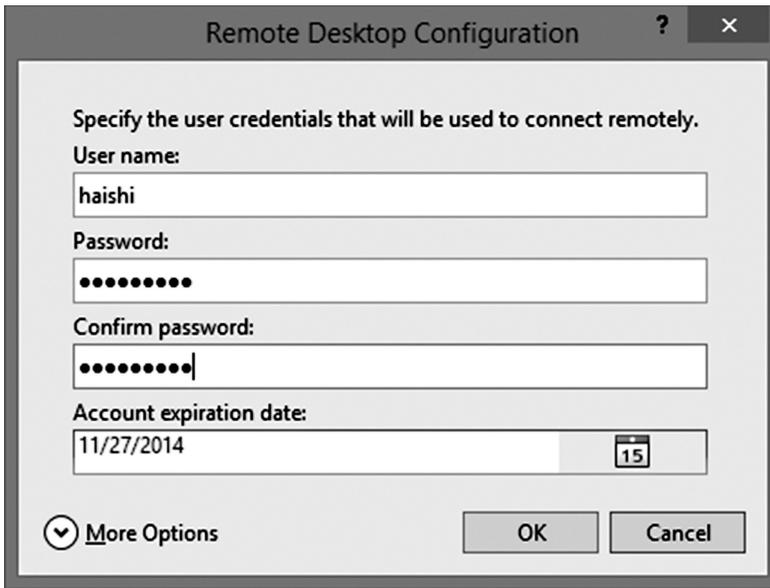


Figure 3.19 Remote Desktop Configuration dialog.

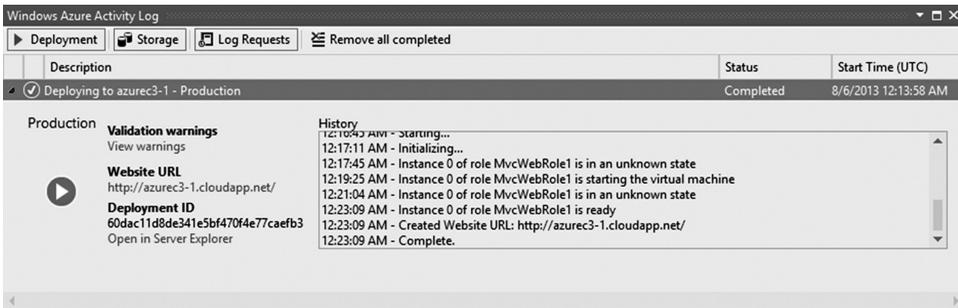


Figure 3.20 Microsoft Azure Activity Log window.

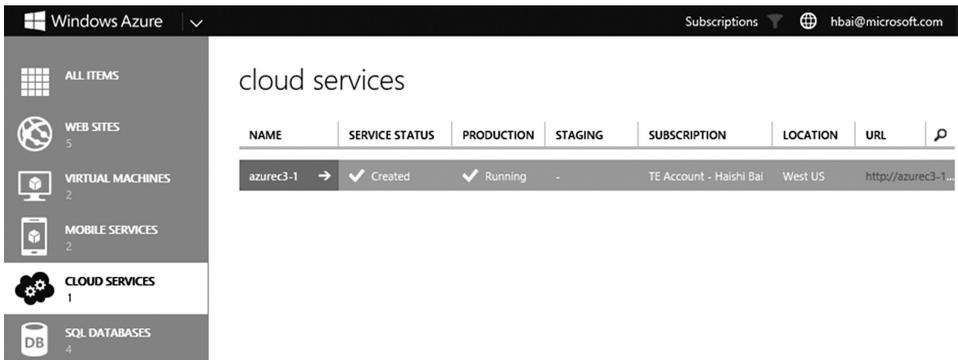


Figure 3.21 Cloud service management UI on Management Portal.

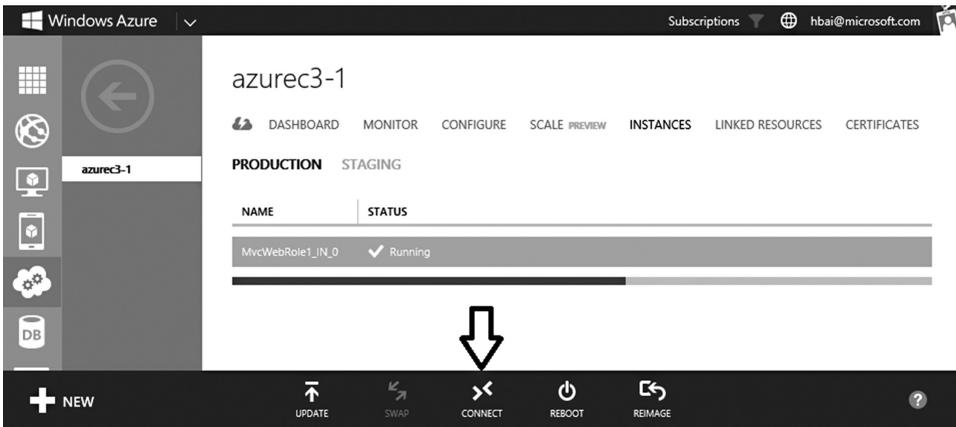


Figure 3.22 Instance page of a cloud service.

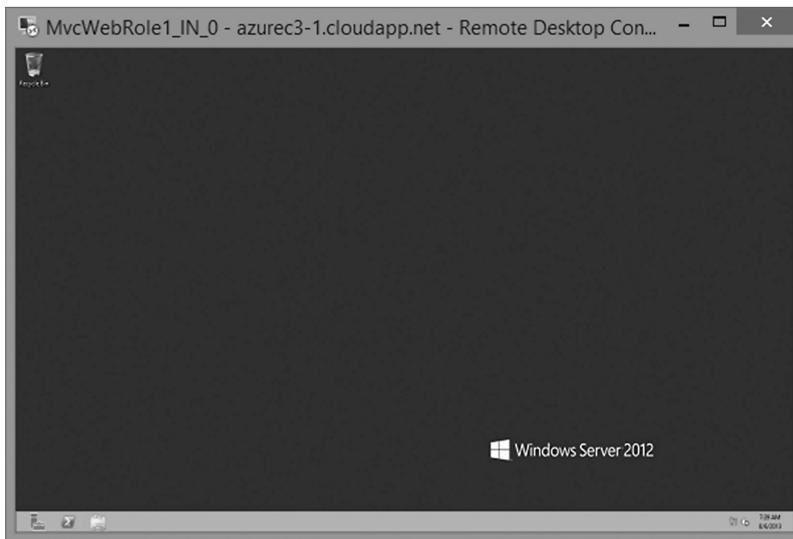


Figure 3.23 Remote desktop connection to role instance.

the **CONNECT** icon on the command bar to open the remote desktop to the virtual machine (Figure 3.22).

Once connected to the remote desktop, you can observe that your role instance is running on a Windows Server 2012 virtual machine. In the next section, we introduce how to choose the operation system on the virtual machines (Figure 3.23).

3.4 Cloud Service Deployments and Upgrades

We have learned the basic deployment steps in Example 3.2. Now let us dig a little deeper in the deployment and upgrade processes.

3.4.1 Incremental Updates (Update Domain Walk)

Upgrading a cloud service in Visual Studio is very simple. All you need to do is to right-click on the cloud project, and select the **Publish** menu again. Then, on **Publish Microsoft Azure Application** dialog, click the **Publish** button to publish a new version of your cloud service. Finally, in **Deployment Environment In Use** message dialog, click the **Replace** button (Figure 3.24).

By default, MACS performs an update domain walk (see Section 1.2.2) during upgrade. Each cloud service has 5 update domains by default, and role instances are evenly distributed into these update domains. For example, if your cloud service role has 3 instances, then 3 out of 5 update domains will be used, with 1 instance in each update domain. On the other hand, if your cloud service role has 11 instances, then 4 update domains will have 2 instances, and 1 update domain will have 3 instances. The update domain walk ensures your cloud service has at least one group of running instances during upgrades (assuming all your cloud service roles have at least 2 instances), hence the availability of the service is protected—this is the so-called zero-downtime upgrades. One problem of this upgrade strategy is speed (especially when you have many update domains). Another problem is that at any given moment during upgrades, you may have two versions of instances running at the same time. We discuss the second problem in Section II of this book. In Section 3.4.2, we introduce a different upgrade method to improve speed.

3.4.2 Simultaneous Updates

Instead of the update domain walk, you can choose to update all instances simultaneously. Obviously, this update method provides faster speed, but will cause service interruption during upgrades. To use simultaneous updates, on **Publish Microsoft Azure Application** dialog, go to the **Advanced Settings** tab, check the **Deployment update** checkbox, and then click on the **settings** link besides the checkbox, as shown in Figure 3.25.

On **Deployment Settings** dialog, select the simultaneous update option, and then click **OK** to continue (Figure 3.26).

3.4.3 Multiple Deployment Environments

MACS provides two deployment environments—production and staging—for cloud services. In previous examples, all our operations were performed directly on the production



Figure 3.24 Deployment Environment In Use prompt.

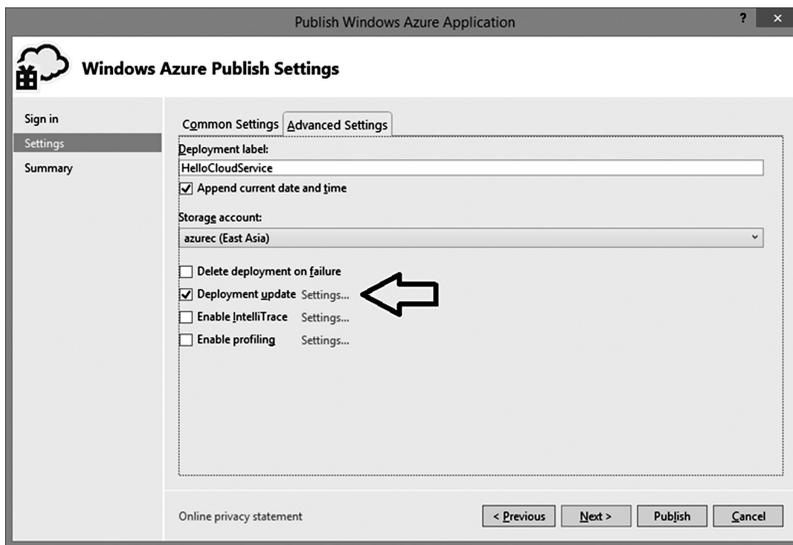


Figure 3.25 Deployment method settings link.

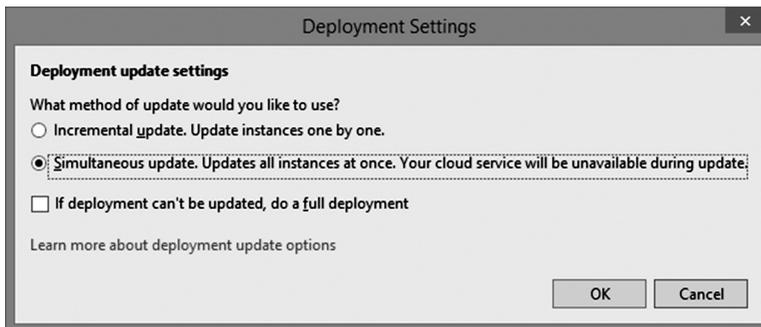


Figure 3.26 Simultaneous update option.

environment. However, in most cases, you would like your new version to go through a series of tests before it can be deployed on to your production environment. The staging environment provides a parallel environment to your production environment, allowing you to test new bits in an equivalent environment before release. Once you are satisfied with the new version, you can swap the two environments by a process called VIP swap. VIP swap promotes your staging environment to production, and makes your production environment staging for future versions. MACS defines a public virtual IP address (VIP) for each deployment environment. Production VIP is mapped to *[cloud service name].cloudapp.net*. Staging VIP is mapped to a *[cloud service identifier].cloudapp.net*. The VIP swap simply switches the DNS mapping between the two IP addresses.

You can choose different environments during deployments, as shown in Figure 3.27.

On Microsoft Azure Management Portal, you can use the INSTANCE page to inspect the two environments separately.

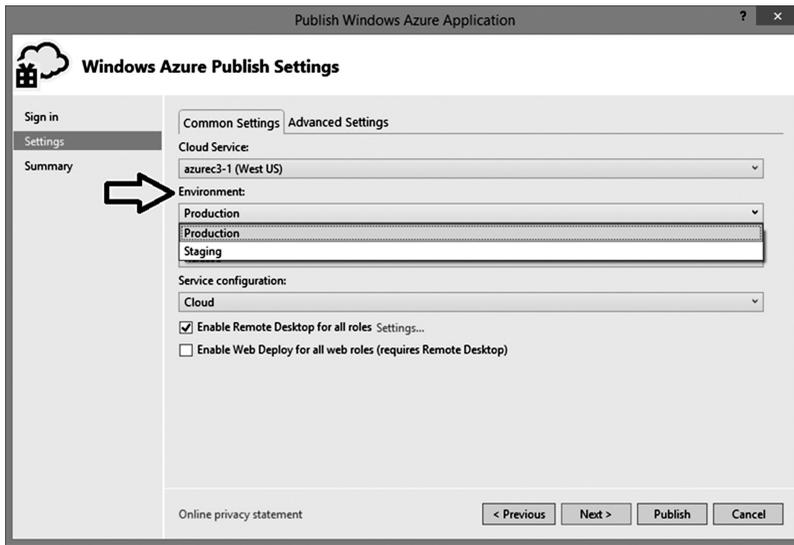


Figure 3.27 Choose deployment environment.

Example 3.3: Cloud Service Deployments and VIP Swaps on Management Portal

Difficulty: *

In Section 3.3, we introduced how a cloud service was packaged into a .cspkg file and a .cscfg file during the deployment process. Visual Studio automates the packing and uploading processes for us so we do not have to deal with files directly. On the other hand, you can use Microsoft Azure Management Portal to manually upload and deploy these two files. I assume you have made some changes to the cloud service in Example 3.1 and want to deploy the new version to the staging environment. Here are the steps:

1. Launch Visual Studio as an administrator. Open the solution of Example 3.1.
2. Right-click the cloud service project, and choose the Package menu (Figure 3.28).
3. On **Package Microsoft Azure Application** dialog, click on the **Package** button. We will introduce the options on this dialog in the next section (Figure 3.29).
4. The results of this operation are the .cspkg file and .cscfg file (Figure 3.30).
5. Open Microsoft Azure Management Portal. Open the **INSTANCE** page of your cloud service, and then click the **STAGING** link to switch to the staging environment page. You can see we have not deployed any versions to staging yet. Click on the **UPLOAD** icon on the command bar to start deployment (Figure 3.31).
6. On **Upload a package** dialog, enter a name for your deployment. We recommend using a name that reflects your version number for easy identification. Then, choose the local .cspkg file and .cscfg file you created in step 3. Finally, check the **Deploy even if one or more roles contain a single instance** check box, and click on the check button to complete the operation (Figure 3.32).

Note: Microsoft Azure's SLAs apply only to roles with more than one instance. Single-instance services are not protected by SLA.

7. After the deployment is done, you can observe the two environments on the cloud services' dashboard. Note that the URL to the staging environment differs from the production

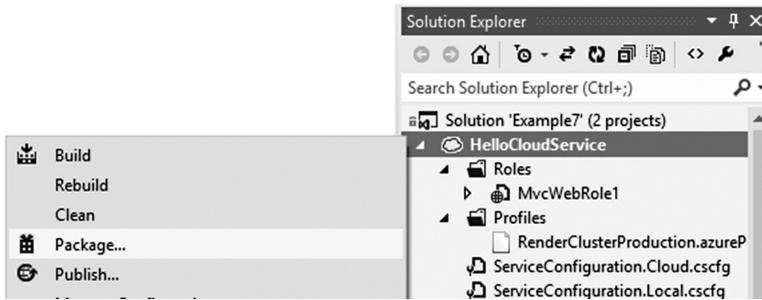


Figure 3.28 Package a cloud service in Visual Studio.

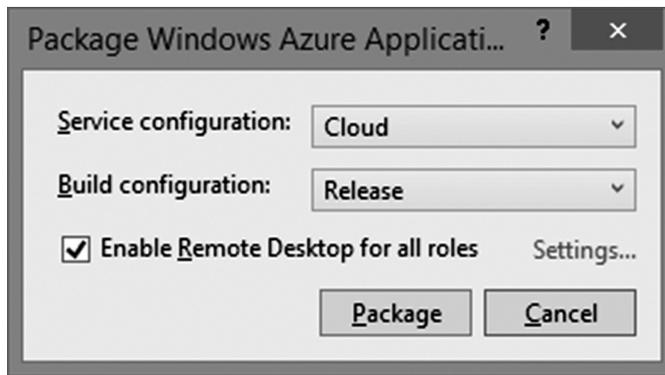


Figure 3.29 Package Microsoft Azure Application dialog.

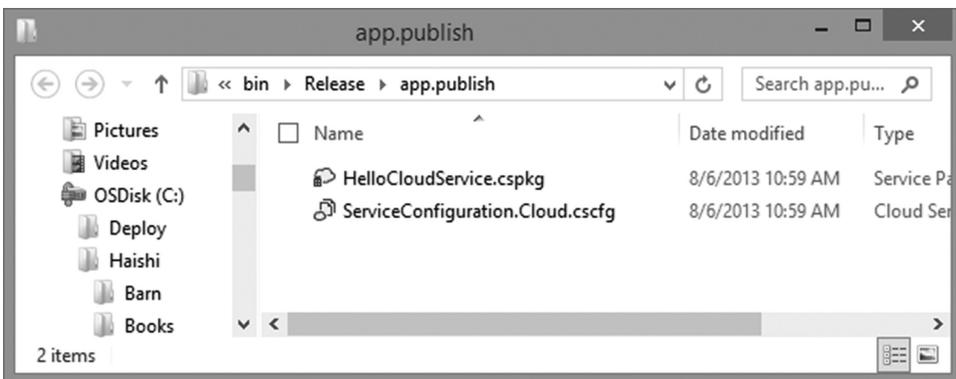


Figure 3.30 Packaged files.

URL. You can click on the staging URL to test your service on the staging environment (Figure 3.33).

- Now let us assume the new version has passed your tests on the staging environment. You can click on the SWAP icon to perform VIP swap between production and staging (Figure 3.34).
- Microsoft Azure Management Portal will present a brief summary of the swap operation. Click YES to swap (Figure 3.35).

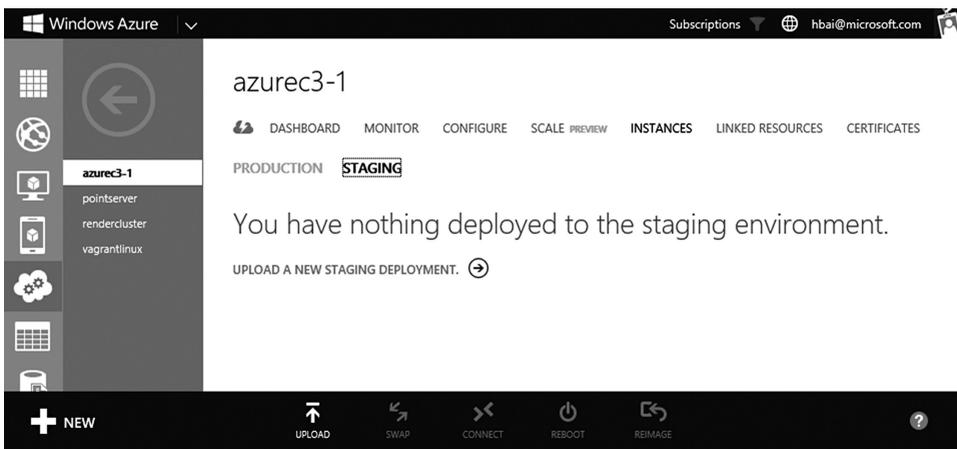


Figure 3.31 Staging page on Management Portal.

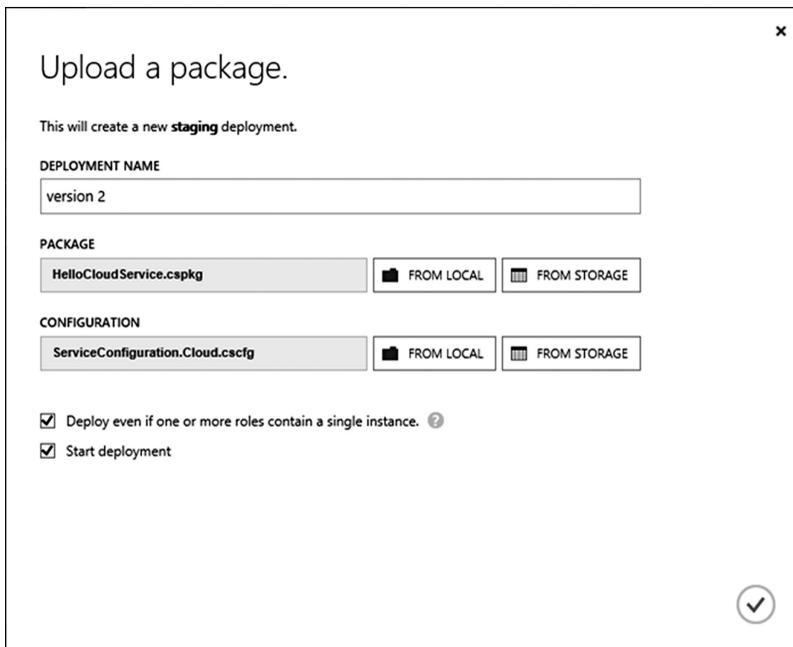


Figure 3.32 Upload a package dialog.

3.5 Instances and Load Balancing

The deployment process is a process of mapping a cloud service project to run virtual machines. During development, you define the logical definition of the cloud service and implement service logic details. During deployment, the logical definition and the compiled code are instantiated on actual virtual machines.

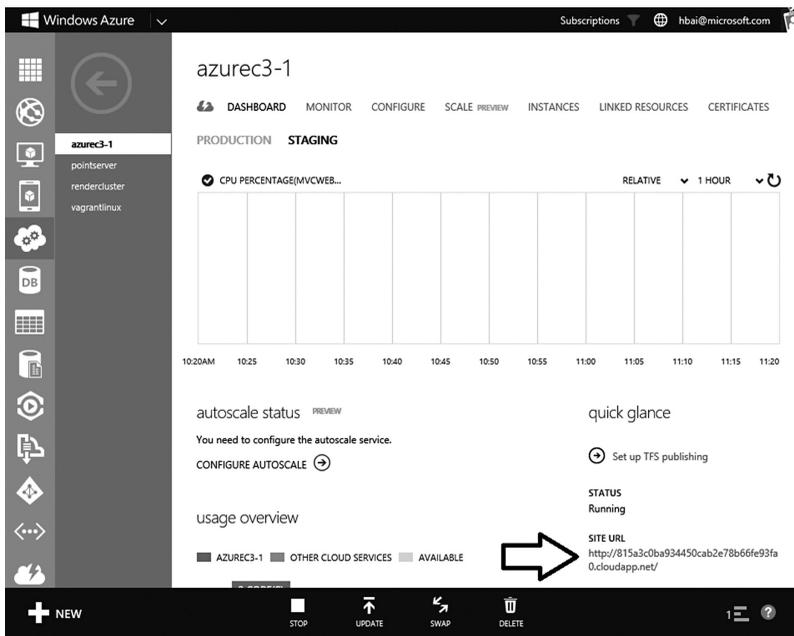


Figure 3.33 Staging environment dashboard on Management Portal.



Figure 3.34 The SWAP icon on the command bar.

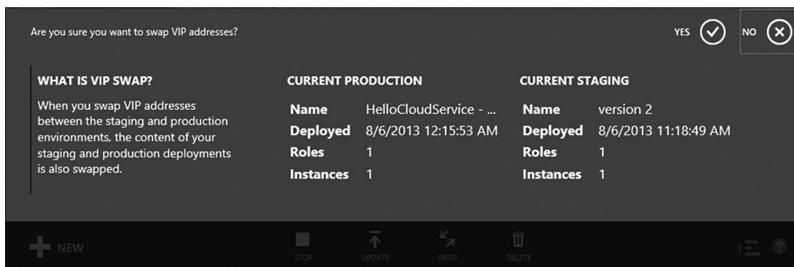


Figure 3.35 VIP swap summary.

3.5.1 Instances

An instance is the role code running on a virtual machine. For example, in the cloud service mentioned earlier, we defined a web role. During the deployment process, the role is instantiated as a virtual machine running the Windows operation system, and the role code is deployed on the IIS running on this virtual machine. A role can be instantiated into multiple instances as well. For instance, if we find that we need more capacity to satisfy customer needs, we can modify our cloud

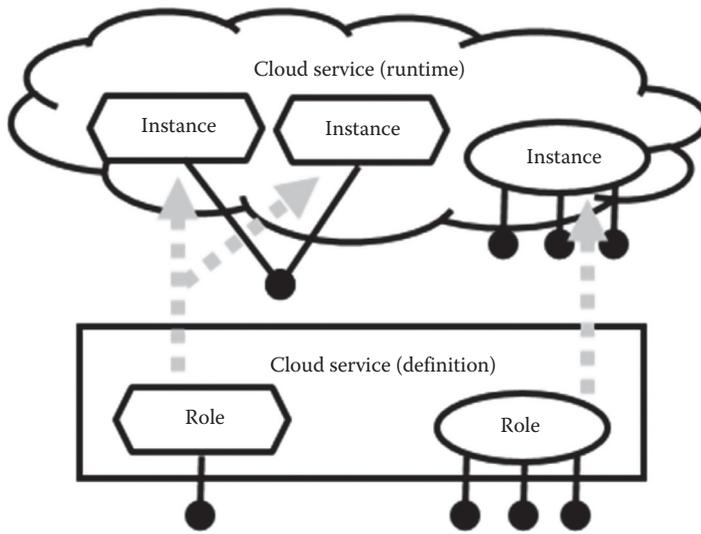


Figure 3.36 Mapping between roles and instances.

service definition and specify our web role to be deployed on two virtual machines. This is the so-called horizontal scaling. When we deploy this updated definition to Microsoft Azure, Microsoft Azure will launch two identical virtual machines, and deploy the same role code to these two machines. As depicted in Figure 3.36, the role to the left is mapped to two instances, while the role to the right is mapped to a single instance.

3.5.2 Load Balancing

Careful readers may have noticed that in Figure 3.36, when a role is mapped to multiple instances, these instances have the same endpoint, and share the system workload via this common endpoint. This kind of work sharing is also called load balancing. Load balancing is an important concept of system scalability. Readers should review Section 1.2.3 if needed.

How is load balancing achieved? By introducing load balancers. A load balancer connects to multiple role instances and evenly distributes system workloads to these instances. When a cloud service is deployed on Microsoft Azure, the endpoints defined on the roles are actually provided by load balancers. When a user request arrives, the load balancer forward the request to the corresponding virtual machine via its private network address. On the left side of Figure 3.37, we have a single web role in a cloud service, with the following address: <http://samplesite.haishibai.com>. When the service is deployed to Microsoft Azure, MACS starts two instances (virtual machines) for the role based on our service definition, and joins the two instances to a load balancer. The load balancer provides the <http://samplesite.haishibai.com> endpoint, and evenly distributes user requests to these two instances.

Note: A common algorithm used in load balancing is round-robin, which means to send jobs to participants in turn. In other words, all participating virtual machines are given the

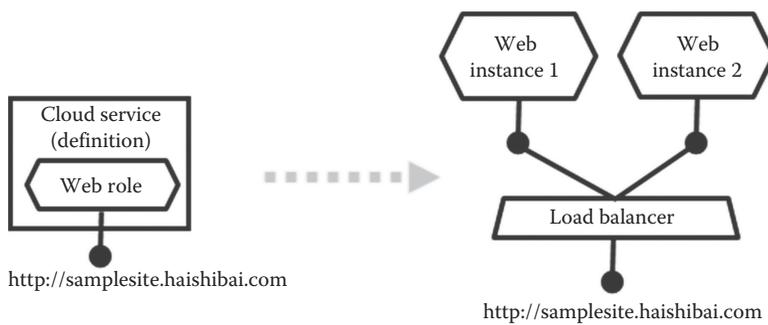


Figure 3.37 Cloud service load balancing.

same number of requests. Such algorithms often assume that all the virtual machines are homogeneous, and they do not consider the actual loads on these machines. Obviously, such assumptions greatly simplify the algorithm. To put the algorithm into a formula, we can use

$$i = (i + 1) \bmod n$$

where i is the index of the next virtual machine to be picked, and n is the total number of virtual machines.

It is very easy to configure horizontal scaling in Visual Studio. We will learn the steps in the next example.

Example 3.4: Horizontal scaling of Cloud Services

Difficulty: **

1. Launch Visual Studio as an administrator. Open the solution file in Example 3.1.
2. Before we scale the service, let us modify the source code to display the current instance id on the web page so that we can observe which instance is serving the current page.
3. In the web role project, open **HomeController.cs** under the **Controllers** folder. We will modify its **Index()** method and use the **RoleEnvironment** class, which is provided by **Microsoft.WindowsAzure.Service Runtime.dll** assembly from Microsoft Azure SDK, to retrieve the identifier of the current instance:

```
using Microsoft.WindowsAzure.ServiceRuntime;
...
public ActionResult Index()
{
    ViewBag.Message = "Current instance": + RoleEnvironment.
        CurrentRoleInstance.Id;
    return View();
}
```

4. Save changes to **HomeController.cs**.
5. Modify the **Views\Home\Index.cshhtml** file to display the message:

```
...
<div class="jumbotron">
  <h1>ASP.NET</h1>
  <p class="lead">@ViewBag.Message</p>
  <p><a href="http://asp.net" class="btn btn-primary btn-
large">Learn more &raquo;</a></p>
</div>
...
```

6. Then, double-click the web role to open its **Properties** page (Figure 3.38).
7. On the **Properties** page, you can scale your web role by simply changing the **Instance count** field. In addition, you can vertically scale the service by change virtual machine sizes. Here we will increase the instance count to 2, and then press **Ctrl + S** to save the file (Figure 3.39).
8. Redeploy the service. After the deployment is done, open the site in a browser. You can see the current instance id on the page. Figure 3.40 shows that the current page is rendered by the **WebRole1_IN_0** instance. Note that the result you observe may be different, as two instances have equal opportunities to handle the request.
9. Refresh the page several times. You will notice that sometimes the page is provided by instance **WebRole1_IN_0**, while at other times the page is provided by **WebRole1_IN_1**. This shows that the load balancer is at work.

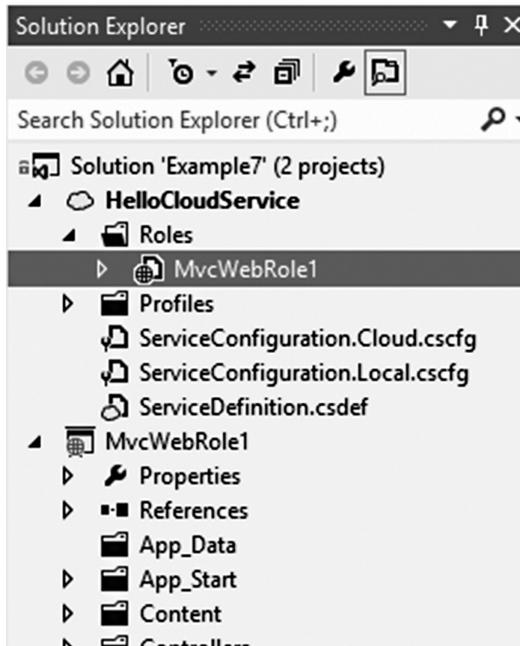


Figure 3.38 Web role in the cloud service.

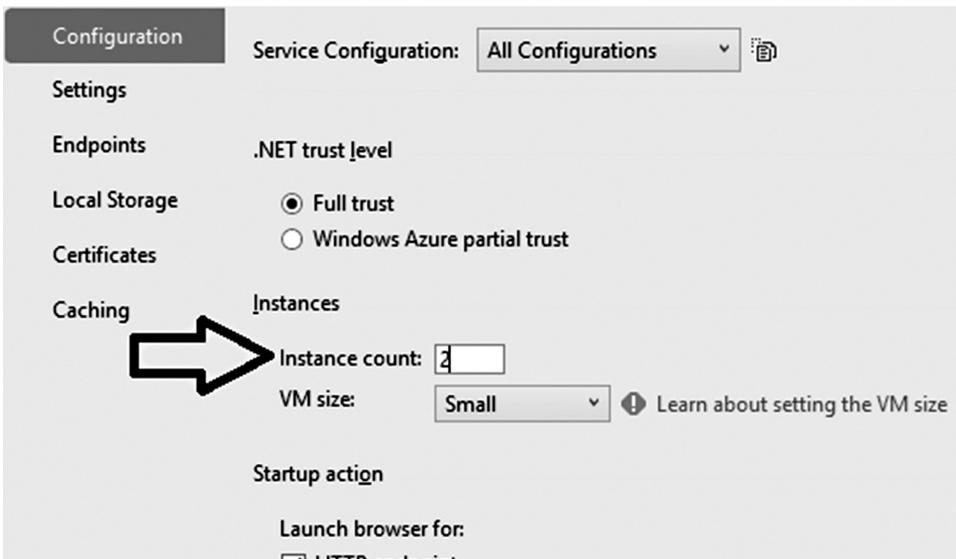


Figure 3.39 Instance count on Properties page.

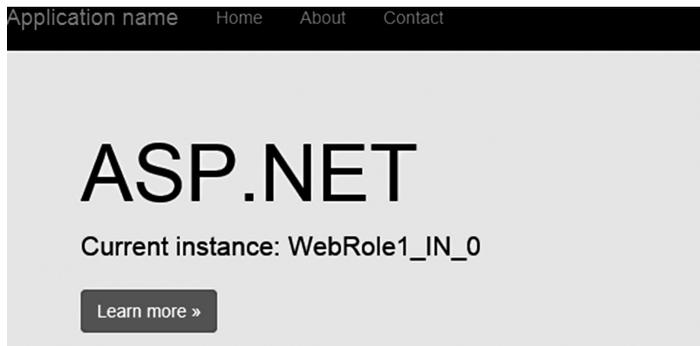


Figure 3.40 Web role id displayed on page.

3.6 Configuration File and Definition File

We have already learned that the result of cloud service packaging is a .cspkg file and a .cscfg file. The .cspkg file is a standard zip file, which contains compiled role code and content files, as well as the .csdef file from the cloud service project. The .cscfg file comes from the .cscfg file in the cloud service project. The packaging process can be simply illustrated in Figure 3.41.

Then, what are the .csdef file and the .cscfg file? First, let us examine their locations in a cloud service project. Figure 3.42 shows that the cloud service has one .csdef file and two .cscfg files. All these files are XML files, and together they define the topology of the cloud service as well as its configurations. For example, the instance count we just modified is recorded in the .cscfg file.

Now let us look into the structures of these files.

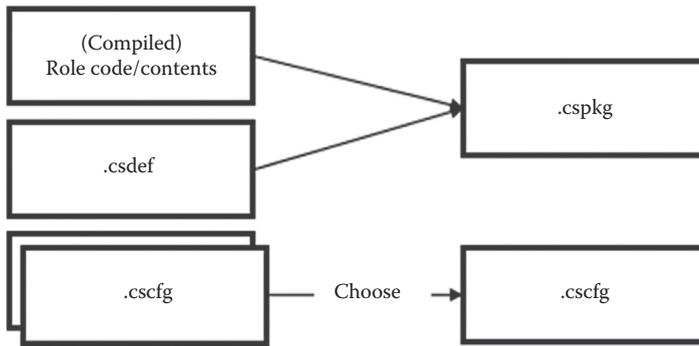


Figure 3.41 Cloud service packaging process.

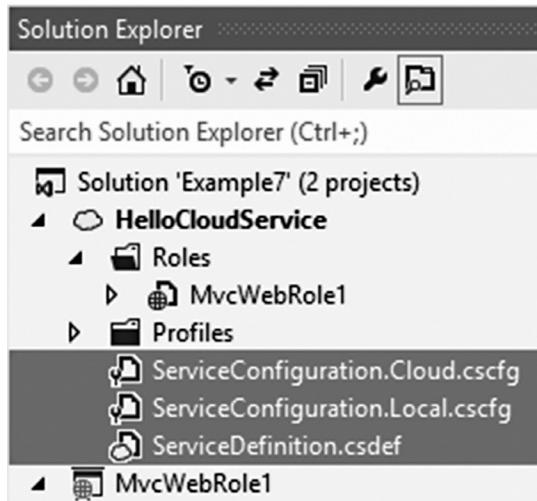


Figure 3.42 Locations of .csdef file and .cscfg file.

3.6.1 Cloud Service Definition File (.csdef)

The cloud service definition file defines the topology of a cloud service, including its roles, endpoints, and supported settings (actual setting values are saved in the .cscfg file). The schema of this XML document is fairly straightforward. For example, the service definition file for a cloud service containing a single web role looks as shown in Code List 3.1.

Code List 3.1 defines a web role (line 1), which has an endpoint based on HTTP, port 80 (line 10). Although you can directly modify this file, we suggest you use graphic UI to edit this file when possible to avoid possible mistakes caused by manual changes.

Because of limitations of space, we will not go through the entire schema here. The only attribute we want to mention here is the **upgradeDomainCount** attribute on the root **<ServiceDefinition>**

CODE LIST 3.1 CLOUD SERVICE DEFINITION FILE EXAMPLE

```

1: <WebRole name="MvcWebRole1" vmsize="Small">
2:   <Sites>
3:     <Site name="Web">
4:       <Bindings>
5:         <Binding name="Endpoint1" endpointName="Endpoint1" />
6:       </Bindings>
7:     </Site>
8:   </Sites>
9:   <Endpoints>
10:    <InputEndpoint name="Endpoint1" protocol="http"
11:      port="80" />
12:  </Endpoints>
13:  <Imports>
14:    ...
15:  </Imports>
16: </WebRole>

```

element. You can control the number of update domains (see Section 3.4.1) using this attribute. The default value of this attribute is 5.

Note: Refer to <http://msdn.microsoft.com/en-us/library/windowsazure/ee758711.aspx> for the complete schema of this file.

3.6.2 Cloud Service Configuration File (.cscfg)

You can add multiple service configuration files (.cscfg) to a cloud service project. When you use Microsoft Azure SDK to create a new cloud service project, the toolkit automatically adds two configuration files, which are *Cloud.cscfg* and *Local.cscfg*. Multiple configuration files allow you to use different configuration settings for different environments, such as local environment, staging environment, and production environment. When you package a cloud service, you can choose the configuration file you want to use (see step 3 in Example 3.3).

The .cscfg file specifies instance counts, settings, as well as digital certificates used by the roles. Let us study the file schema in detail, starting from the root element:

- Root: ServiceConfiguration

The attributes on this element are listed in Table 3.1.

In addition to these attributes, the element can contain one or more role elements, and an optional network configuration element (NetworkConfiguraiton).

- Role element

Attributes on the role element are listed in Table 3.2.

The role element supports several child elements. Here we will pick several frequently used ones.

- Instances element

The Instances element has a single attribute on it (Table 3.3).

Table 3.1 ServiceConfiguration Attributes

<i>Attribute</i>	<i>Type</i>	<i>Description</i>
serviceName	String	Name of the cloud service. This name has to match with the one in the service definition file.
osFamily	String	<i>[Optional]</i> OS family of the virtual machine. Currently supported values are: 1—Use Windows Server 2008 SP2 compatible OS (this is the default value) 2—Use Windows Server 2008 R2 compatible OS 3—Use Windows Server 2012 compatible OS
osVersion	String	<i>[Optional]</i> Version number of the virtual machine OS. The version you pick has to be compatible with Microsoft Azure SDK. ^a You can specify an exact version number, or use "*" to represent the latest version. When you use "*" as version number, Microsoft Azure will automatically upgrade your OS to the latest available releases (within the same OS family) when maintaining your virtual machines. We generally recommend using "*" as a version number. If you want to use a specific version, you need to set the string to: WA-GUEST-OS-M.m_YYYYMM-nn where M.m is the major version and minor version; YYYY is year; MM is month; and nn is a sequence number. For example, WA-GUEST-OS-1.17-201112-01. Refer to footnote "a" for a detailed version list. <i>Note:</i> This list changes over time.
schemaVersion	String	<i>[Optional]</i> Schema version of the .cscfg file. Because you can have multiple versions of Microsoft Azure SDKs installed on the same machine, this attribute helps Visual Studio to correctly handle definition files from different SDK versions.

^a <http://msdn.microsoft.com/en-us/library/windowsazure/ee758710.aspx>.

Table 3.2 Role Attributes

<i>Attribute</i>	<i>Type</i>	<i>Description</i>
name	String	Name of the role. This name has to match with the one in the cloud service definition file.
vmName	String	Name of the virtual machine. This value will be used as the DNS name of the virtual machine. This value follows the RFC1123 standard, but is constrained to 10 characters.

Table 3.3 Instances Attributes

<i>Attribute</i>	<i>Type</i>	<i>Description</i>
count	Integer	Number of instances

– ConfigurationSettings element

This element holds custom settings of a cloud service. For example, the following ConfigurationSettings element defines two settings: *Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString* (this is added by default. We will come back to this in Section 4.6) and a custom-defined *MySetting*:

```
<ConfigurationSettings>
  <Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.
    ConnectionString"
    value="UseDevelopmentStorage=true" />
  <Setting name="MySetting" value="Value of my setting" />
</ConfigurationSettings>
```

You should avoid editing this file manually, because the setting names in this file have to match with the ones defined in the .csdef file. Manual edits are very error-prone. Instead, you should use the role's Properties page to edit these settings, as shown in Figure 3.43.

Using this UI, you can edit the setting values simultaneously for all configuration files, or use the **Service Configuration** dropdown box to select the specific configuration file to edit. All your edits will be kept in sync with the .csdef file. In addition, if you select **<Manage...>** from the dropdown box, you can create new configuration files by making copies of existing ones. As you can see, using the UI is a much easier and effective way for editing settings.

In your code, you can read these settings using the *Microsoft.WindowsAzure.CloudConfigurationManager* class. For example,

```
using Microsoft.WindowsAzure;
...
var value = CloudConfigurationManager.GetSetting("MySettings");
```

The screenshot shows the 'Settings' page in the Azure Management Console. The 'Service Configuration' dropdown is set to 'All Configurations'. Below the dropdown, there are buttons for 'Add Setting' and 'Remove Setting'. A table lists various settings, including 'MySetting' which is highlighted. The table has columns for Name, Type, and Value.

Name	Type	Value
Microsoft.WindowsAzure.Plugins.Diagnostics.Connec...	Connection String	<Select Configuration>
Microsoft.WindowsAzure.Plugins.RemoteAccess.Enabled	String	true
Microsoft.WindowsAzure.Plugins.RemoteAccess.Acco...	String	haishi
Microsoft.WindowsAzure.Plugins.RemoteAccess.Acco...	String	MIIBnQYJKoZIhvcNAQcDollBjJCCAYo
Microsoft.WindowsAzure.Plugins.RemoteAccess.Acco...	String	2014-08-06T23:59:59.0000000-07:00
Microsoft.WindowsAzure.Plugins.RemoteForwarder.En...	String	true
MySetting	String	Some value

Figure 3.43 Use Properties page to edit custom settings.

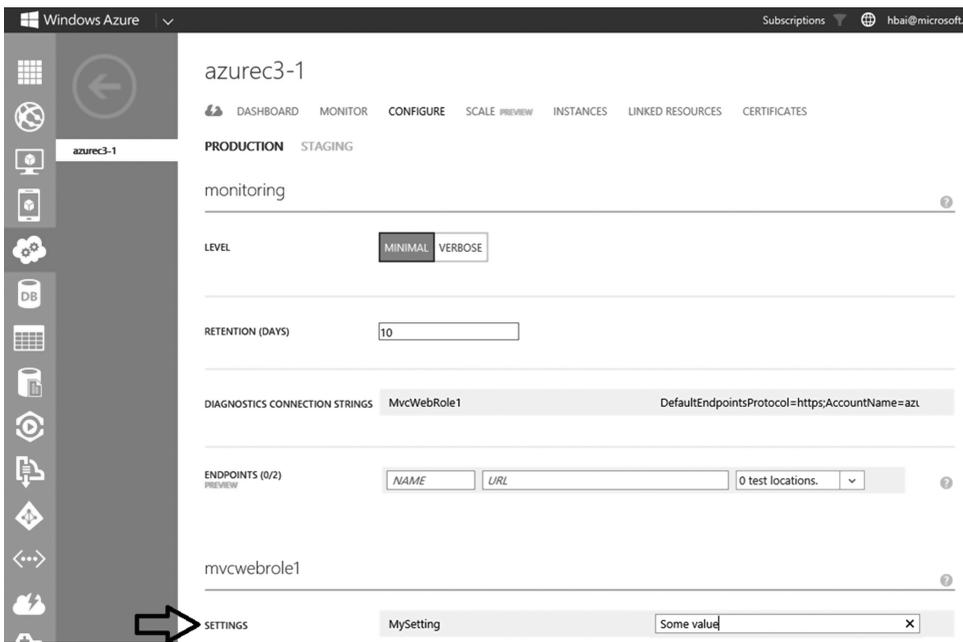


Figure 3.44 Manage settings on Microsoft Azure Management Portal.

Once the cloud service is deployed to Microsoft Azure, system administrators can modify these settings using Microsoft Azure Management Portal (Figure 3.44).

- NetworkConfiguration element

This element defines virtual network and DNS settings. We discuss this element in Section 4.3.

Note: Refer to <http://msdn.microsoft.com/en-us/library/windowsazure/ee758710.aspx> for a complete schema of this file.

3.7 Summary

In this chapter, we introduced basic concepts of cloud service, including roles, instances, and their relationship. Through a series of examples, we studied basic processes of cloud service development and deployment. We also experienced how to horizontally scale our cloud services and how to update cloud services using different methods. Finally, we introduced how to use multiple deployment environments, and how to define and use custom settings.

Chapter 4

Advanced Cloud Service

4.1 Endpoint Types

When we define a web role in a cloud service project, the development tool automatically defines an endpoint based on HTTP and port 80. This endpoint is a publicly accessible endpoint called Input Endpoint, and its DNS address is *[cloud service name].cloudapp.net*. This is why we can access the website via the following address: *http://[cloud service name].cloudapp.net* (default port 80 is omitted).

As mentioned earlier, a cloud service role can have multiple endpoints. These endpoints can use different protocols. For example, many websites need to support HTTPS, and some other services may need to support TCP and UDP, etc. In addition to Input Endpoints, you can also find Internal Endpoints and InstanceInput Endpoints on cloud services.

4.1.1 Input Endpoint

Input endpoints are public endpoints. Service consumers can access these endpoints at the following address:

[protocol]://[cloud service name].cloudapp.net:[port]

These endpoints are provided by load balancers. User requests are evenly distributed to all role instances. In other words, any two requests, no matter how close they are in time, might be routed to different instances. For example, displaying an HTML page with a single picture requires two HTTP GET requests, one to download HTML tags and the other to download image bits. These two requests may be served by different instances. Another example is that two consecutive AJAX calls may be handled by different instances as well. This is why websites should not save their states in memory, but should save states in a shared storage such as a database, a storage service, or a cache cluster.

4.1.2 Internal Endpoint

In Section 3.2, we introduced the concept that, as a container of roles, a cloud service also defines a security boundary. Within this security boundary, role instances can exchange data via Internal

Endpoints. There are two main differences between Input Endpoints and Internal Endpoints: first, Internal Endpoints are private to cloud role instances only, while Input Endpoints are publicly accessible; second, Internal Endpoints do not go through load balancers because they are designed for instance-level communications. We have already learned that you can address Input Endpoints via a cloud service's DNS addresses. However, when we use Internal Endpoints, how do we find the address of a specific instance that uses dynamic IP? We will explain the process with an example in Section 4.3.

4.1.3 InstanceInput Endpoint

InstanceInput Endpoints are special in several different ways:

- InstanceInput Endpoints are publicly accessible endpoints.
- InstanceInput Endpoints do not go through load balancers.
- An InstanceInput Endpoint has an associated port range. Service consumers can directly access different instances by picking different ports in this range. For example, a cloud service role defines an HTTP-based InstanceInput endpoint with port range from 8080 to 8084, and the role has five instances. Then, service consumers can use `http://[cloud service name].cloudapp.net:8080` to access the first instance, `http://[cloud service name].cloudapp.net:8081` to access the second instance, `http://[cloud service name].cloudapp.net:8082` to access the third instance, and so on.

Now, let us study endpoints with an example.

Example 4.1: Configure an HTTPS Endpoint for a website

Difficulty: ****

Security Socket Layer (SSL) is a common protocol for secured data exchange over the Internet. In this example, we define an HTTPS endpoint for a website. To complete this exercise, we need to acquire a digital certificate first. Of course, you can request a certificate from a Certificate Authority (CA), but here we use a self-signed certificate.

1. Launch **Developer Command Prompt for VS2012** as an administrator.
2. Go to the folder where you want to save your certificate.
3. Use the command

```
makecert -r -pe -n "CN=azurec3-1.cloudapp.net" -sky 1 "azurecert.
cer" -sv "azurecert.pvk" -ss My -sr LocalMachine
```

to create a self-signed certificate. During the process you will be asked to enter a password for the private key three times, as shown in Figure 4.1.

4. Then, use the command

```
pvk2pfx -pvk "azurecert.pvk" -spc "azurecert.cer" -pfx "zurecert.
pfx" -pi [cert password]
```

to convert the certificate into a pfx format.

Now, we will add the certificate to our cloud service project, and define a new HTTPS for the web role.



Figure 4.1 Create a self-issued certificate.

5. Launch Visual Studio as an administrator. Open the solution in Example 3.1.
6. In cloud service project, double click the web role to open its Properties page. Then, click the **Certificate** tab to the left of the screen to open the certificate configuration page. Click on the **Add Certificate** link to add a new row to the table. In the newly created row, enter **MyCertificate** as **Name**, and pick **LocalMachine** as **Store Location**. Finally, click on the [...] button in the **Thumbprint** column, pick the certificate you just created, and click the **OK** button, as shown in Figure 4.2.
7. Press Ctrl + S to save the file.
8. Click on the **Endpoints** tab to switch to the endpoint configuration page. Then, click on the **Add Endpoint** link. In the new row, enter **SSLEndpoint** as **Name**, pick **https** as **Protocol**, enter **443** as **Public Port**, and pick **MyCertificate** for **SSL Certificate Name**, as shown in Figure 4.3.
9. Press Ctrl + S to save the file.

Before we deploy the updated cloud service to Microsoft Azure, we need to upload the certificate used by the web role to Microsoft Azure.

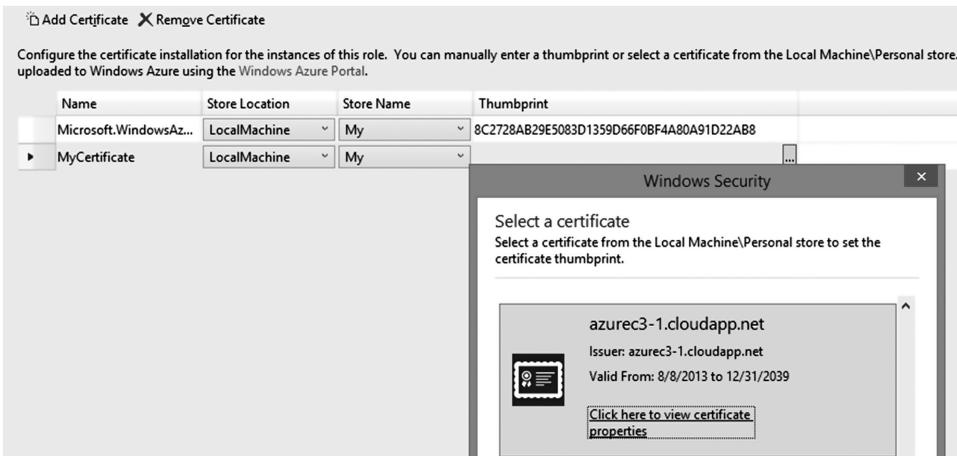


Figure 4.2 Add a certificate to the web role.

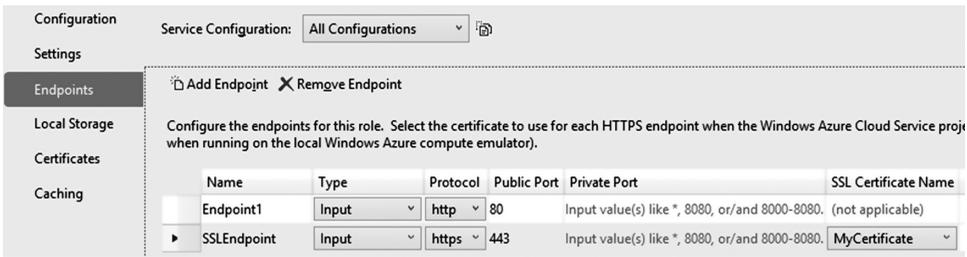


Figure 4.3 Add HTTPS endpoint.

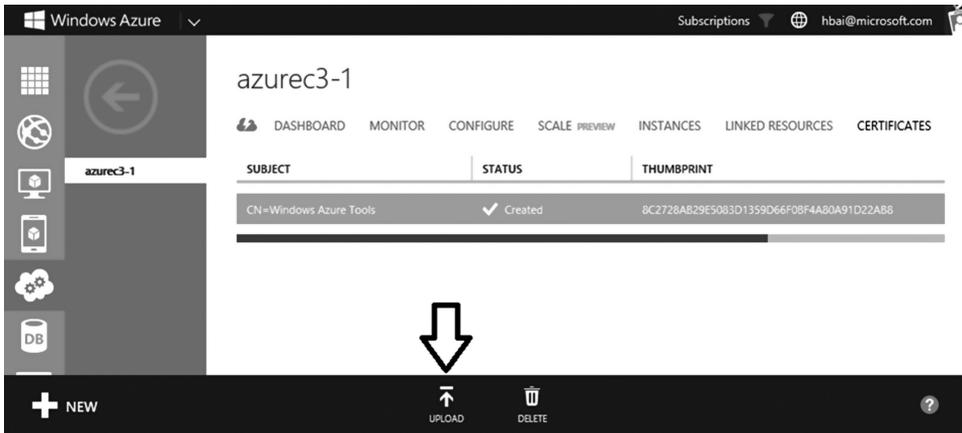


Figure 4.4 Upload icon on the command bar.

10. Sign in to Microsoft Azure Management Portal. Select the cloud service you want to update. Then, go to its **CERTIFICATES** page. Here, click the **UPLOAD** icon on the command bar, as shown in Figure 4.4.
11. On **Upload certificate** dialog, pick the .pfx file you created in step 3. Enter the password, and then click the check button to upload the certificate, as shown in Figure 4.5. Now we can redeploy the cloud service and test the HTTPS endpoint.
12. Redeploy the cloud service in Visual Studio.
13. Once the deployment is done, open the browser, and enter the address *https://[cloud service name].cloudapp.net* to open the website.
14. Because we are using a self-signed certificate, the browser will throw a certificate warning, as shown in Figure 4.6. If you want to eliminate this warning, you have to import the certificate to **Trusted Root Certification Authorities**.
15. Here, simply click the **Continue to this website** link to continue to the site, as shown in Figure 4.7.

4.2 Worker Role

As a container, a cloud service can hold different types of roles. Here, we will introduce a new role type, the worker role. Different from web roles, which provide website services, worker roles are

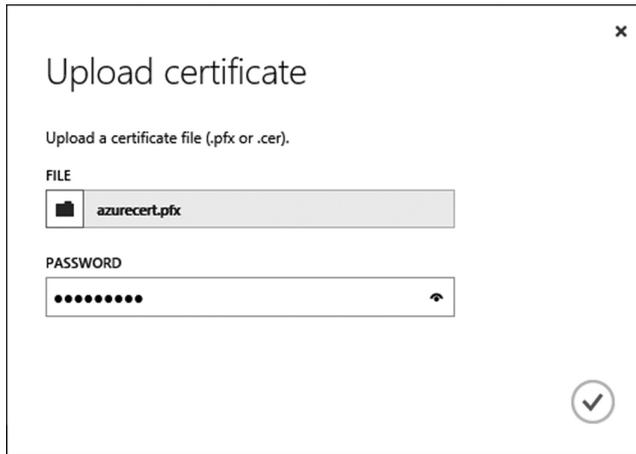


Figure 4.5 Upload certificate dialog.

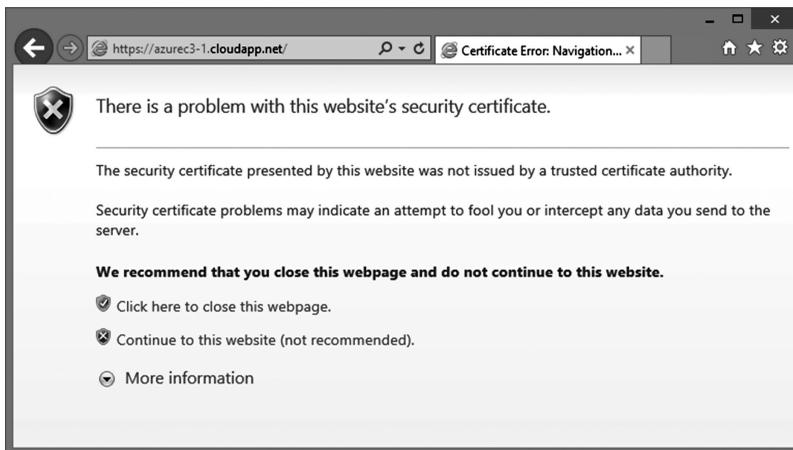


Figure 4.6 Certificate warning.

often used to provide backend services. For example, in a multitier application, you can use a web role for a presentation layer, and a worker role to implement business logics. This kind of separation of concerns is one of the basic principles in designing maintainable, extensible systems. We discuss multitier applications in detail in Chapter 8.

Of course, the responsibilities of a web role or a worker role are not fixed and can vary from project to project. For example, a worker role can directly take user inputs by defining Input Endpoints. A web role may provide a REST-ful API or a SOAP-based web service without offering any user interfaces. In other words, the types of service provided by a role are not constrained by role types. It is just that web roles are optimized for web sites and worker roles are optimized for backend services.

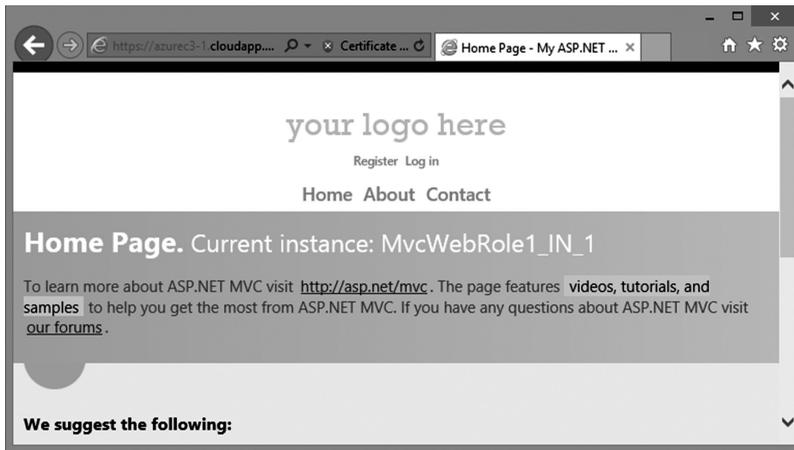


Figure 4.7 Open a website via HTTPS endpoint.

4.2.1 Worker Role Application Scenarios

Common application scenarios of worker roles include the following:

- Processing jobs sent from a web role.

When we design a cloud service, we should separate complex tasks from the web role to ensure responsiveness of the user interface. The web role can offload complex tasks to a worker role via direct or indirect communication, so it can free up its resources to accept more user requests. We discuss different communication options among roles in Section 4.3. In addition, once we separate the frontend and the backend, we can scale different layers separately in order to optimize resource utilization. For example, a system may need only 1 web role instance to take user inputs, but 10 worker role instances to process complex tasks. When system load decreases, we can scale the worker role down to 4–5 instances to save on renting costs. On the contrary, if we find the presentation layer too busy, we can spin up more web role instances to ensure the performance of the user interface.

- Running web services such as WCF services.

The worker role code runs as a separate process on a Windows virtual machine. So, in theory, most Windows-based services can be loaded to a worker role. For services that need prerequisites to be installed, we can use startup tasks, which we cover in Section 4.5. By directly exposing Input Endpoints, you can run various cloud-based web services, such as WCF services, on worker roles.

Now, let us study how to use a worker role with an example.

Example 4.2: A worker role with a UDP endpoint

Difficulty: ****

In this example, we will define a new cloud service with a worker role. The worker role will take user requests through a UDP endpoint.

1. Launch Visual Studio as an administrator. Create a new cloud service project.
2. On **New Microsoft Azure Cloud Service** dialog, add a **Worker Role** to the cloud service, as shown in Figure 4.8.

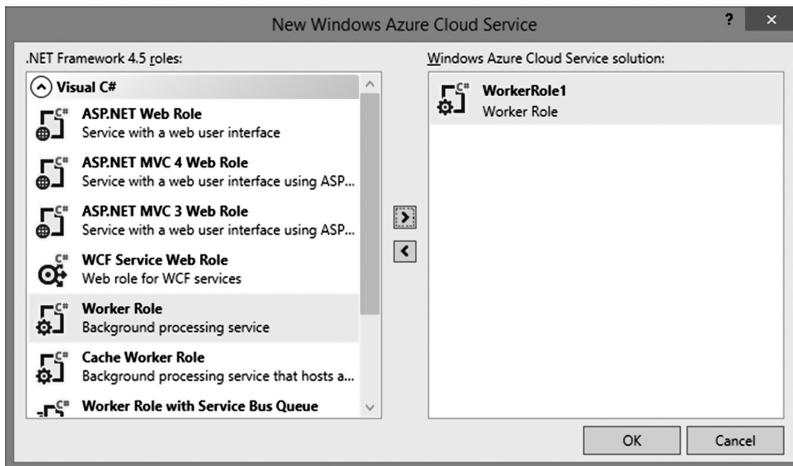


Figure 4.8 Add a worker role to a cloud service.

3. This operation defines a new worker role in your cloud service. The worker role is added to the solution as a separate class library with the name *WorkerRole1*. The class library contains only one *WorkerRole* class. When the worker role instance is started, Microsoft Azure will call its *OnStart()* method and then its *Run()* method. The default *Run()* method is nothing but an empty loop:

```
public override void Run()
{
    // This is a sample worker implementation. Replace with your
    // logic.
    Trace.TraceInformation("WorkerRole1 entry point called",
        "Information");

    while (true)
    {
        Thread.Sleep(10000);
        Trace.TraceInformation("Working", "Information");
    }
}
```

4. Modify the *Run()* method. We will listen to port 8080, attach current role instance id to the requests we get, and then send them back. The modified *Run()* method is shown as Code List 4.1.
5. In the cloud service project, double click on *WorkerRole1* to open its Properties page.
6. Click on the **Endpoints** tab to open the endpoint page. Click on the **Add Endpoint** link. Then, in the newly added record, enter **UDPEndpoint** as **Name**, select **udp** for **Protocol**, enter **8080** for **both Public Port** and **Private Port**, as shown in Figure 4.9. Press **Ctrl + S** to save changes.
7. Now let us add a test client. Right click the solution, and select the **Add→New Project** menu (see Figure 4.10).
8. On **Add New Project** dialog, select **Console Application** under the **Windows** category. Enter **TestClient** as project name, and click the **OK** button to continue, as shown in Figure 4.11.

CODE LIST 4.1 MODIFIED RUN() METHOD

```

public override void Run()
{
    var endpoint = new IPEndPoint(IPAddress.Any, 8080);
    var client = new UdpClient(endpoint);
    IPEndPoint sender = null;
    while (true)
    {
        try
        {
            var data = client.Receive(ref sender);
            var text = Encoding.UTF8.GetString(data);
            var bytes = Encoding.UTF8.GetBytes(string.Format
                ("Instance {0} plays back: {1}",
                 RoleEnvironment.CurrentRoleInstance.Id, text));
            client.Send(bytes, bytes.Length, sender);
        }
        catch
        {
            Thread.Sleep(1000);
        }
    }
}

```

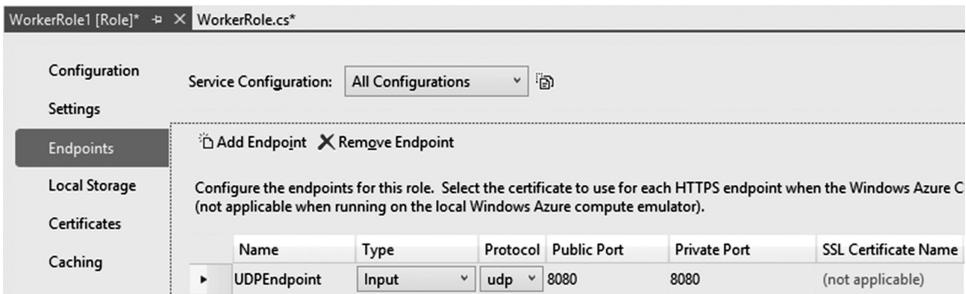


Figure 4.9 Add UDP endpoint.

9. Modify Program.cs in the TestClient project. The modified code is shown as Code List 4.2.
10. Now we are ready to test the solution locally. Right-click on the solution, and select the **Set Startup Projects** menu (Figure 4.12).
11. On the **Solution Property Pages** dialog, select the **Multiple startup projects** option. Then set both the cloud project (not the WorkerRole1 project) and the TestClient project to **Start**. Click the **OK** button to confirm, as shown in Figure 4.13.
12. Press F5 to start the test. If you see a Windows Security Alert (because the worker role process attempts to listen to 8080 port), click **Allow access** to continue, as shown in Figure 4.14.
13. Enter a couple of test strings in the console application, and observe feedback from the worker role, as shown in Figure 4.15. Enter an empty string to stop the test client. If you are interested, you can deploy the service to Microsoft Azure. By default, the worker role runs under an account without administrative privileges, which cannot listen to an arbitrary port. There are two ways to solve this problem. One is to modify ACL in the startup

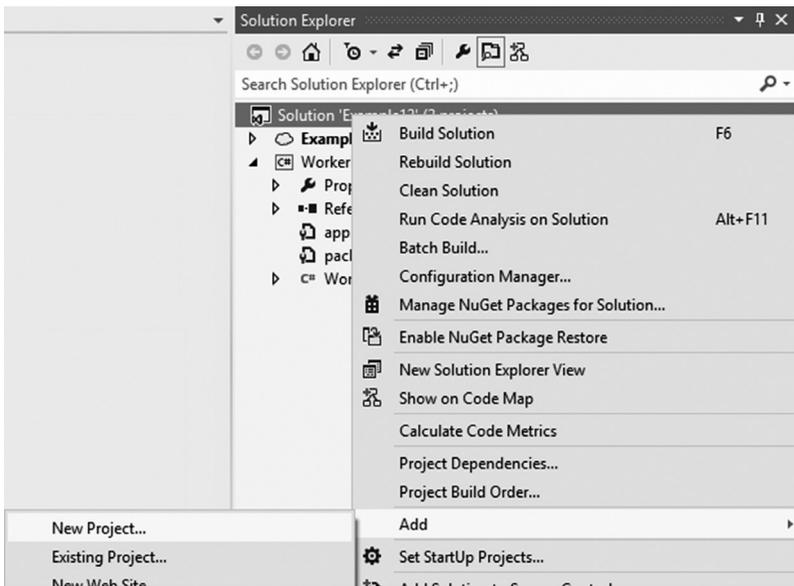


Figure 4.10 Add test client project.

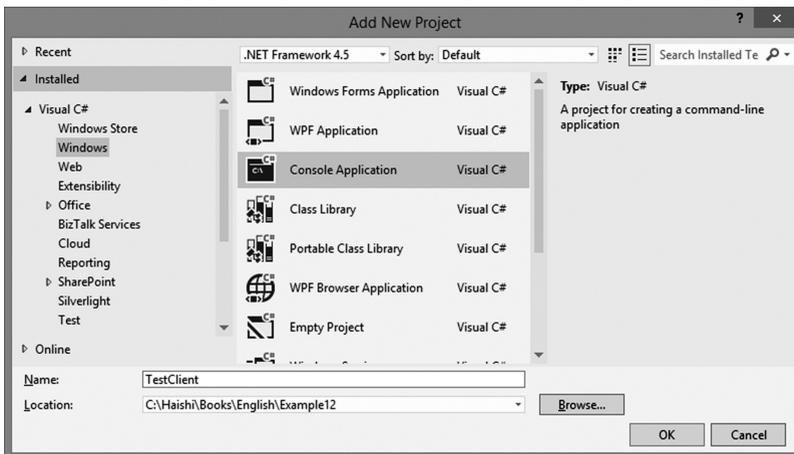


Figure 4.11 Create a new Console Application.

task to grant the account privilege to listen to the UDP protocol, 8080 port. An easier way is to modify the cloud definition file (.csdef) to launch the worker role with administrative privilege. Here, we use the second method.

14. Modify the ServiceDefinition.csdef file under the cloud service project. Add

```
<Runtime executionContext="elevated" />
```

under <WorkerRole> element. This allows the worker role process to be launched with administrative privileges.

CODE LIST 4.2 UDP TEST CLIENT

```

using System.Net.Sockets;
...
static void Main(string[] args)
{
    var address = IPAddress.Parse("127.0.0.1");
    var endpoint = new IPEndPoint(address, 8080);
    var socket = new Socket(AddressFamily.InterNetwork, SocketType.
        Dgram,
        ProtocolType.Udp);
    var buffer = new byte[1024];
    while (true)
    {
        Console.Write("Enter text to be sent: ");
        var payload = Console.ReadLine();
        if (string.IsNullOrEmpty(payload))
            return;
        var data = Encoding.UTF8.GetBytes(payload);
        socket.SendTo(data, endpoint);
        int size = socket.Receive(buffer);
        Console.WriteLine(Encoding.UTF8.GetString(buffer, 0, size));
    }
}

```

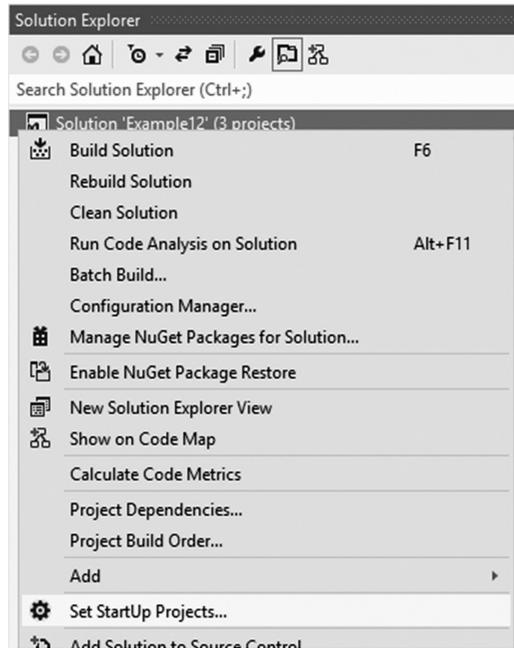


Figure 4.12 Set Startup Projects menu.

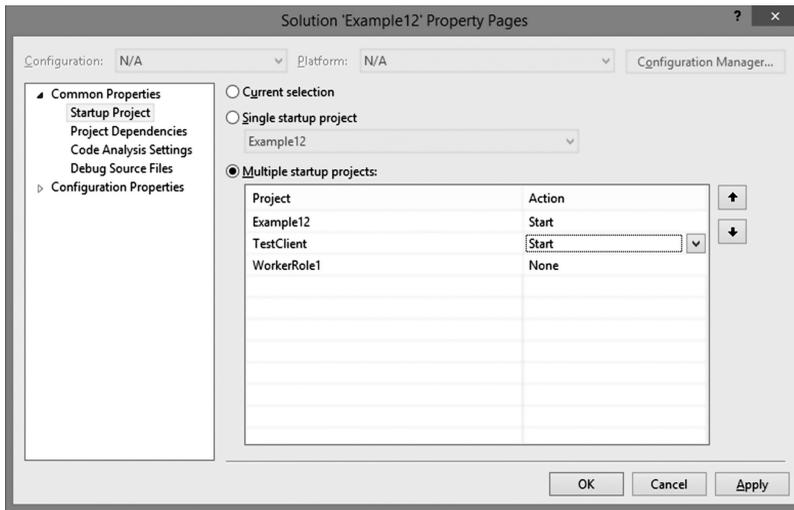


Figure 4.13 Set multiple startup projects.



Figure 4.14 Windows Security Alert.

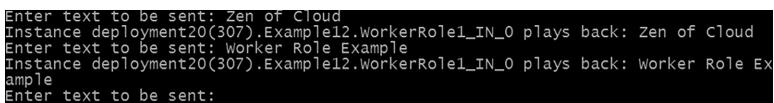


Figure 4.15 Test client running result.

15. Deploy the service to Microsoft Azure.
16. Modify the first line of the test client:

```
var address = Dns.GetHostAddresses("[cloud service name].cloudapp.net")[0];
```

17. Set the test application as a startup project, and then press F5 to test.

4.3 Inter-Role Communications

In the previous example, the worker role directly takes user inputs from an Input Endpoint. In reality, a more common topology is to use a web role to interact with users, and handle tasks that consume large amount of resources or CPU cycles on a worker role. Before we learn the exact techniques, let us summarize different options for inter-role communication.

4.3.1 Options for Inter-Role Communication

Common options for inter-role communication include the following:

- **Direct communication**
Roles can communicate with each other directly not only using their Input Endpoints, but also their Internal Endpoints. We will learn about using Internal Endpoints in Example 4.3.
- **Communication through a shared storage**
Roles can also exchange data through a shared storage, such as a database, a virtual disk, a distributed cache cluster, or a storage service. We introduce different storage services in Chapter 5 and caching in Chapter 11.
- **Communication through a job queue**
Another commonly used method for inter-role communication is to use a job queue. A job creator (such as a web role) adds jobs to a queue. A job processor (such as a worker role) reads jobs from the queue and handles them. Queues have several significant advantages compared to other communication techniques, which we discuss further in Chapter 15.
Now, let us implement a direct communication scenario. You will see an example of using queues in Chapter 15.

Example 4.3: Using a Web Frontend with a Backend Service

Difficulty: ***

Compared to the examples described earlier, this example has a more complex architecture, as we are going to implement a multitiered system. The system consists of a web role as the presentation layer and a worker role as the business layer. To simplify the process, instead of starting a new example, we will continue with Example 4.2. We will use a web frontend to replace the original console client. The modified system architecture is shown in Figure 4.16.

From a service consumer's perspective, the whole service only provides a single HTTP endpoint. Internals of the system are totally hidden from the consumer. In other words, as long as the service keeps its Input Endpoints (and API) stable, its consumers are not affected by any internal changes. Now let us study the exact process:

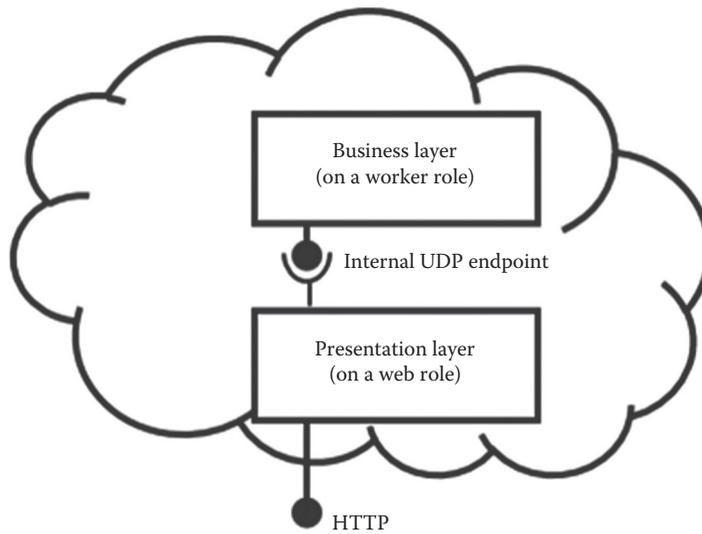


Figure 4.16 System architecture of Example 4.3.

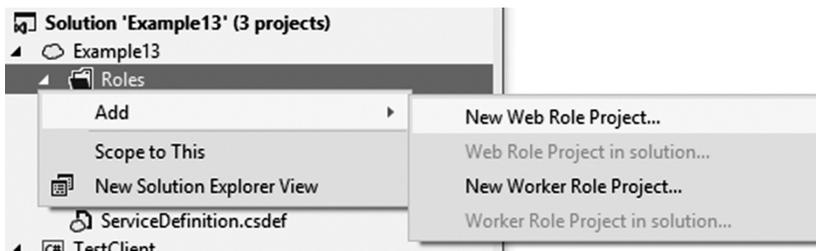


Figure 4.17 Add New Web Role Project menu.

1. Launch Visual Studio as an administrator. Open the solution in Example 4.2.
2. In **Solution Explorer**, expand the cloud service project. Then, right click on the **Roles** folder, and select **Add**→**New Web Role Project**, as shown in Figure 4.17.
3. Choose the **ASP.NET MVC Web Role** or **ASP.NET Web Role** template, and then click the **Add** button to add the new web role, as shown in Figure 4.18. This is also the common method to add a new role to an existing cloud service project.
4. Choose the **Internet Application** template, and then click the **OK** button, as shown in Figure 4.19. Depends on your ASP.NET MVC version, you may see a dialog with different options. Select “MVC” in this case.
5. Edit **index.cshtml** under the **Views\Home** folder in the new web role. We will implement a simple web page to send and receive messages from the backend server. The modified code is shown in Code List 4.3.

Note: Here, we use jQuery to write the client-side script. jQuery is a cross-browser JavaScript library that has gained great popularity in the past few years.

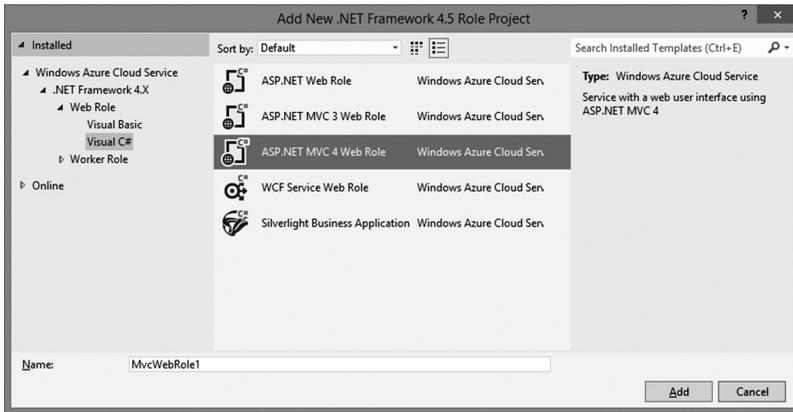


Figure 4.18 Add a new Web Role to a cloud service.

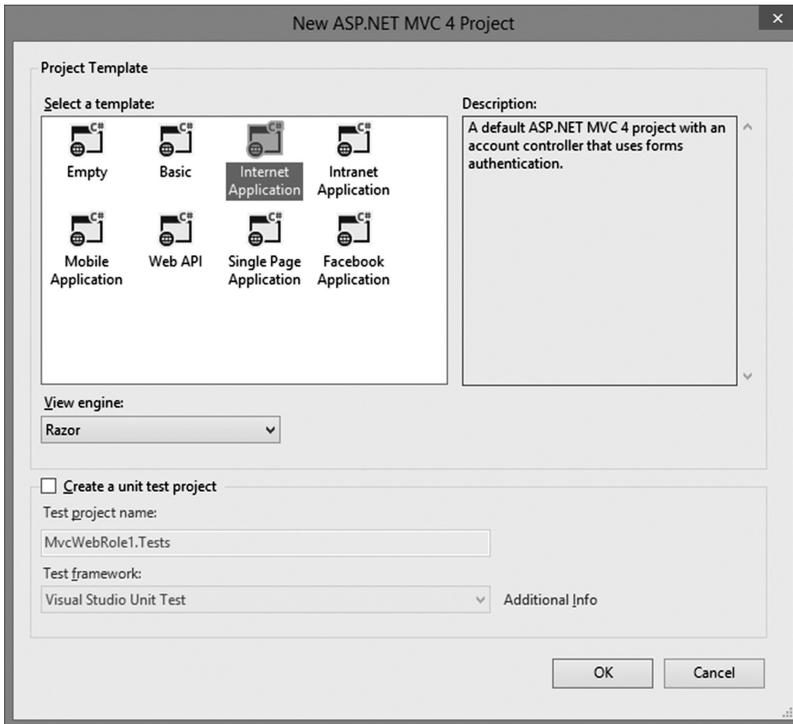


Figure 4.19 Internet Application template.

6. Modify **HomeController.cs** under the **Controllers** folder of the web role project. We'll add a new **SendMessage()** method for the client to call. The method code is shown in Code List 4.4.

One thing to note is that in line 3 in Code List 4.4, we use the name of the role, *WorkerRole*, to locate the worker role. If your worker role has a different name, you have to update your code accordingly. In addition, to make it shorter, the code cuts some corners. For instance, the code uses *Instance[0]* to choose the first instance of the worker role. This means that the

CODE LIST 4.3 SIMPLE MESSAGING WEB PAGE

```

@{
    ViewBag.Title = "Home Page";
}
<div id="log"></div>
<br />
<label>Enter message to send: </label>
<input type="text" id="message" style="width:300px;" />
<input type="button" id="send" value="Send" />
@section Scripts {
    <script>
        $('#send').click(function () {
            $.getJSON('/Home/SendMessage?message=' +
                encodeURIComponent(message.value),
                function (data) {
                    $('#log').append(data).
                        append('<br/>');
                });
        });
    </script>
}

```

CODE LIST 4.4 SENDMESSAGE() METHOD

```

1:public ActionResult SendMessage(string message)
2:{
3:    var endpoint = RoleEnvironment.Roles["WorkerRole1"].
        Instances[0].
4:    InstanceEndpoints.First().Value.IPEndpoint;
5:    var socket = new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram,
6:        ProtocolType.Udp);
7:    var buffer = new byte[1024];
8:    var data = Encoding.UTF8.GetBytes(message);
9:    socket.SendTo(data, endpoint);
10:    int size = socket.Receive(buffer);
11:    return Json(Encoding.UTF8.GetString(buffer, 0, size),
12:        JsonRequestBehavior.AllowGet);
13:}

```

code does not work well in a multi-instance deployment, as only the first worker role instance is used. In addition, the code assumes the UDP endpoint as the first endpoint on the worker role (*InstanceEndpoints.First()*). Look out for these pitfalls if your project structure is different.

7. Modify the worker role to change the original Input Endpoint to Internal Endpoint, as shown in Figure 4.20.
8. Press Ctrl + S to save the file.
9. Set the cloud service project as the startup project, and then press F5 to launch it.
10. Enter some test messages on the web page, and observe the responses from the backend server (see Figure 4.21).

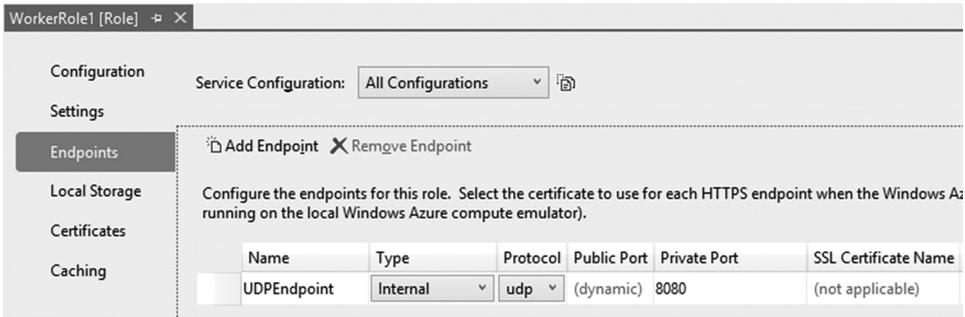


Figure 4.20 Modify endpoint type.

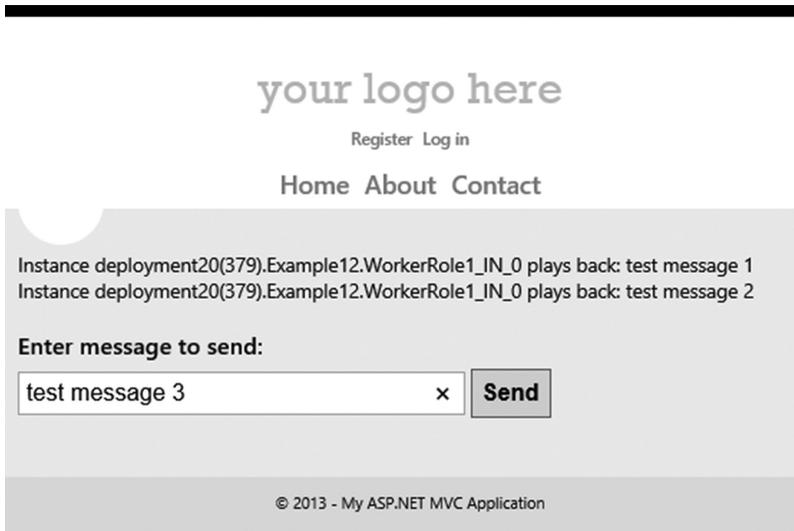


Figure 4.21 Test results on the website.

4.4 Role Lifecycle

In Example 4.2, we described how Microsoft Azure will invoke the *OnStart()* method and then the *Run()* method on a role instance when launching it. In this section, we will examine role life-cycle in more detail. First, let us go behind the scenes and see how a role is deployed on a virtual machine and launched.

4.4.1 Process of Deploying and Launching a Role Instance

The following is a summary of the process to deploy and launch a role instance:

1. Microsoft Azure picks a physical server with sufficient CPU cores from its running servers pool or starts up a new server that can satisfy the requirements of running the role instance.

2. Microsoft Azure copies the service package and its configuration file to the hosting server. A Host Agent running on the host server launches the virtual machine.
3. A *WaAppAgent* program on the virtual machine configures the virtual environment and then launches a *WaHostBootstrapper* process.
4. If the role has startup tasks, *WaHostBootstrapper* executes these tasks and waits for all simple tasks to finish successfully (see Section 4.5).
5. If the role is a web role, *WaHostBootstrapper* launches an *IISConfigurator* process to configure IIS.
6. If the role is a worker role, *WaHostBootstrapper* launches a *WaWorkerHost* process. If the role is a web role, *WaHostBootstrapper* launches a *WaIISHost* process.
7. These processes load the corresponding role assembly and search for a *RoleEntryPoint* subclass implementation.
8. The *OnStart()* method is called.
9. The *Run()* method is called. At the same time, the instance is marked as “Ready” and is joined to the load balancer.
10. If the *Run()* method exits, the *OnStop()* method is called, and the *WaWorkerHost*/*WaIISHost* process stops. The role instance is recycled.
11. Otherwise, *WaHostBootstrapper* starts to monitor status changes of the instance.

As we have learned earlier, it is not necessary for a web role to implement a *RoleEntryPoint* subclass. This is because the web role instance is deployed to IIS, and it is IIS that takes and forwards user requests. So, when we removed the *WebRole.cs* from the web role project earlier, the website still worked. However, if your web role does provide a *RoleEntryPoint* subclass, you need to ensure that the *Run()* method does not exit; otherwise, the role instance will be recycled.

4.4.2 Role Instance Statuses

The lifecycle of a role instance is depicted in Figure 4.22.

Figure 4.22 shows that a role instance can be in either Busy status or Ready status. Only when an instance is under the Ready status does it participate in job distributions from the load balancer.

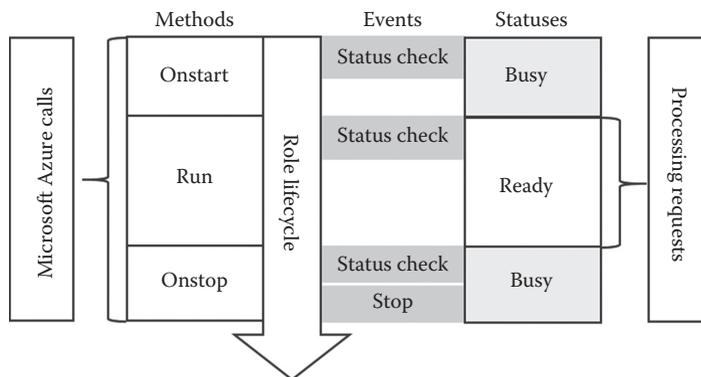


Figure 4.22 Role lifecycle.

If you want to programmatically change the role status, you can respond to the *StatusCheck* event on the *RoleEnvironment* class, and use the *Setbusy()* method on the *RoleInstanceStatusCheckEventArgs* parameter to mark the instance as Busy. Then, the load balancer will not send jobs to the instance until *WaHostBootstrapper* performs the next status check.

If the *Run()* method exits, or throws an unhandled exception, *WaHostBootstrapper* will attempt to restart your role instance. Of course, during the restart process, the instance cannot take in any user requests because it is not in the Ready status. If *WaHostBootstrapper* itself crashes, or the virtual machine crashes, Microsoft Azure will restart the virtual machine and relaunch the role instance. Finally, if the hosting server fails, Microsoft Azure will deploy the instance to a healthy server after several attempts to restore the failing server. This comprehensive autorecovery mechanism increases the availability of your role instance.

When Microsoft Azure shuts down a role instance, it will trigger a *Stopping* event, and will invoke the *OnStop()* method on the instance. This gives the code a chance to perform the necessary operations, such as releasing allocated resources, before the instance is stopped.

Finally, if you want to force your instance to restart, you can call the *RoleEnvironment.RequestRecycle()* method to inform Microsoft Azure to restart your instance.

4.5 Startup Tasks

In Section 4.1.1, we mentioned startup tasks. The so-called startup tasks are scripts or programs that run before the role instance starts. A startup task can be an executable (.exe), or a batch file (.cmd). You can customize the virtual machine running the role instance via startup tasks:

- Install prerequisites
- Register COM components
- Modify the Registry
- Run PowerShell scripts
- Launch backend processes

4.5.1 Defining Startup Tasks

You can define startup tasks in cloud service definition files. For example, the following code list defines a startup task for a web role. The task is to run a batch file named *Startup.cmd*. You can define multiple startup tasks under the *<Startup>* element, and they will be executed in the order they are defined.

```
<WebRole name="MvcWebRole1" vmsize="Small">
  ...
  <Startup>
    <Task commandLine="Startup.cmd"
          executionContext="elevated"
          taskType="simple" />
  </Startup>
</WebRole>
```

4.5.2 Startup Task Properties

A startup task has the following properties:

- `commandLine`. This property defines the executable or the script to be executed. Note that the script file has to be saved in ANSI format.
- `executionContext`. A startup task can run with (default) limited or elevated privileges. With this property set to “limited,” the startup task runs under the same privileges as what the role instance has. With the property set to “elevated,” the startup task runs with administrative privileges. If you need to install software, register COM components, or modify the Registry in the startup task, you need to set its execution context to “elevated.”
- `taskType`. A startup task can be one of three types: simple, background, or foreground. Microsoft Azure waits for all simple tasks to successfully return (with return value 0) before it starts the role instance. If any of the simple startup tasks fails (do not return 0) or hang, role instance launch will be canceled. On the other hand, Microsoft Azure does not wait for background tasks but continues to launch the role instance. The foreground tasks are similar to the background tasks. The only difference is that a role instance cannot be recycled before foreground tasks are completed.

Note: While writing a batch script, ensure that the script returns 0 upon success. Otherwise, the role instance may fail to start. Commonly, you would add to the end of your script:

```
EXIT /B 0
```

Next, we will learn startup task usage through an example.

Example 4.4: Startup Task—convert an image to ASCII web page

Difficulty: ***

In this example, we create a simple website, which converts user-submitted images to ASCII characters and displays them on the page. The conversion is done by a separate Windows Console application, which takes an image via HTTP and converts the image to corresponding ASCII characters. The returned ASCII presentation is wrapped in an HTML `<div>` tag, so it can be directly displayed on a web page. This example simulates the scenario when you want to package a legacy program and provide a new web frontend for it. The architecture of the completed system is shown as Figure 4.23.

1. Launch Visual Studio as an administrator. Create a new Microsoft Azure Cloud Service with a single ASP.NET MVC 4 web role (using the Internet Application template).
2. Add a new Windows Console Application to the solution. Add references to `System.Drawing` assembly and `System.Web` assembly. Then, open its **Program.cs** file, and add the namespace references:

```
using System.Net;
using System.Drawing;
```

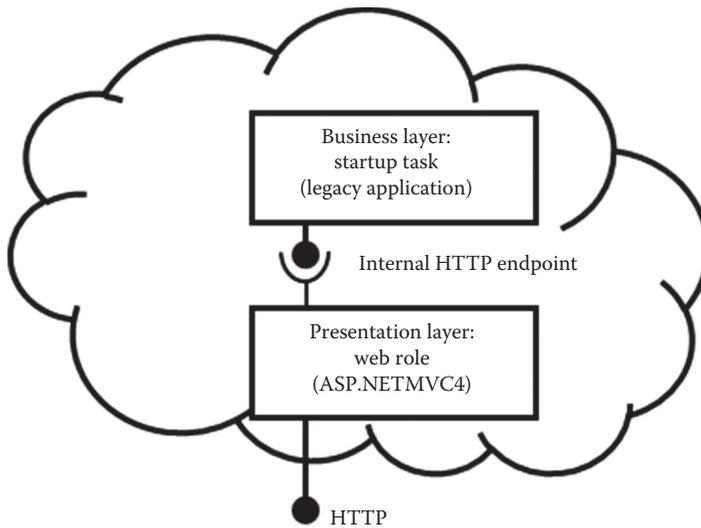


Figure 4.23 Architecture of Example 4.4.

3. Implement the *Main()* method, which starts up an *HttpListener* to take in user requests:

```
static void Main(string[] args)
{
    HttpListener listener = new HttpListener();
    listener.Prefixes.Add("http://*:8080/");
    listener.Start();
    while (listener.IsListening)
    {
        var context = listener.GetContext();
        ProcessRequest(context);
    }
    listener.Stop();
}
```

4. Implement the *ProcessRequest()* method, which takes an image, converts it to a grayscale image with about 70 levels of intensities, and then maps the pixels to corresponding ASCII characters. The method returns an HTML <div> tag, which contains rows of ASCII characters representing the original image. The complete source code is shown in Code List 4.5.
5. Modify the **Index.cshhtml** file under the **Views\Home** folder of the web role project. Replace the entire file content with the code in Code List 4.6.
6. Modify the **HomeController.cs** file under the **Controllers** folder of the web role project. Add a new method as shown in Code List 4.7. The method allows the user to upload an image, and then submits the image to the background program for processing. Finally, it returns the HTML result to the browser.
7. Rebuild the solution to ensure everything builds at this point.
8. Right-click the web role project, and select the **Add→Existing Item** menu. Then, add the Console application output file `bin\debug\ConsoleApplication1.exe` (assuming you did not change the project name) to the root folder of the web role.

CODE LIST 4.5 IMAGE TO ASCII CONVERSION

```
//Characters in this string represent different intensities
const string grayPixel =
"$@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/\ \ | () 1 {} [] ? -
_+~<>i!lI;:;,\"^'`. ";
static void ProcessRequest(HttpListenerContext context)
{
    Bitmap picture = new Bitmap(context.Request.InputStream);
    int pixSize = 1;//How many pixels an ASCII character covers
    string retVal = "<div style=\"font-family:SimHei;font-
        size:11px;\">";
    //Analyze the pixels one by one and covert them to corresponding
        ASCII characters
    string retLine = "";
    for (int y = 0; y < picture.Height; y += pixSize)
    {
        retLine = "";
        for (int x = 0; x < picture.Width; x += pixSize)
        {
            int grayscale = 0;
            int pixCount = 0;
            for (int j = 0; j < pixSize; j++)
            {
                for (int i = 0; i < pixSize; i++)
                {
                    int indx = x + i;
                    int indy = y + j;
                    if (indy < picture.Height && indx < picture.
                        Width)
                    {
                        var pixel = picture.GetPixel(indx, indy);
                        grayscale += (int)(pixel.R * .3
                            + pixel.G * .59 + pixel.B * .11);
                        pixCount++;
                    }
                }
            }
            int grayLevel = (int)(grayscale / pixCount / 3.66);
            retLine += grayPixel[grayLevel];
        }
        retVal += System.Web.HttpUtility.HtmlEncode(retLine) +
            "<br/>";
    }
    retVal += "</div>";
    //Generate HTML output
    byte[] b = Encoding.ASCII.GetBytes(retVal);
    context.Response.StatusCode = 200;
}
```

```

context.Response.KeepAlive = false;
context.Response.ContentLength64 = b.Length;
var output = context.Response.OutputStream;
output.Write(b, 0, b.Length);
context.Response.Close();
}

```

CODE LIST 4.6 IMAGE UPLOAD UI

```

@{
    ViewBag.Title = "Home Page";
}
@using (Html.BeginForm("Index", "Home", FormMethod.Post,
    new { enctype = "multipart/form-data" }))
{
    <input type="file" name="file" />
    <input type="submit" value="Upload" />
}

```

CODE LIST 4.7 IMAGE UPLOAD METHOD

```

[HttpPost]
public ActionResult Index(HttpPostedFileBase file)
{
    var result = "";
    if (file != null && file.ContentLength > 0)
    {
        Bitmap picture = new Bitmap(file.InputStream);
        WebClient client = new WebClient();
        ImageConverter converter = new ImageConverter();
        var retData = client.UploadData("http://localhost:8080/",
            (byte[])converter.ConvertTo(picture, typeof(byte[])));
        result = Encoding.ASCII.GetString(retData);
    }
    return Content(result);
}

```

9. Right-click the added file, and select the **Properties** menu. Then, in its **Properties** window, choose **Copy to Output Directory** to **Copy always**.
10. Right-click the Console application project, and select the **Properties** menu. Configure a post-build event to copy the Console application to the web role to avoid repeating step 8, as shown in Figure 4.24.
11. Right-click the solution, and select the **Project Dependencies** menu. Then, on the **Project Dependencies** window, ensure the web role project depends on the Console application

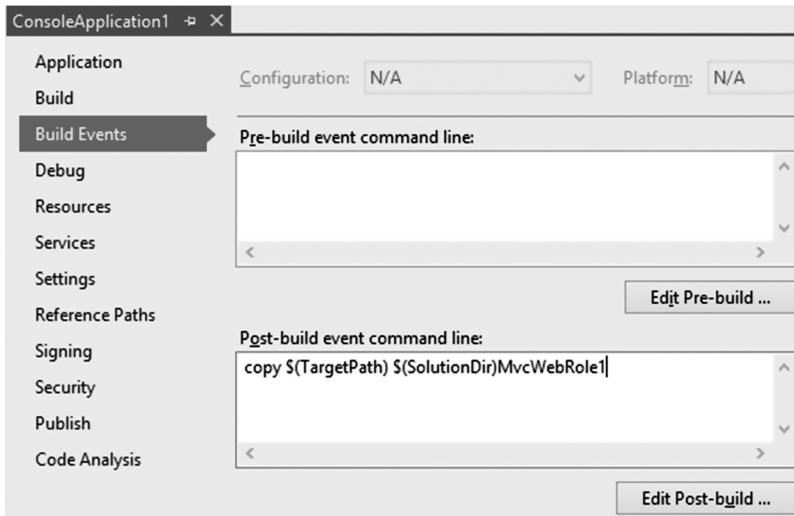


Figure 4.24 Copy the Console application to the web role upon successful builds.

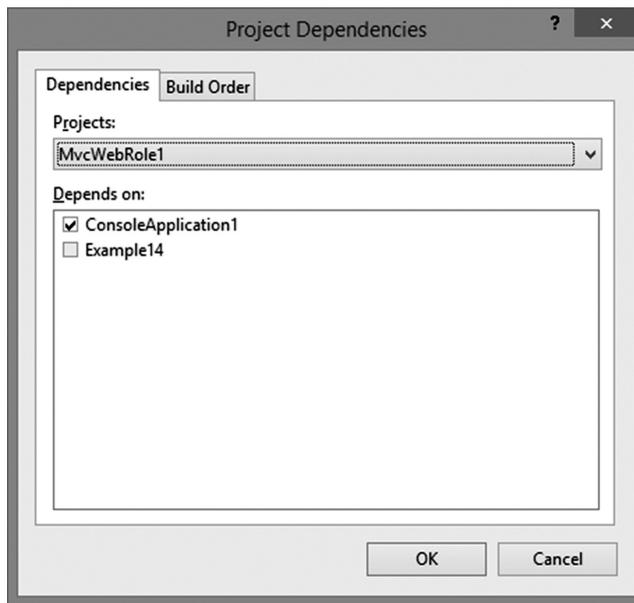


Figure 4.25 Setting up project dependencies.

(see Figure 4.25) so that you can always get the latest Console application packaged in your web role.

12. Modify the cloud service definition file (**ServiceDefinition.csdef**) in the cloud service project to add a startup task definition under the `<WebRole>` element. This task launches **ConsoleApplication1.exe** with administrative privileges in the background before the web role starts:

```
<Startup>
<Task commandLine="ConsoleApplication1.exe"
executionContext="elevated"
        taskType="background"/>
</Startup>
```

13. Press F5 to launch the application. On the web page, click on the Browse button and pick a local image (we recommend using an image with a smaller size, not exceeding 500 × 500 pixels). Then, click the Upload button to upload the image, as shown in Figure 4.26.
14. Figure 4.27 shows the result page.

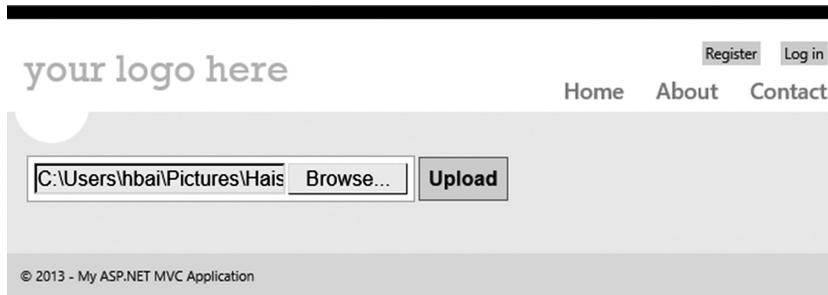


Figure 4.26 Image upload UI.



Figure 4.27 Sample result of conversion.

4.6 Diagnostics and Debug

Microsoft Azure provides comprehensive support for cloud service monitoring, diagnostics, and debugging. Cloud service developers receive strong support in both development and operation.

4.6.1 Debugging Locally

As you have already learned, Microsoft Azure SDK provides local emulators for computing as well as storage services. No matter whether you are developing a simple website or a complex multitiered solution, you can debug your cloud services just like debugging a local program, such as using breakpoints and stepped execution, inspecting thread states and call stacks, etc. In addition, you can observe tracing information recorded by *System.Diagnostics.Trace* statements in the Compute Emulator UI (see Example 3.1). For example, the default implementation of the `Run()` method in a worker role is instrumented with tracing statements:

```
public override void Run()
{
    Trace.TraceInformation("WorkerRole1 entry point called",
        "Information");

    while (true)
    {
        Thread.Sleep(10000);
        Trace.TraceInformation("Working", "Information");
    }
}
```

When the worker role runs, you can observe the tracing information on the Compute Emulator UI, as shown in Figure 4.28. If you are already familiar with Visual Studio and coding with ASP.NET, ASP.NET MVC, C#, or VB, you can apply the skills and tools you already know to cloud service developments. We recommend that you fully use the advantages of such knowledge and ensure your cloud service works as designed locally before deploying to Microsoft Azure.

4.6.2 Microsoft Azure Diagnostics

Debugging on cloud platforms is a bit more complex. Traditionally, the main method of server-side diagnostics is to collect various log files. In other words, services record tracing information to local file systems or databases so it can be available for diagnostics when problems happen. However, you cannot simply rely on the log files written on virtual machines hosted by Microsoft Azure. For example, when a virtual machine crashes, your role instance running on the machine will be migrated to a new, healthy virtual machine and it will lose all the local files. So, on Microsoft Azure, you are never supposed to store permanent data locally on virtual machines (though you can save temporary data) as these machines are supported to be stateless. Obviously, one easy solution to this problem is to write the logs directly to an external storage so we can store the data independently. However, the frequent service calls to external storage services may jeopardize system performance. To compromise for this, the log files can be saved to local storages temporarily, and a separate process will transfer the logs periodically to external storages on

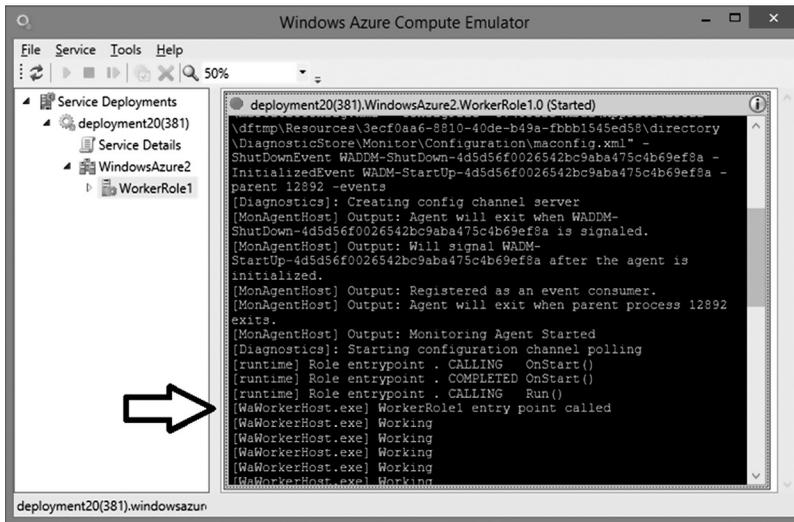


Figure 4.28 Tracing information in Compute Emulator UI.

a schedule or on demand. This method provides not only a balance between performance and log persistence but also the capability to record events such as role instance crashes because it runs independently from role instances.

Microsoft Azure Diagnostics is a Microsoft Azure module, which you can import to your cloud services to gain diagnostic capabilities. You can configure Microsoft Azure Diagnostics in code as well as in configuration files. You can also modify the settings remotely during operation. Once enabled, Microsoft Azure Diagnostics agents run on the role instance virtual machines to collect tracing information, and transmit tracing data to user-specified storage accounts. Next, let us learn the basics of configuring and using Microsoft Azure Diagnostics.

Example 4.5: Configure Microsoft Azure Diagnostics

Difficulty: **

In this example, we learn how to configure Microsoft Azure Diagnostics for a simple cloud service, and how to transfer log data to Microsoft Azure Storage services. We use Microsoft Azure Table Storage in this example. As we have not introduced the service yet, you can simply consider these tables to be database tables (this understanding is actually wrong—but sufficient for now). We introduce Microsoft Azure Storage services in Chapter 6.

1. Launch Visual Studio as an administrator. Create a new cloud service with a worker role with the name **WorkerRole1**.
2. By default, your cloud service roles have already imported the Microsoft Azure Diagnostics module. You can observe how this is done in the **ServiceDefinition.csdef** file:

```

<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition ...>
  <WorkerRole name="WorkerRole1" vmsize="Small">
  
```

```

<Imports>
  <Import moduleName="Diagnostics" />
</Imports>
</WorkerRole>
</ServiceDefinition>

```

3. You can configure Microsoft Azure Diagnostics on your role's Properties page. As shown in Figure 4.29, we have changed Microsoft Azure Diagnostics to transfer all logged information to emulated storage account. By default, the transfer interval is 1 min. You can further customize the settings by switching the option to **Custom plan**, and then use the **Edit** button to customize the diagnostics plan.

In a real deployment, you should configure the target Microsoft Azure Storage account to an actual storage account. But for local tests, you can use the emulated storage services provided by Microsoft Azure SDK. This is the default setting when you create a new cloud service. The location of the storage service is defined by a connection string (similar to a database connection string), which is read from the `Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString` setting of the role. For an emulated environment, the connection string is fixed to "UseDevelopmentStorage=true" (see Figure 4.29).

4. Press F5 to launch the service. After the service has been running for several minutes, you can examine the logs that are transferred to the emulated storage service. In Visual Studio, select **VIEW**→**Server Explorer** to open the Server Explorer, which provides integrated management UI for many Microsoft Azure services within Visual Studio. Expand the tree to the **Microsoft Azure**→**Storage**→**(Development)**→**Tables** node, which contains a `WADLogsTable` (see Figure 4.30). This is where outputs from `System.Diagnostics.Trace` calls are recorded.
5. Double click **WADLogsTable** and you will see the data in the table (see Figure 4.31; we have dragged the Message column to the left to fit it on the screen).
6. Once the service has been launched, you will see a `diagnostics.wadcfg` file appearing under the role node in **Solution Explorer**. This file contains Microsoft Azure Diagnostics settings. The changes you make in step 3 are reflected in this file (Figure 4.32).

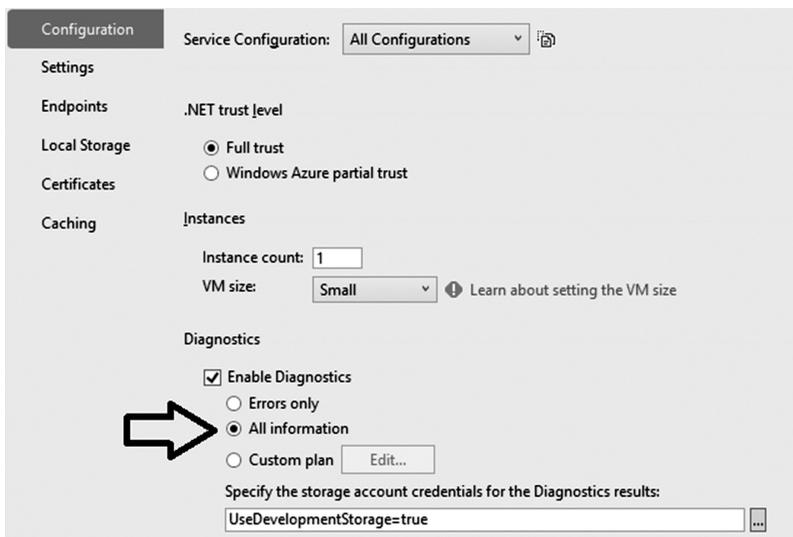


Figure 4.29 Microsoft Azure Diagnostics configuration.

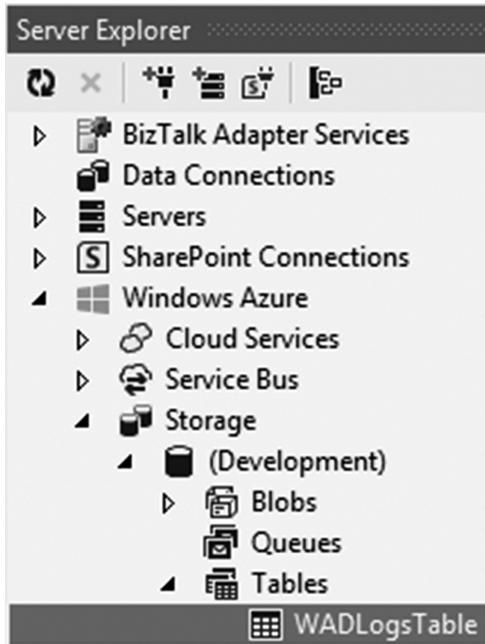


Figure 4.30 WADLogsTable in Server Explorer.

PartitionKey	RowKey	Timestamp	Message
06351179322000...	deployment20(...	8/11/2013 4:48:...	WorkerRole1 entry point called; TraceSource 'WaWorkerHost.e
06351179322000...	deployment20(...	8/11/2013 4:48:...	Working; TraceSource 'WaWorkerHost.exe' event
06351179322000...	deployment20(...	8/11/2013 4:48:...	Working; TraceSource 'WaWorkerHost.exe' event
06351179322000...	deployment20(...	8/11/2013 4:48:...	Working; TraceSource 'WaWorkerHost.exe' event
06351179322000...	deployment20(...	8/11/2013 4:48:...	Working; TraceSource 'WaWorkerHost.exe' event
06351179328000...	deployment20(...	8/11/2013 4:50:...	Working; TraceSource 'WaWorkerHost.exe' event
06351179328000...	deployment20(...	8/11/2013 4:50:...	Working; TraceSource 'WaWorkerHost.exe' event

Figure 4.31 Sample log entries.

- The *diagnostics.wadcfg* file is an XML file. You can double click it to view its contents. Although you can edit this file manually, it is highly recommended to change the settings using the custom plan editor on the role's Properties page (see Figure 4.29). For example, Figure 4.33 shows the specifications to trace a couple of performance counters and to transfer their values by 1 min intervals.

Note: On the **Properties** page, there is a checkbox with a very long text: Update development storage connection strings for Diagnostics and Caching with Microsoft Azure

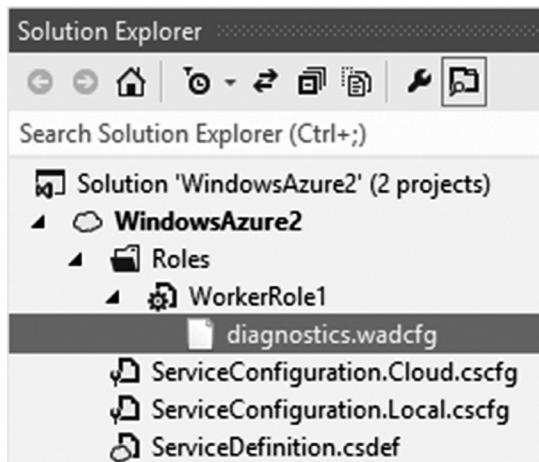


Figure 4.32 diagnostics.wadcfg file.

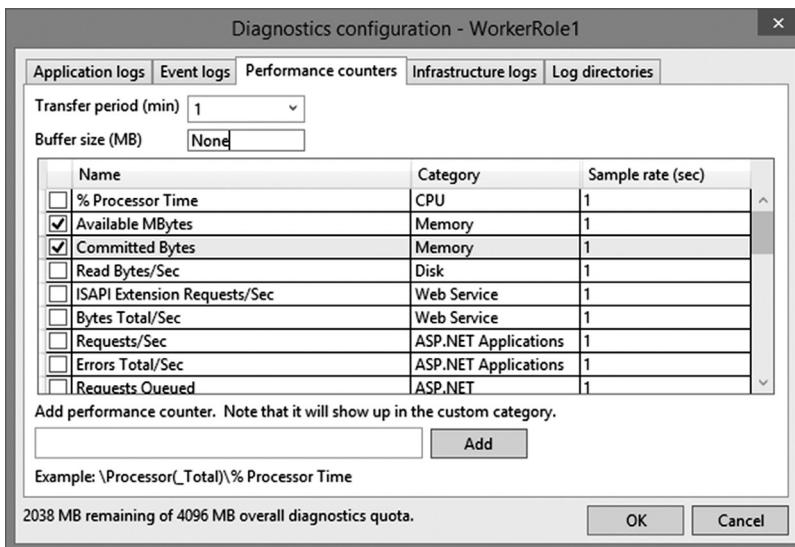


Figure 4.33 Custom diagnostics plan editor.

storage account credentials when publishing to Microsoft Azure. This means that when you deploy a cloud service, Microsoft Azure SDK will update connection string settings (such as `Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString`) to use actual Microsoft Azure Storage accounts.

Microsoft Azure Diagnostics information is saved in different locations based on the types of logged data, as shown in Table 4.1 (we have seen `WADLogsTable` in the previous example).

Table 4.1 Microsoft Azure Diagnostics Information Storage Locations

<i>Diagnostics Data</i>	<i>Supported Role Types</i>	<i>Service</i>	<i>Location</i>
Microsoft Azure Logs	Web, Worker	Table	WADLogsTable
IIS Logs	Web	BLOB	Wad-iis-logfiles
Windows Diagnostics Logs	Web, Worker	Table	WADLogsTable
Failed Requests Logs	Web	BLOB	Wad-iis-failedreqlogfiles
Windows Event Logs	Web, Worker	Table	WADWindowsEventLogTable
Performance Counters	Web, Worker	Table	WADPerformanceCountersTable
Crash Dumps	Web, Worker	BLOB	Wad-crash-dumps
Custom Logs	Web, Worker	BLOB	User-defined location

4.6.3 IntelliTrace

IntelliTrace allows you to collect detailed diagnostics data on your role instances. While debugging, you can download IntelliTrace data from Microsoft Azure and play it back in Visual Studio. In other words, you will be able to reconstruct the running context when an error occurs on the server and step through the code. The running environment of a cloud service role instance is dynamic. Triggering conditions often disappear after errors have occurred and are hard to be reestablished. By using IntelliTrace, you can capture the server states and play them back in Visual Studio so that you can easily find out what exactly happened when the error occurred. So, IntelliTrace is a very powerful feature that you should leverage for diagnostics.

Note: Using IntelliTrace requires Visual Studio Ultimate edition.

Now let us learn how to use IntelliTrace with an example.

Example 4.6: IntelliTrace—Greatest Common Divisor

Difficulty: ****

In this example, we will create a simple website. The website takes two positive integers from the user and calculates their greatest common divisor. Of course, the exact feature is not important—we just need a simple scenario to learn about the debugging process.

1. Launch Visual Studio as an administrator. Create a new cloud service project with an ASP.NET/Web Forms web role (note that in this case we are not using ASP.NET MVC).
2. Replace the code in **Default.aspx** with the code in Code List 4.8.
3. Edit **Default.aspx.cs** to modify the server-side code to add the methods in Code List 4.9.

CODE LIST 4.8 USER INTERFACE

```

<%@ Page Title="Home Page" Language="C#"
MasterPageFile=~\Site.Master" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="WebRole1._Default" %>
<asp:Content runat="server" ID="BodyContent"
ContentPlaceHolderID="MainContent">
    <table>
        <tr><td>Value 1</td>
        <td><asp:TextBox ID="valueA" runat="server">
            </asp:TextBox></td></tr>
        <tr><td>Value 2</td>
        <td><asp:TextBox ID="valueB" runat="server">
            </asp:TextBox></td></tr>
        <tr><td>Result</td>
        <td><asp:TextBox ID="result" runat="server">
            </asp:TextBox></td></tr>
        <tr><td colspan="2"><asp:Button ID="calculate"
            runat="server"
            Text="Calculate" OnClick="calculate_Click" />
            </td></tr>
    </table>
</asp:Content>

```

CODE LIST 4.9 BACK END LOGIC

```

protected void calculate_Click(object sender, EventArgs e)
{
    result.Text = findGreatestCommonDivisor(valueA.Text, valueB.
        Text);
}
private string findGreatestCommonDivisor(string value1, string value2)
{
    return euclid(int.Parse(value1), int.Parse(value2)).ToString();
}
private int euclid(int a, int b)
{
    if (b == 0)
        return a;
    else
        return euclid(b, a % b);
}

```

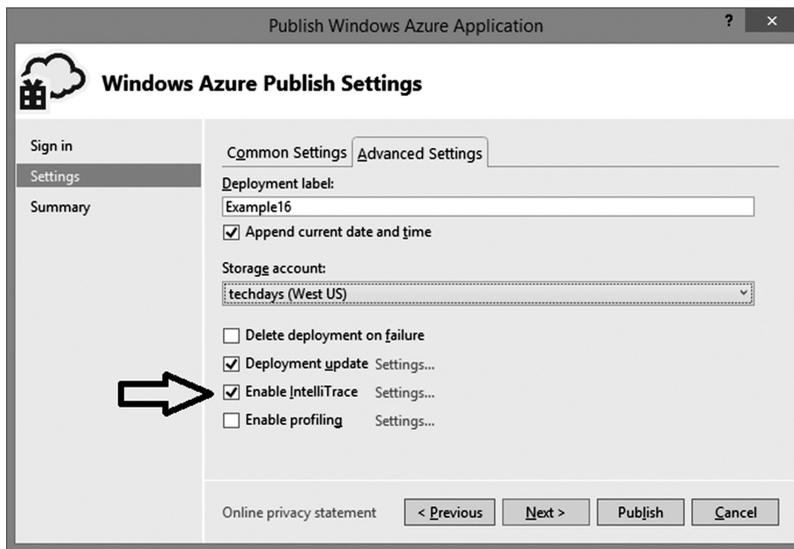


Figure 4.34 Enabling IntelliTrace when publishing a cloud service.

4. To use IntelliTrace, you need to enable it in your cloud service configuration and deploy the service to Microsoft Azure, as shown in Figure 4.34.
5. Once the service has been deployed, it is ready for some simple tests, as shown in Figure 4.35.
6. Now, enter an invalid number, for instance “teapot,” in the **Value 1** field. Click on the **Calculate** button again. This will cause the website to crash, as shown in Figure 4.36.
7. In many circumstances, when a customer reports a problem, he often cannot provide detailed information, especially when the service has crashed. When the service is restored, the trigger condition has gone, making recreating and debugging such problems very difficult. Fortunately, we have enabled IntelliTrace, and now we can use IntelliTrace to help us to fix the problem. In Visual Studio, select the **VIEW**→**Server Explorer** menu to open the Server Explorer. Expand to the role instance, right-click, and select the **View IntelliTrace Logs** menu, as shown in Figure 4.37.
8. You can observe IntelliTrace Logs being downloaded in the **Microsoft Azure Active Log** window (see Figure 4.38).
9. Once IntelliTrace Logs are downloaded, the **IntelliTrace Summary** view will be displayed. In the **Exception data** section, you will see the “Input string was not in a correct format” exception, as shown in Figure 4.39.
10. Double click on the exception data, and you will be directly taken to the line where the exception is thrown (see Figure 4.40). This is already very cool. For a simple case like this, this feature is already sufficient for us to discover and resolve the problem. However, for more complex cases, we might need a little more assistance.
11. In Visual Studio, select the **DEBUG**→**Windows**→**Call Stack** menu to open the call stack view. You can easily see that when the *findGreatestCommonDivisor* method is called, the first parameter is set to “teapot.” In other words, not only can you see the detailed call stack, you can also examine the exact parameter values when a function is called (see Figure 4.41).
12. IntelliTrace provides very rich debugging features, which cannot be covered in full here. Instead, we will present just one more IntelliTrace feature. In Visual Studio, select the **DEBUG**→**Windows**→**IntelliTrace Event** menu. In this view, you can see the series of events before the error occurred, from the page being loaded to the user clicking the **Calculate** button, as shown in Figure 4.42.



Figure 4.35 Positive test result.

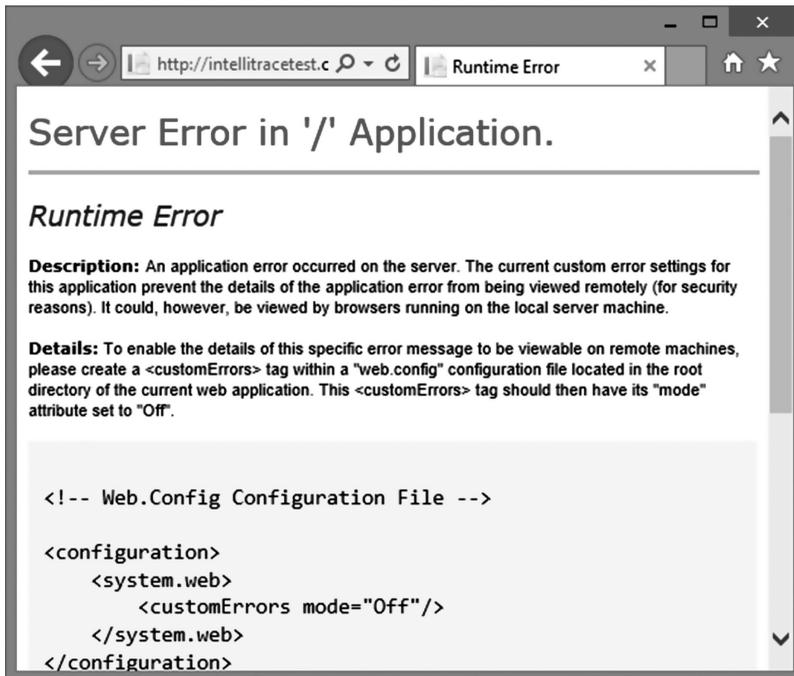


Figure 4.36 Crashed website.

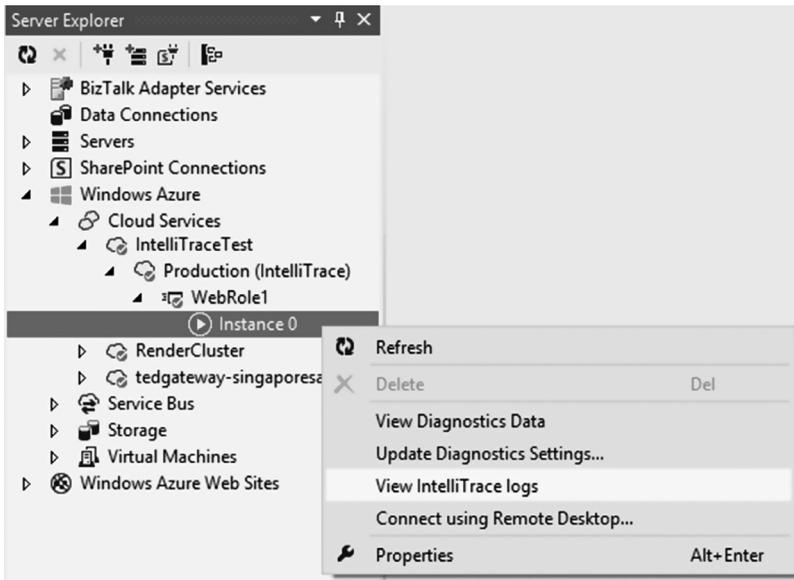


Figure 4.37 View IntelliTrace Logs menu in Server Explorer.

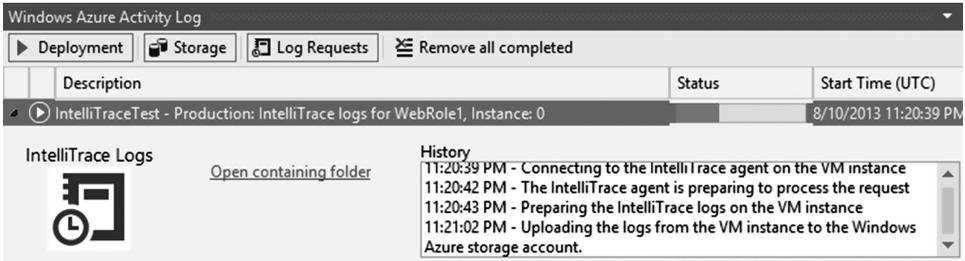


Figure 4.38 Downloading IntelliTrace Logs.

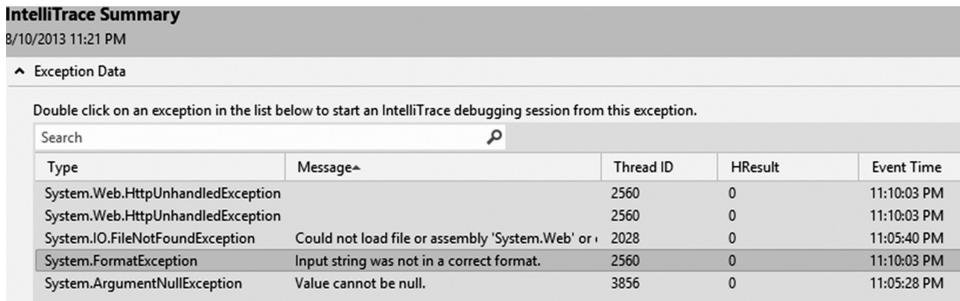


Figure 4.39 IntelliTrace summary.

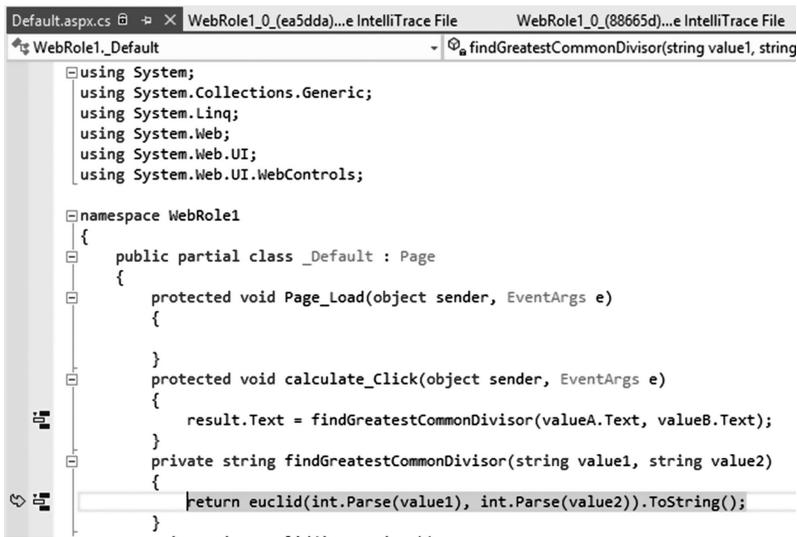


Figure 4.40 IntelliTrace takes the developer to the line where exception is thrown.

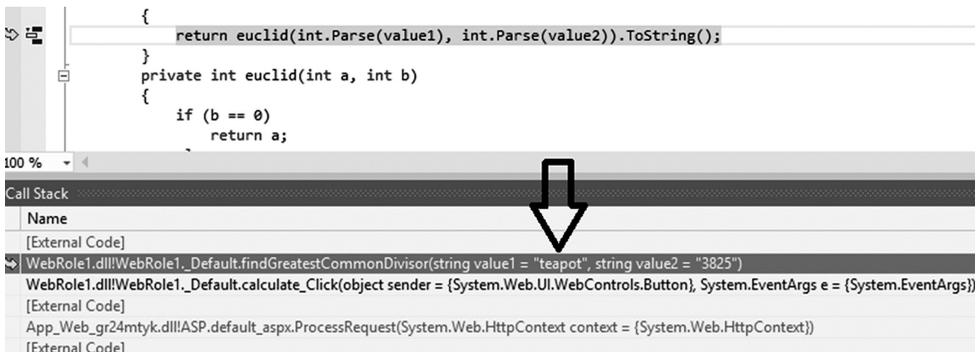


Figure 4.41 Parameter details in IntelliTrace Logs.

4.6.4 Monitoring Cloud Service

Microsoft Azure also provides means of monitoring the health of your cloud services with Microsoft Azure Management Portal and third-party tools.

■ Customizing cloud service monitor

Similar to the DASHBOARD view and the MONITOR view of Azure Websites (see Section 2.8.2), MACS also allows you to monitor your cloud services on Microsoft Azure Management Portal (see Figure 4.43).

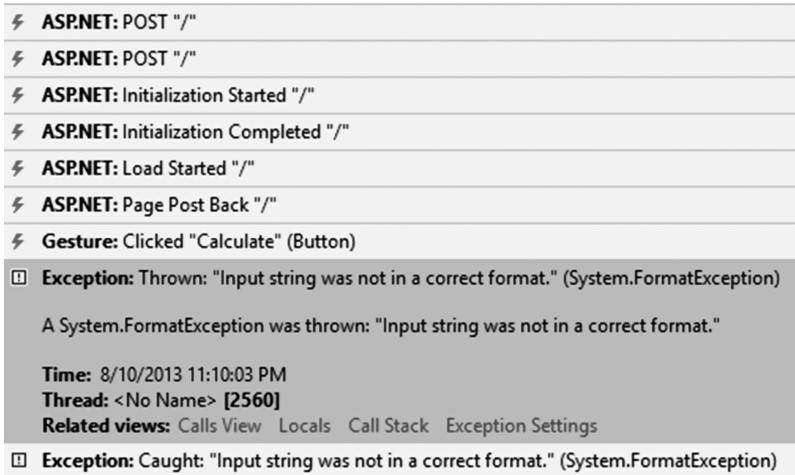


Figure 4.42 IntelliTrace Event view.

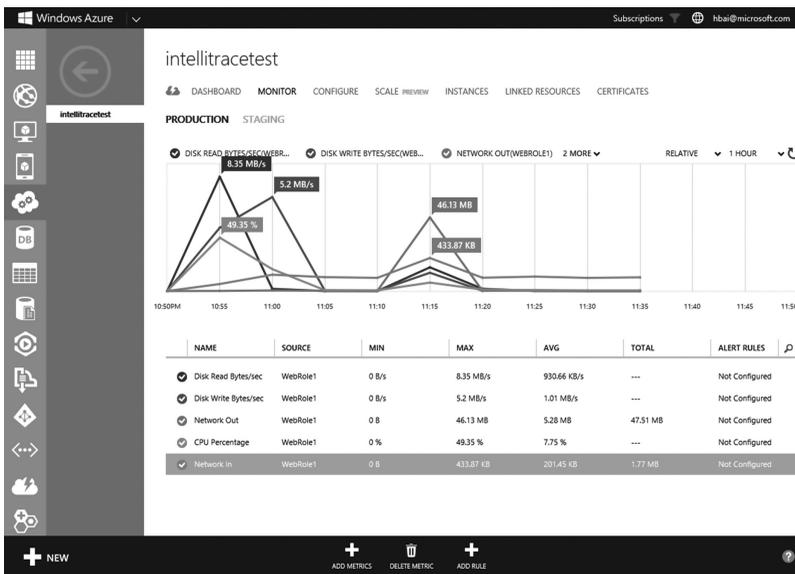


Figure 4.43 Monitoring cloud service on Microsoft Azure Management Portal.

You can use the **ADD METRICS** icon to define data series to be presented on the monitor view. By default you can choose from the following:

- CPU Percentage
- Disk Read Bytes/Sec
- Disk Write Bytes/Sec
- Network In
- Network Out

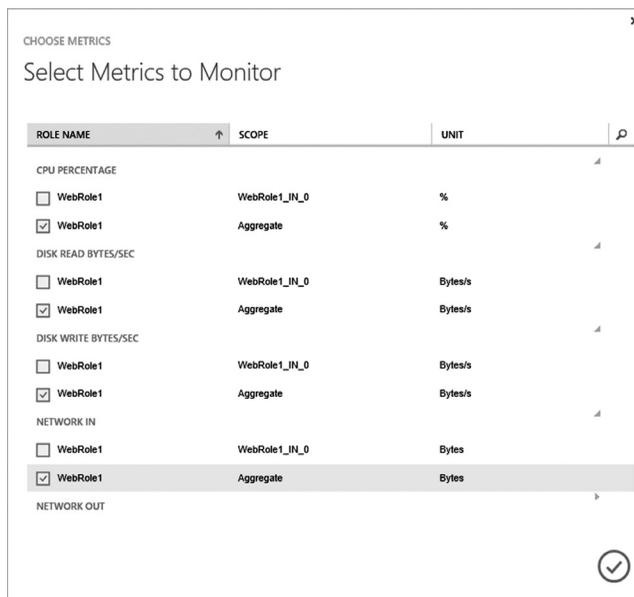


Figure 4.44 Select metrics to monitor.

In addition, for each performance counter, you can select to observe the data series of each instance, or average values across all instances, as shown in Figure 4.44.

On the cloud service's CONFIGURE page, you can configure monitoring levels (minimal or verbose). With minimal level, Microsoft Azure only provides the metrics introduced earlier. With verbose level, you can not only collect more metrics, but also define custom performance counters. However, because recording additional data requires more storage space, you will be warned when you make the change, as shown in Figure 4.45.

■ Customizing monitoring view

You can easily customize the monitoring view. At the upper-right corner of the view, you can change the time period of the monitoring window. You can also choose whether the data axis should be RELATIVE or ABSOLUTE. In relative mode, each data series is normalized to the percentage of its value range. In absolute mode, each data series is presented in its original values. Relative mode works better when you want to observe each data line separately, and absolute mode is more suitable to compare different data series. In addition, you can click on the check icon besides a data line (see the arrow in Figure 4.46) to add it to the view or to remove it from the view.

■ Alerts

On the monitor view of a cloud service, you can define alert rules based on different metrics. When a monitored value exceeds the threshold you specify, an alert email will be sent to the email addresses of your choice. For example, to get email alerts when the server is too busy, you can follow these steps:

- On the monitor view of a cloud service, ensure **CPU Percentage** is selected. Then, click the **ADD RULE** icon on the command bar.
- On **CREATE ALERT RULE** dialog, enter a name and a description for your new rule, and click the next icon to continue (Figure 4.47).



Figure 4.45 Change cloud service monitoring settings.

	NAME	SOURCE	MIN	MAX	AVG	TOTAL	ALERT RULES	
<input checked="" type="checkbox"/>	Disk Read Bytes/sec	WebRole1	0 B/s	1.73 MB/s	194.53 KB/s	---	Not Configured	
<input checked="" type="checkbox"/>	Disk Write Bytes/sec	WebRole1	6.15 KB/s	1.01 MB/s	114 KB/s	---	Not Configured	
<input checked="" type="checkbox"/>	Network Out	WebRole1	156.33 KB	46.13 MB	4.75 MB	47.53 MB	Not Configured	
<input checked="" type="checkbox"/>	CPU Percentage	WebRole1	0.57 %	7.8 %	1.41 %	---	Not Configured	
<input checked="" type="checkbox"/>	Network In	WebRole1	172.85 KB	433.87 KB	203.76 KB	1.99 MB	Not Configured	



Figure 4.46 Customize monitoring view.

- On the next screen, set the notification condition to be greater than 80%, and alert action to send an email to the service administrator. Click the check icon to save the rule (Figure 4.48).
- Other monitoring methods

Microsoft Azure Management Portal is not the only way to monitor cloud services. You can also choose to use other first-party or third-party tools to monitor your Microsoft Azure resources. For example, Microsoft System Center provides a Management Pack for Microsoft Azure, which allows you to manage and monitor your Microsoft Azure resources within the System Center (you can download the pack from <http://www.microsoft.com/en-us/download/details.aspx?id=38414>).

With Microsoft Azure gaining popularity, an increasing number of third-party tools have emerged in the market to provide Microsoft Azure monitoring capabilities, such as New Relic and AppDynamics. Figure 4.49 is a sample view from New Relic that displays regional performance metrics collected using distributed workloads.

CREATE ALERT RULE

Define Alert

NAME [?]

Server too busy

DESCRIPTION

Alert me when server is too busy

SERVICE TYPE SERVICE NAME

Cloud Service intellitracetest

CLOUD SERVICE DEPLOYMENT

3858afa7fbdb44b49584ae475d78a5f9 (currently in Production environment)

CLOUD SERVICE ROLE

WebRole1

2

Figure 4.47 Defining a new alert rule.

CREATE ALERT RULE

Define a condition for notifications

METRIC [?]

CPU Percentage

CONDITION THRESHOLD VALUE UNIT

greater than 80 %

ALERT EVALUATION WINDOW [?]

Average over the last 10 minutes

ACTIONS [?]

Send an email to the service administrator and co-administrators

Enter the email address for another administrator

Enable Rule

1

Windows Azure privacy statement

Figure 4.48 Defining notification condition.

4.7 Developer Community

Developing and managing cloud services is a relatively new field. It is unavoidable to encounter various problems in the process of learning and using it. There is no single book or documentation that can answer all the questions. Fortunately, there have always been developer communities who are interested in Microsoft's development techniques and tools. An active developer's community is very helpful to improve your development speed and efficiency. To learn about Microsoft Azure,

United States Apdex →

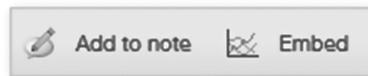
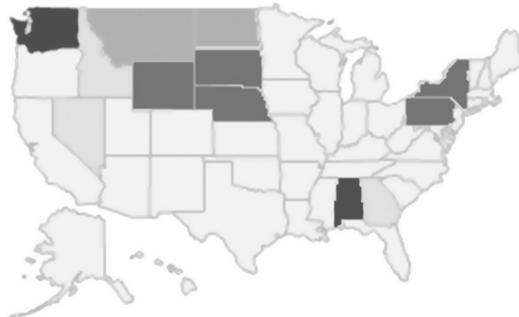


Figure 4.49 Sample view from New Relic.

you can find detailed documentation and rich demos at www.windowsazure.com and MSDN, as well as hundreds of teaching videos on Channel 9. In addition, you can exchange knowledge and experience at various developer communities such as the MSDN forum and Stack Overflow. We highly recommend that you take part in the community and bring your experience, proposals, and ideas to the community. Last but not the least, you can of course Bing the answers to your questions.

Microsoft also provides various supportive services to the developers and software vendors of different levels, such as the DreamSpark program dedicated to the students, the BizSpark program for startups, the MVP program for individuals, MSDN subscriptions, and different certifications. Many MSDN subscription levels include a certain number of free-of-charge Microsoft Azure services. If you are a subscriber of MSDN, you may contact your administrator for more information regarding this benefit. You can also consult <http://www.windowsazure.com/en-us/pricing/member-offers/msdn-benefits/> for details. Table 4.2 provides a list of some of these resources.

Table 4.2 Microsoft Azure Resources

<i>Service</i>	<i>Site</i>
BizSpark	http://www.microsoft.com/bizspark/
Channel 9	http://channel9.msdn.com/
DreamSpark	https://www.dreamspark.com/
Imagine Cup	http://www.imaginecup.com
MSDN	http://msdn.microsoft.com/en-us

4.8 Summary

In this chapter, we learned about some advanced cloud service topics, including different types of endpoints, startup tasks, diagnostics, and service monitoring. We also learned how to use worker roles, and how to combine web roles and worker roles to construct multitiered cloud service solutions. We also examined role lifecycle in detail and practiced different ways for inter-role communications. Finally, we reviewed some of the community resources that you can leverage to improve your skills of cloud service development.

Chapter 5

Data Storage

Relational Database

In this chapter and the next, we will focus on data storage. We will first learn about relational databases on Microsoft Azure, and about NoSQL storages in Chapter 6. Before we go into further details, we will start with a brief overview of all data storage solutions provided by Microsoft Azure.

5.1 Microsoft Azure Data Storage Solutions

Microsoft Azure provides several data storage solutions for cloud service developers. You can choose to use any of the solutions based on the requirements and constraints of your project, or combine multiple solutions for your storage needs.

- Running an SQL Server on virtual machines

If you want to use the complete feature set of an SQL Server, you can run an SQL Server instances on Microsoft Azure virtual machines. When you provision a new virtual machine on Microsoft Azure, you can pick from several built-in SQL Server images (see Figure 5.1). Note that you will need to bring your own SQL Server license.

Note: Microsoft Azure provides high availability and disaster recovery capabilities for an SQL Server running on Microsoft Azure virtual machines. As this is a developer-oriented book, we will not dig deeper into these subjects. Interested readers may visit <http://msdn.microsoft.com/en-us/library/windowsazure/jj870962.aspx> for further details.

- SQL Database

An SQL Database is a relational database SaaS provided by Microsoft. Instead of managing any underlying infrastructures, you simply subscribe to the service and use it via an

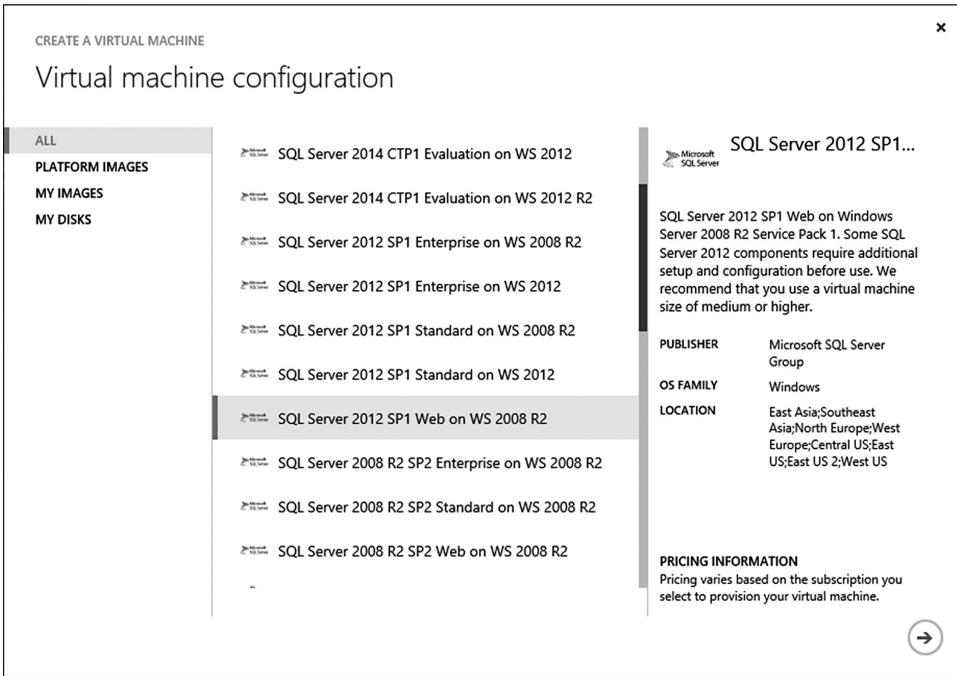


Figure 5.1 Built-in SQL Server images.

endpoint. Because the endpoint supports Tabular Data Stream (TDS) protocol, which is the same protocol used by an SQL Server, you can use an SQL Database just as if you were connected to a regular SQL Server instance, as shown in Figure 5.2. Although there are some differences between an SQL Database and an SQL Server, many of the existing SQL Server clients can be directly used against SQL Databases, given that database schemas on both services are compatible.

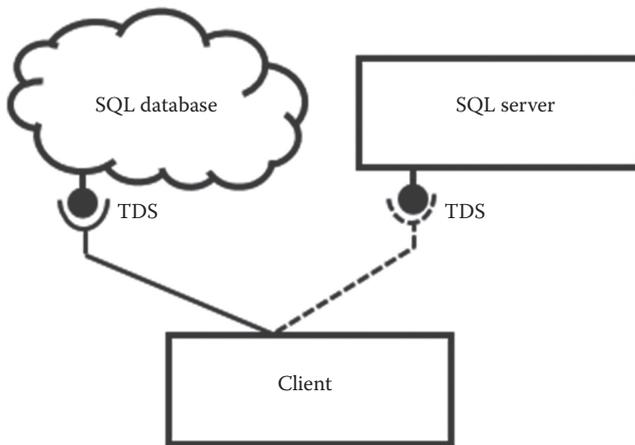


Figure 5.2 Protocol compatibility of SQL Database and SQL Server.

Because of the protocol-level compatibility between an SQL Server and an SQL Database, you can apply your existing SQL Server knowledge, skills, and tools to an SQL Database.

- Microsoft Azure Storage Services

Microsoft Azure provides a rich set of NoSQL storage services, including tables, virtual disks, BLOBs, and queues. These scalable services provide tremendous throughputs and outstanding performance to satisfy the requirements of storing, sharing, and accessing large amounts of data. You can access these services using their client libraries. You can also access these services via REST-style service calls and management API. We will study these services in Chapter 6.

5.2 SQL Database Overview

An SQL Database is a relational database service based on SQL Server technologies. It provides convenient, secure, self-maintained, highly available, and highly scalable relational database services for your cloud services.

- Maintainability

When using an SQL Database, because you only subscribe to a service endpoint, you do not need to manage any virtual or physical servers or install or update any software. Instead, you can provision a new database service in seconds and adjust your subscription at any time. In other words, instead of spending time on the tedious work of managing servers, you can put all your time and energy into service development and operation.

- Availability

An SQL Database automatically maintains two backups of your databases to ensure high availability. When there is a problem in the infrastructure, an SQL Database fails over to backups to ensure the availability of your data tier (Microsoft Azure's SLA guarantees 99.9% availability).

- Scalability

Once you share your data, you can horizontally scale your data tier as the data volume increases, in order to avoid data tier becoming the bottleneck of your system. On the other hand, because you only pay for what you use, you can achieve optimum system performance with minimum cost.

- Familiar development environments and tools

Just as we mentioned in Section 5.1, an SQL Database is highly compatible with an SQL Server. All your SQL Server-related knowledge, such as T-SQL, stored procedures, ADO. Net, Entity Framework, ODBC, and SQL Server Management Studio, can be applied to an SQL Database.

5.2.1 Differences between an SQL Database and an SQL Server

Although an SQL Server and an SQL Database are highly compatible, there are still some differences between the two. Some of these differences are caused by the differences in their architectures, and some others are caused by disparity in their feature sets. For developers, some noticeable differences include the following: an SQL Database does not support distributed queries; it does not support .NET CLR types; it does not support backup and restore (you can use Data Sync and Data-Tier Applications to achieve backups and restores—see Section 5.4),

and it requires a clustered index on every table. You may consult MSDN documents for a detailed list of the differences.

The biggest difference between an SQL Database and an SQL Server is that an SQL Database is an SaaS provided by Microsoft Azure, while an SQL Server often refers to the combination of a physical server and the database software (SQL Server) running on it. When you purchase a database server and install an SQL Server on it, you have total control over the hardware and the software, and all hardware resources are dedicated to you. When you subscribe to an SQL Database, however, all you get is an SQL Server-compatible endpoint. You don't need to or have access to the underlying computing and storage resources. For example, if you need more storage on your SQL Server, you can install additional data disks to the server, as long as there is enough space for the additional hardware. On the other hand, an SQL Database provides two service types: Web and Business. With the Web type, the maximum database size is 5 GB, and with the Business type, your database can grow up to 150 GB. These are hard limits that cannot be exceeded. If you need more storage, you will have to share your data on multiple databases.

At the time of writing this book, Microsoft Azure also provides an SQL Database Premium service (preview), which uses dedicated resource to back up your database service. Dedicated resource provides better performance predictability compared to the Web and Business editions. You may refer to www.windowsazure.com for further details.

Now let us learn the basics of using an SQL Database.

Example 5.1: SQL Database—online ordering system

Difficulty: ***

In this example, we create a simple online ordering system. The system allows users to manage their customers as well as their orders. Customer information and orders are saved in an SQL Database, which contains three tables: *Customer*, *Order*, and *OrderDetail*.

First, let us create the database.

1. Sign in to Microsoft Azure Management Portal.
2. Click on the **NEW** icon on the command bar. Then, select **DATA SERVICES**→**SQL DATABASE**→**CUSTOM CREATE**. On **NEW DATABASE—CUSTOM CREATE** dialog, enter *MyCustomerOrderDB* as **NAME**, select **New SQL database server** as **SERVER**, and then click the right arrow to continue, as shown in Figure 5.3.
3. On the next page, configure the user credentials for authentication, pick a region where you want the database to be hosted (usually this should be the same region as where your cloud services are hosted to reduce network latency), and then click the check button to create the database, as shown in Figure 5.4.

Note: By default, the new SQL database server does not allow access from any IP addresses. You will need to explicitly white-list IP addresses you want to grant accesses to. By checking the last checkbox on this dialog, you are granting access to the database server from any virtual machines hosting your own cloud services. You still need to authenticate to the server during connection.

4. After the database has been created, open its details page, where you can get a lot of database-related information, including the database server name. In this case, the server name is `dyo4zfnv67.database.windows.net`, which is autogenerated and cannot be changed, as shown in Figure 5.5. Click on the **Design your SQL database** link.

NEW SQL DATABASE - CUSTOM CREATE

Specify database settings

NAME
MyCustomerOrderDB

EDITION
WEB BUSINESS

LIMIT DATABASE SIZE (MAX SIZE)
1 GB

COLLATION
SQL_Latin1_General_CP1_CI_AS

SERVER
New SQL database server

2

Figure 5.3 Creating a new SQL Database.

NEW SQL DATABASE - CUSTOM CREATE

SQL database server settings

LOGIN NAME
haishi

LOGIN PASSWORD
●●●●●●●●

CONFIRM PASSWORD
●●●●●●●●

REGION
West US

ALLOW WINDOWS AZURE SERVICES TO ACCESS THE SERVER.

1

Figure 5.4 Setup credentials for the new SQL database.

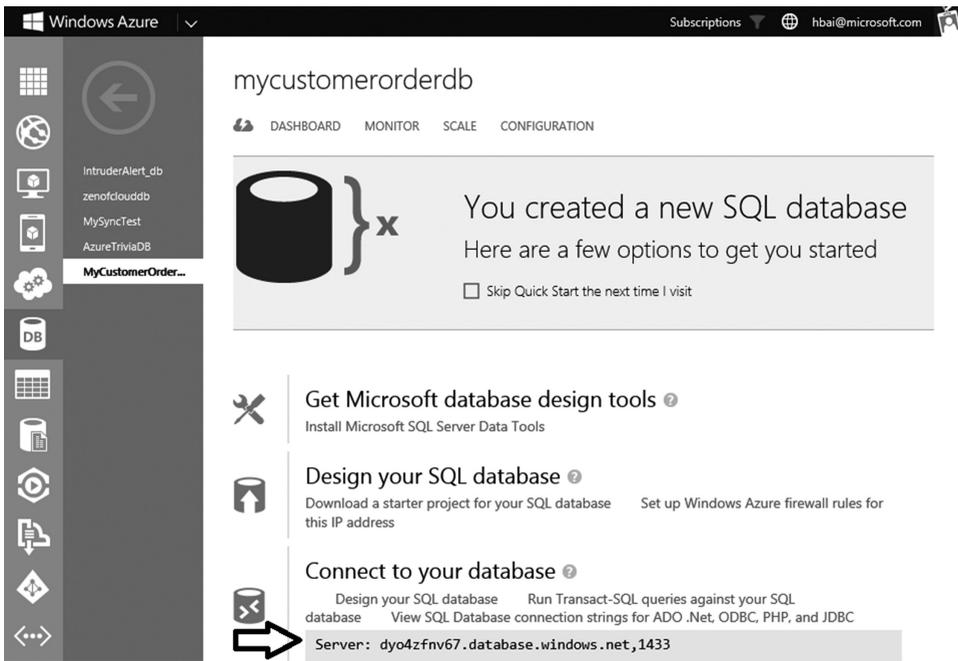


Figure 5.5 SQL Database details page.

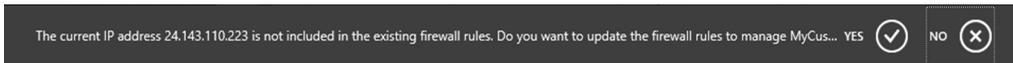


Figure 5.6 Adding firewall rule for current computer.

- The portal will prompt you to add the IP address of your current computer to the firewall rules so that you can manage an SQL Database from your current computer. Click **YES** to continue, as shown in Figure 5.6.

Note: You can manage an SQL Database server firewall settings on its **CONFIGURE** page. Note that the database management page and the server management page are different. To switch to the server view, click **SQL DATABASE** in the navigation bar, and then switch to **SERVERS** list, or click on the server name in the **SERVER** column in the **DATABASES** list, as shown in Figure 5.7.

- Once the firewall rule is changed, you will be asked if you want to continue with database design. Click **YES** to continue, as shown in Figure 5.8.

Note: If your browser blocks the pop-up, expand Options for this site and select Always allow (see Figure 5.9). Then repeat step 4.

sql databases

DATABASES **SERVICES** SYNC PREVIEW

NAME	STATUS	LOCATION	SUBSCRIPTION	SERVER
IntruderAlert_db →	✓ Online	East Asia	TE Account - Haishi Bai	azfwaq514k
zenofclouddb	✓ Online	West US	TE Account - Haishi Bai	d9i4konmd8
MySyncTest	✓ Online	West US	TE Account - Haishi Bai	ddqervzo4r
AzureTriviaDB	✓ Online	West US	TE Account - Haishi Bai	js6cqu8su
MyCustomerOrderDB	✓ Online	West US	TE Account - Haishi Bai	dyo4zfrv67

Figure 5.7 Switching to server view.

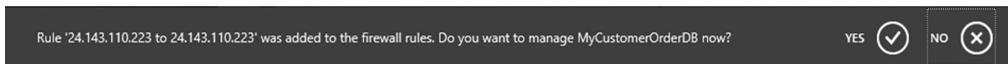


Figure 5.8 Confirmation message of firewall rule change.



Figure 5.9 Browser pop-up settings.

- An SQL Database management UI will open in a pop-up window (you will need to install/enable Silverlight). Use the user credentials you have entered in step 3 to log on, as shown in Figure 5.10.
- Click the **New table** link, as shown in Figure 5.11.
- Create a new Customer table, which contains an ID column, a Name column, and an Address column, with ID column being the primary key. The table schema is shown in Figure 5.12. After editing, click on the **Save** button to save the table schema.
- Similarly, create an *Order* table, as shown in Figure 5.13.

Tip: Click on the Tables link at the top of the screen to return to the table list.

- Once the table schema is saved, click on the **Indexes and Keys** link. Then, click on the **Add a foreign key relationship** link on the page.
- In the pop-up frame, specify the *ID* column in the *Customer* table as the foreign key to *CustomerID* column (as shown in Figure 5.14). Click the **Save** button to save the foreign key.
- The foreign key constraint should look like the diagram in Figure 5.15.
- Create the *OrderDetails* table. Then, set the *ID* column of the *Order* table as the foreign key to its *OrderID* column, as shown in Figure 5.16.

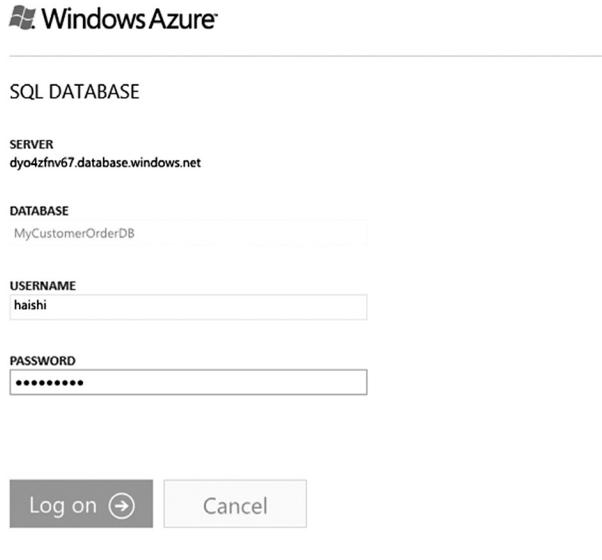


Figure 5.10 SQL Database log on page.

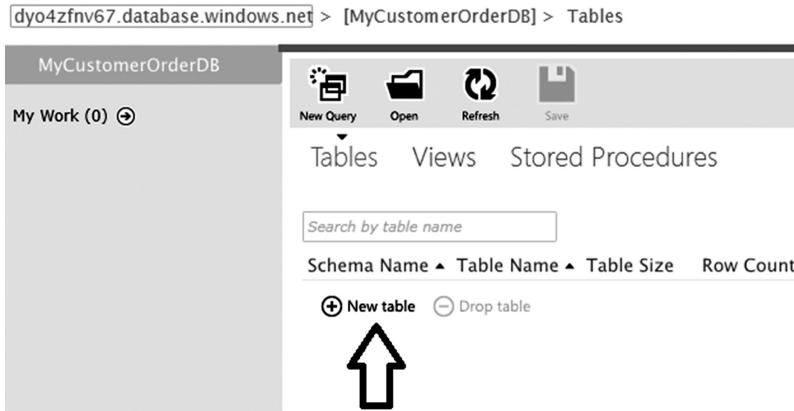


Figure 5.11 New table link to create a new table.

15. Click the **Design** link at the lower-left corner of the page, or the **Tables** link at the top of the page to return to the tables list. Hover your mouse to the right of any table, and click the **Dependencies** link, as shown in Figure 5.17.
16. On the Dependencies view, click the **Show all dependencies** link (see Figure 5.18). This view allows you to examine dependency relationships among your tables (you can zoom in or out by dragging the slider bar at the lower-left corner).
17. Now we are ready to create two default records in the Customer table. Click on the **New Query** icon at the top of the screen. Then, in the online T-SQL editor, enter the SQL statements as shown in Figure 5.19. Then, click the **Run** icon to execute the script. The execution result is shown in Figure 5.19.
18. As you have experienced the basics of using an SQL Database management UI, you can now close the pop-up window.

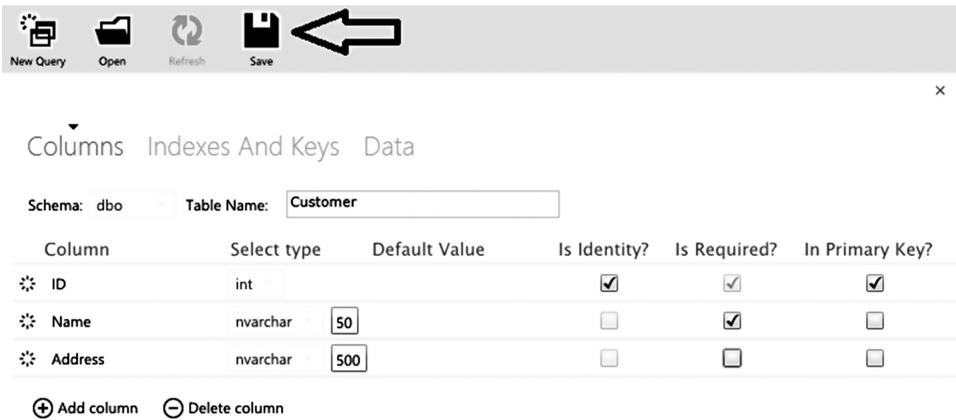


Figure 5.12 Create Customer table.

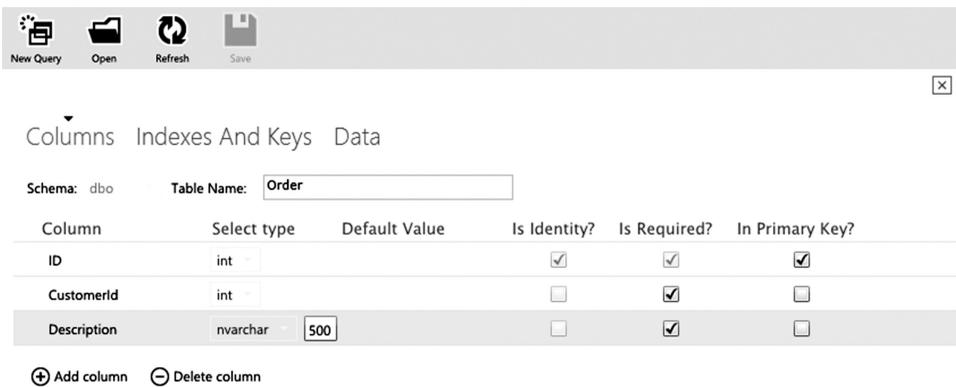


Figure 5.13 Order table.



Figure 5.14 Defining the foreign key.

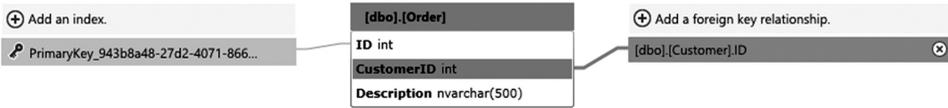


Figure 5.15 Foreign key constraint.

Columns Indexes And Keys Data

Schema: dbo Table Name: OrderDetail

Column	Select type	Default Value	Is Identity?	Is Required?	In Primary Key?
ID	int		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
OrderID	int		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ItemName	nvarchar	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ItemPrice	float	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ItemQuantity	float	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

+ Add column - Delete column

Figure 5.16 OrderDetail table.

Tables Views Stored Procedures

Search by table name

Schema Name	Table Name	Table Size	Row Count	
dbo	Customer	0.00 KB	0	
dbo	Order	0.00 KB	0	
dbo	OrderDetail	0.00 KB	0	<input type="button" value="Edit"/> <input type="button" value="Dependencies"/>

+ New table - Drop table

Figure 5.17 Dependencies link.

Next, we will create a new online cloud service, which allows users to manage customers and orders online.

1. Launch Visual Studio as an administrator. Create a new cloud service project with an ASP.NET MVC 4 Web Role (using the Internet Application template).
2. Right-click the Web Role project, and select the **Add→New Item** menu. Then, select the **Data→ADO.NET Entity Data Model** template. Enter *MyCustomerDB.edmx* as **Name**, and click the **Add** button to add the data model.

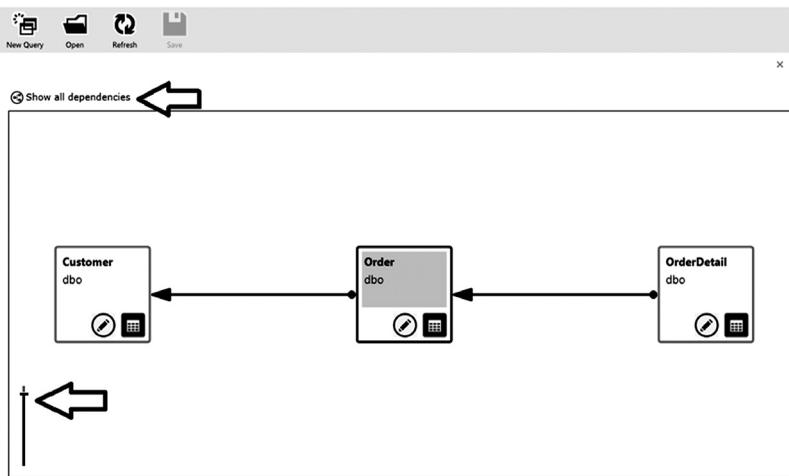


Figure 5.18 Dependencies view.

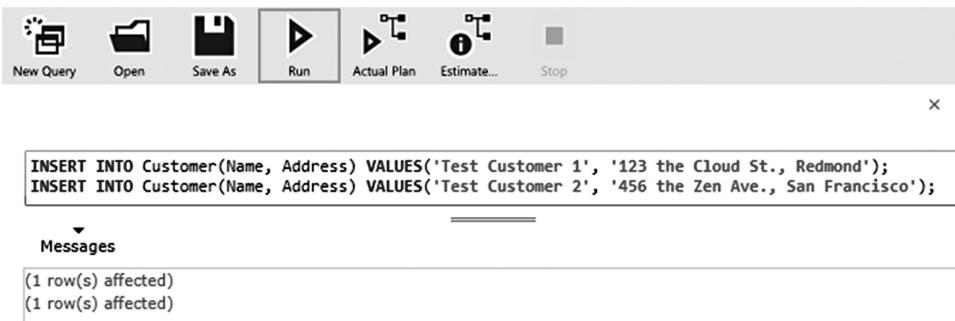


Figure 5.19 Sample T-SQL execution result.

3. On **Entity Data Model Wizard** dialog, select **Generate from database**, and then click the **Next** button to continue.
4. On the next window, click on the **New Connection** button. Then, on the **Choose Data Source** pop-up, select **Microsoft SQL Server**, and then click the **Continue** button, as shown in Figure 5.20.
5. On **Connection Properties** dialog, enter your SQL Database server name (as in step 4). Then, select the **Use SQL Server Authentication** option (an SQL Database does not support Windows Authentication) and enter your user name and password (as you entered in step 3). Check **Save my password** checkbox. Then, select *MyCustomerOrderDB* in the **Select or enter a database name** field, and click the **OK** button to continue (Figure 5.21).
6. Back on the **Entity Data Model Wizard**, select **Yes, include the sensitive data in the connection string** option, and click the **Next** button to continue.
7. Select all tables. Then click on the **Finish** button to continue, as shown in Figure 5.22.
8. If script security warnings show up, click **OK** to continue.
9. Rebuild the solution.
10. In the Web Role project, right-click the **Controllers** folder, and select the **Add→Controller** menu.

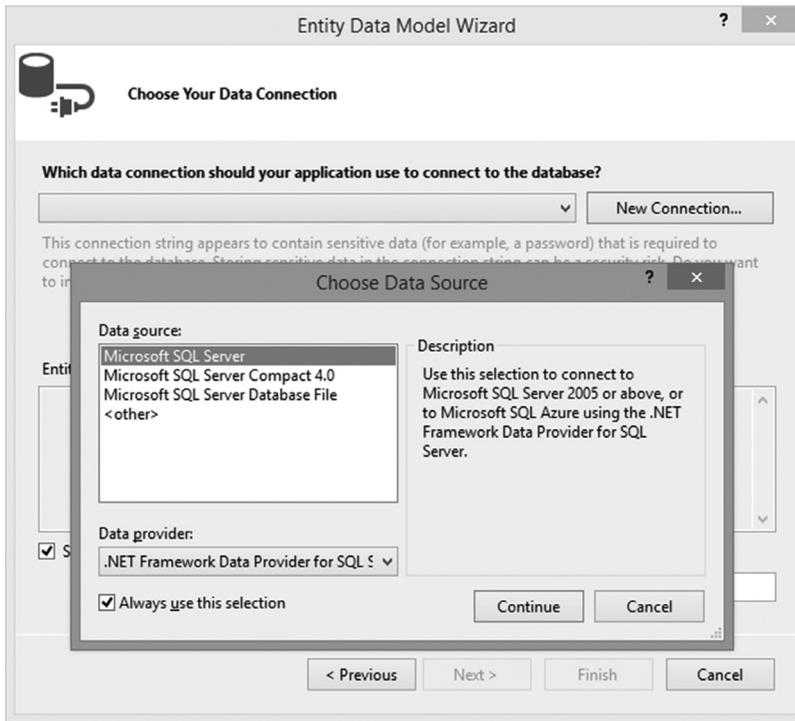


Figure 5.20 Entity Data Model Wizard.

11. On **Add Controller** dialog, set the controller name as *OrderController*. Select **MVC controller with read/write actions and view, using Entity Framework** template. Then, select *Order* as **Model class**, and *MyCustomerOrderDBEntities* as **Data context class**. Click the **Add** button to complete the operation, as shown in Figure 5.23.
12. Press F5 to launch the application. Once the website is launched, enter the address *http://12.0.0.1:81/Order* (your port might be different). Click the **Create New** link, as shown in Figure 5.24.
13. On the new order page, pick a customer, and enter a description for the order. Then, click on the **Create** button, as shown in Figure 5.25.
14. This takes you back to the orders view, where you can add, edit, or delete orders, as shown in Figure 5.26.
15. Similarly, add a *CustomerController* and an *OrderDetailController*.
16. Modify **Views\Shared_Layout.cshtml** to add links to the entities under the <nav> tag:

```

...
<nav>
<ul id="menu">
  <li>@Html.ActionLink("Home", "Index", "Home")</li>
  <li>@Html.ActionLink("Customers", "Index", "Customer")</li>
  <li>@Html.ActionLink("Orders", "Index", "Order")</li>
  <li>@Html.ActionLink("Order Details", "Index",
    "OrderDetail")</li>
...

```

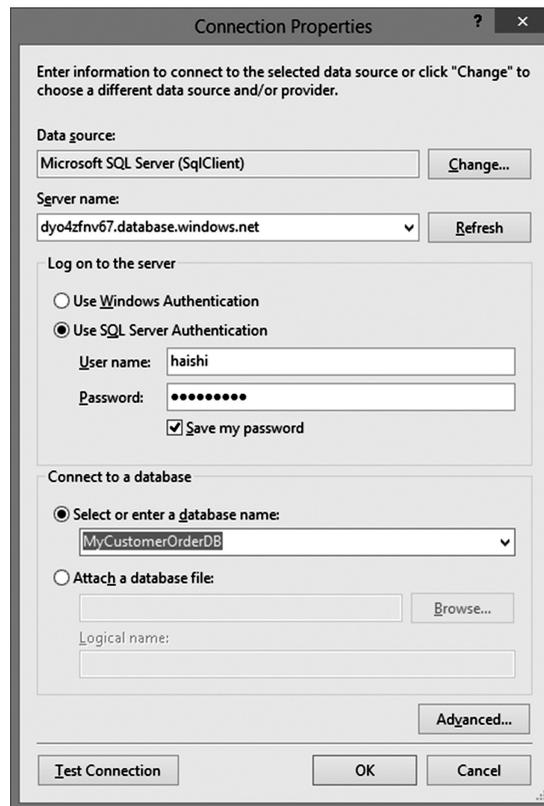


Figure 5.21 Connection Properties dialog.

17. The completed system is shown in Figure 5.27.

By now, we have successfully made use of an SQL Database in our cloud service. If you had worked with ASP.NET and Entity Framework before, you would have discovered that the steps are identical to the steps of using an SQL Server database. In the next example, we will continue to improve our application.

5.3 SQL Database Management and Optimization

We used an online designer to design the database schema in Example 5.1. In this section, we will learn to use several different tools for managing and optimizing an SQL Database.

5.3.1 SQL Server Management Studio

In this section, we will use SQL Server Management Studio to manage Microsoft Azure SQL Database. SQL Server Management Studio (SSMS) is one of the most commonly used tools for managing an SQL Server. Now you can use it to manage an SQL Database as well. Note that you will need SQL Server 2012 Management Studio Express (SSMSE)

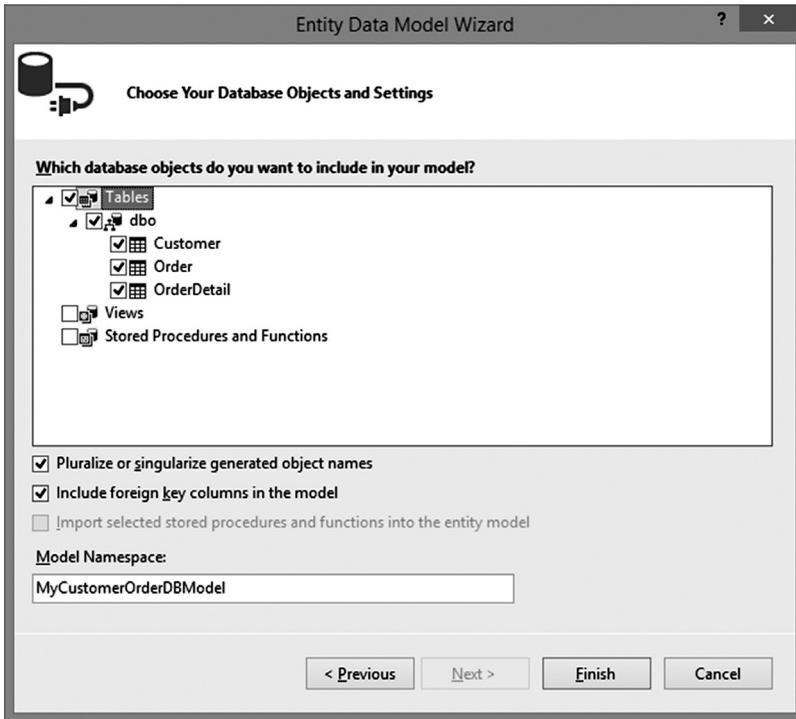


Figure 5.22 Adding all tables.

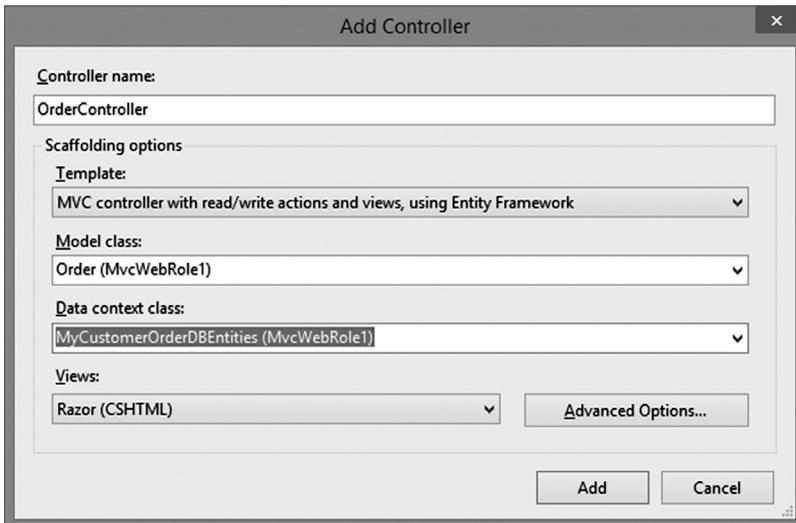


Figure 5.23 Add Controller dialog.

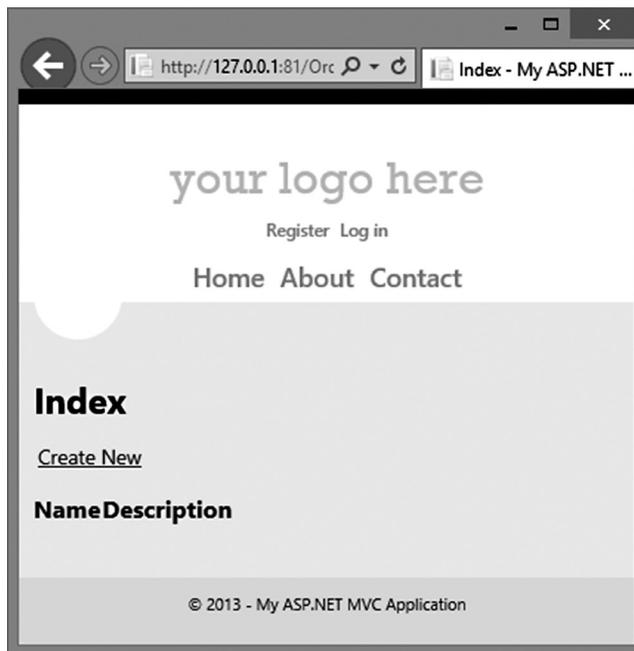


Figure 5.24 Orders view.

version, which can be downloaded for free from <http://www.microsoft.com/en-us/download/details.aspx?id=29062>. You should download either the 32-bit version or the 64-bit version based on your machine environment.

Example 5.2: SQL Database—the order view

Difficulty: **

In this example, we continue with Example 5.1 and use SSMSE to create a new *OrderView*, which automatically calculates the amount of an order based on its details.

1. Launch SSMS or SSMSE.
2. Log in to the database in Example 5.1. The way to log in is similar to that for an SQL Server. Note that you need to select **SQL Server Authentication**, as shown in Figure 5.28.
3. Once signed in, you can browse various database resources, such as tables and views, in the **Object Explorer**. At the time of writing this book, the tool does not provide graphic UI for managing database schemas and objects. If you want to use a graphic UI, you can use SSDT, which will be introduced in the next section. Here we will use a script to create a new view. In the **Object Explorer**, right-click the **Views** node, and select the **New View** menu, as shown in Figure 5.29.
4. In the script editor, enter and execute the following script:

```
CREATE VIEW OrderView AS
SELECT [Order].ID, CustomerID, Description,
       (SELECT SUM(ItemPrice * ItemQuantity)
        FROM OrderDetail WHERE OrderID = [Order].ID) As Amount
FROM [Order]
```



Figure 5.25 New order page.

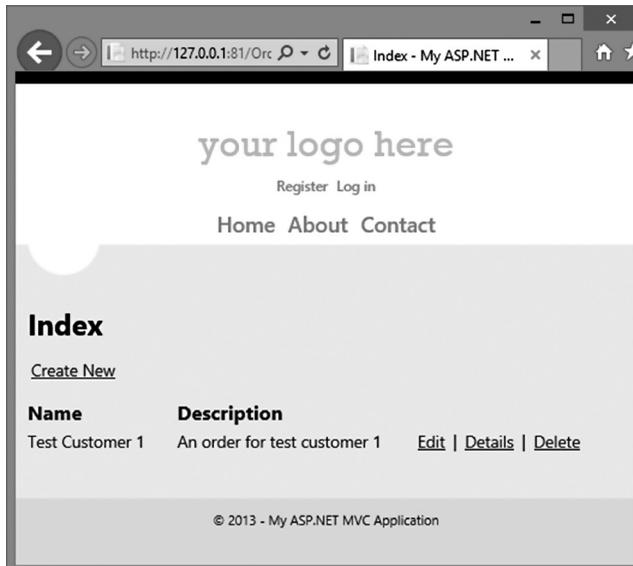


Figure 5.26 Orders page.

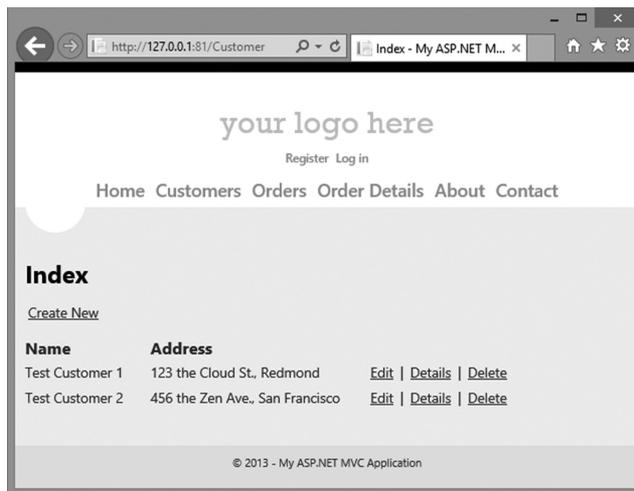


Figure 5.27 Completed online ordering system.

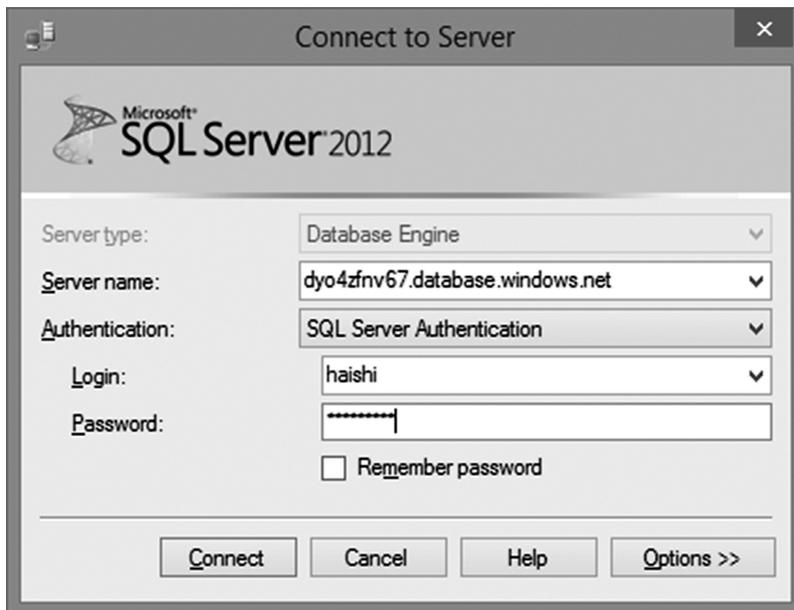


Figure 5.28 Use SSMSE to log in to SQL Database.

5. After the view has been created, you can try to query it. For example,

```
SELECT * FROM OrderView
```

As you can see, you can manage an SQL Database using SSMS just as if you were managing an SQL Server instance. Most of your knowledge of T-SQL still applies. A detailed introduction of T-SQL is beyond the scope of this book.

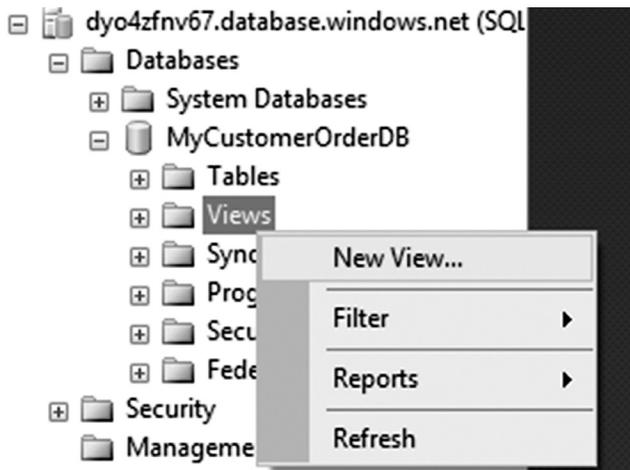


Figure 5.29 Create a new view in SSMSE.

5.3.2 Microsoft SQL Server Data Tools

Microsoft SQL Server Data Tools (SSDT) is a database management tool integrated with Visual Studio. You can directly manage and query databases within Visual Studio using SSDT, which you can download from Microsoft Azure Management Portal, as shown in Figure 5.30.

Once the tool has been downloaded and installed, you can access it from Visual Studio's **VIEW→SQL Server Object Explorer** menu. The tool is very similar to SQL Server Management Studio and very intuitive, so I shall not go into further detail. Figure 5.31 is a screenshot of SSDT.

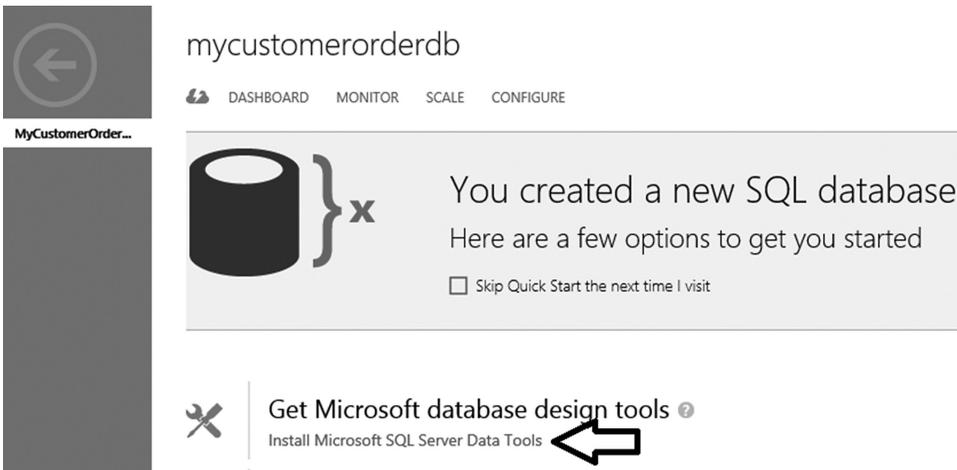


Figure 5.30 Download SSDT.

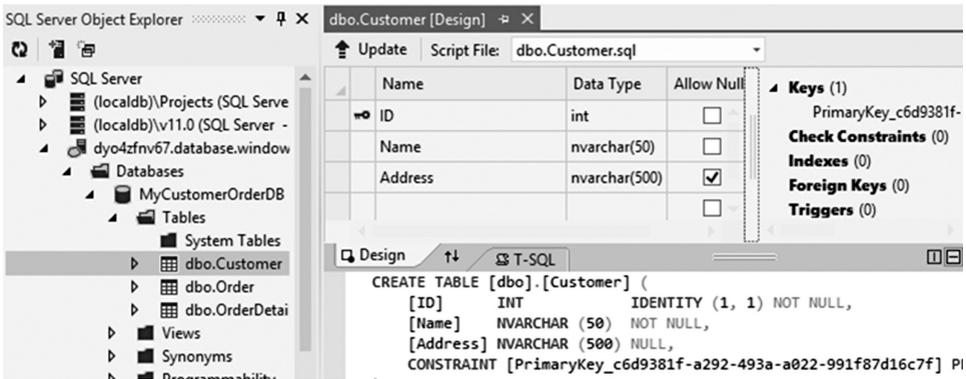


Figure 5.31 SSDT UI.

5.3.3 Dynamic Management Views

Dynamic Management Views (DMVs) provide dynamic information of database service statuses, allowing you to monitor the health of the service, diagnose problems, and optimize system performance. For example, through these views, you can examine various types of information such as database sizes, number of connections, query speed, and index usages. Next, we will learn how to use DMVs with an example.

Example 5.3: Use Dynamic Management Views

Difficulty: *

1. Launch SSMS. Sign in to the SQL Database you want to work with.
2. In **Object Explorer**, right-click on the database, and select the **New Query** menu.
3. To query DMVs, the database user requires VIEW DATABASE STATE privilege. Let us set it up first:

```
GRANT VIEW DATABASE STATE TO <user>;
```

4. Then, let us query for the current database size (in MB) via *dm_db_partition_stats* view:

```
SELECT SUM(reserved_page_count)*8.0/1024 FROM sys.dm_db_partition_
stats;
```

5. Now let us query for the connection information by joining multiple DMVs:

```
SELECT
  e.connection_id,
  s.session_id,
  s.login_name,
  s.last_request_end_time,
  s.cpu_time
```

```

FROM
    sys.dm_exec_sessions s
INNER JOIN
    sys.dm_exec_connections e
ON
    s.session_id = e.session_id;

```

Of course, we barely scratched the surface of DMVs here with only a quick run of a couple of views. To get more information on these management views, you may visit <http://msdn.microsoft.com/en-us/library/windowsazure/ee336238.aspx>.

Note: Samples in this section come from MSDN:

<http://msdn.microsoft.com/en-us/library/windowsazure/ff394114.aspx>

5.3.4 Query Optimization

Query optimization is a common problem we often face. Sometimes, a change of a single index or a single query may have significant impact on system performance. SQL optimization requires specialized knowledge and tools, which cannot be covered in full in this book. Instead, we would like to emphasize on two points: the general process of query optimization, and optimization tools provided by Microsoft Azure.

■ The General Process of Query Optimization

A systematical approach is the key to successful and effective query optimization. We have seen too often that developers spend much time and effort in query optimization without clearly knowing the effectiveness of the exercises, or knowing when to stop. Typically, at the beginning phase of optimization, you can gain much improvement with less effort. However, after the principal problem has been resolved, it becomes increasingly harder to gain additional improvement. As the return rate decreases exponentially, it is important to know when the performance is “good enough” so the optimization work can stop. In other words, in terms of query optimization, you should always shoot for “good enough” instead of “perfect,” because the price of “perfect” is often unacceptable.

An effective query optimization process is illustrated in Figure 5.32.

– Measure

Quantitative measurements are very important to query optimization. Ambiguous assessments such as “system is slow” or “query is too complex” are insufficient for establishing performance baselines, which provide objective benchmarks to evaluate the effectiveness of the optimization work. Only based on accurate measurements can you establish reliable baselines, to which you can compare improved performance and decide if the performance is “good enough.”

– Establish/Refine goals

The goals of query optimization are driven by both technical factors and business factors. Understanding customers’ expectation in performance is crucial for establishing appropriate performance goals. A faster system is not automatically a better system. Instead, system performance should be in tune with the rhythm of how business is

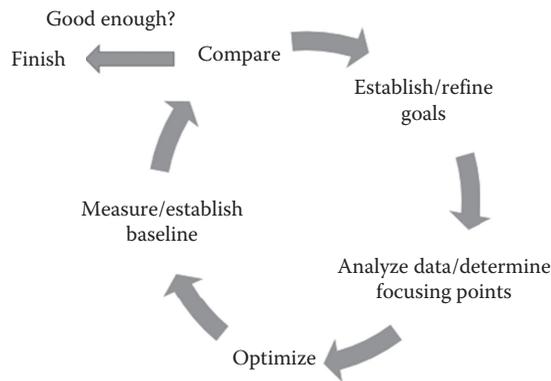


Figure 5.32 Systematical approach of query optimization.

conducted by the customers. Blindly chasing after ultimate speed increases cost and risk, without necessarily bringing the best returns.

- Analyze data/Determine focusing points

Measured data help developers to accurately locate system bottlenecks based on objective evidence instead of hunches and guesses. Experiences tell us that system performance is often decided by how an application layer is designed and implemented. Instead of just focusing on query optimizations, developers should also carefully examine whether the application is using the data layer effectively. For instance, is the application layer making repetitive queries? Is it storing too much unnecessary data? Is it delegating business logic to the data layer? All these are good questions to ask to identify bottlenecks in the application layer.

Note: Different people have different opinions on whether business logic should be implemented on databases. Many systems use stored procedures to implement business logic. This is often not a good approach. Stored procedures can be used to facilitate complex data handling, but they should not be the carriers of business logic. Implementing business logic in stored procedures blurs the boundary between the business layer and the database layer. This is a violation of separation of concerns principle. It binds business logic to a specific database platform.

- Optimize

This is the part where you apply various tools and techniques to optimize queries.

- Compare

After one round of optimization, you need to measure again and compare newly acquired data with established goals. When the result is “good enough,” exit the loop.

- Query optimization tools provided by Microsoft Azure

Common tools for optimizing an SQL Server include SQL Profiler, performance counters, and SQL Server Management Studio. Although you can monitor many database-related performance counters such as deadlocks, successful connections, and failed connections on Microsoft Azure Management Portal, you can find richer tools on the SQL Database management UI. In the following example, we will learn how to analyze a query using an SQL Database management UI.

Example 5.4: Use SQL Database Management UI

Difficulty: *

1. Log in to SQL Database management UI (see Example 5.1).
2. Click on the **Administration** link at the lower-left corner.
3. Click on the **Query Performance** link to see the performances of recent queries (see Figure 5.33).
4. Click on any query to examine its performance details. In this example, we have inserted a couple of order details for the order and have executed query: `SELECT * FROM OrderView`. Click on the **Query Plan** link to see a graphical presentation of the query plan. You can examine each execution step in detail and observe the percentage of resources the step uses. Figure 5.34 shows how much CPU is used in each step of the chosen query. Figure 5.34 shows how much CPU is used in each step of the chosen query.
5. Click on any box to further examine the details of the step. There are many more features on this page, which interested users may explore.

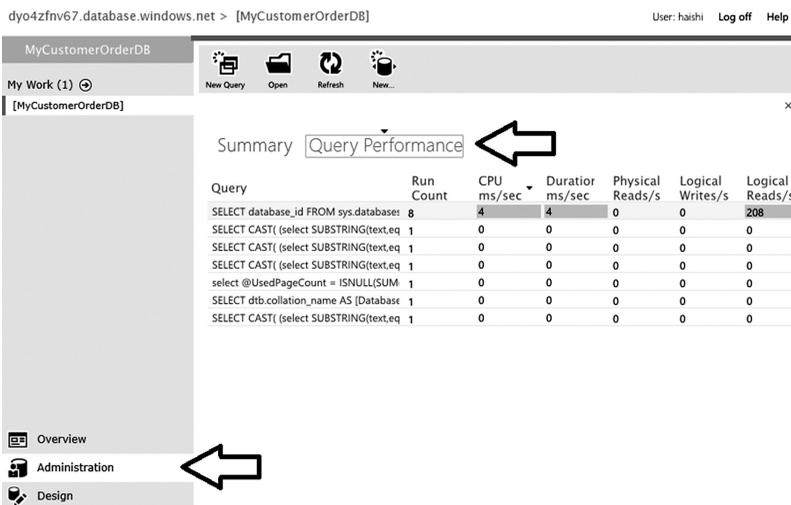


Figure 5.33 Query Performance page.

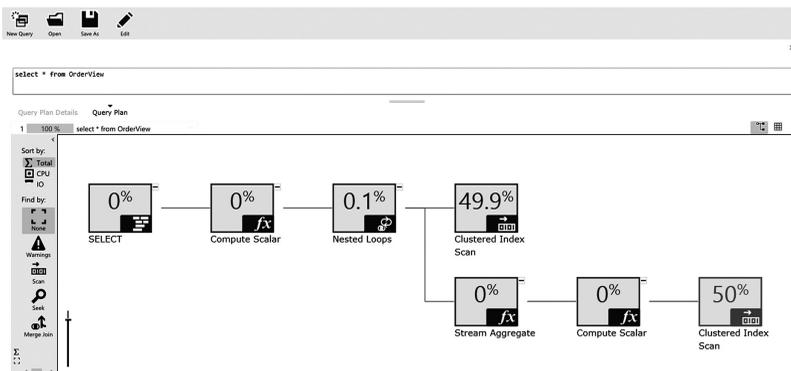


Figure 5.34 Execution plan of a query.

5.4 Data Sync and Migration

An SQL Database provides a series of data migration and synchronization mechanisms. For example, you can deploy a local SQL Server database to an SQL Database. You can also package and download an SQL Database to a local SQL Server database. You can also use Data Sync Service to back up a database, or to sync data among multiple databases.

5.4.1 Data-Tier Application

Data-tier Application (DAC) encapsulates all database objects an application uses, including tables, indexes, views, stored procedures, and users. Instead of directly working on databases, developers can work with this abstraction layer to design the database schema, and then hand the definition to database administrators to deploy to any databases that support DAC. DAC not only eliminates the complexity of maintaining database scripts, but also reduces interdependencies between developers and database administrators. In addition, DAC also supports automatic data migration when the database schema changes so you do not lose your data during schema updates. Last but not least, DAC supports version controls, allowing developers and administrators to manage DAC versions just like managing source code versions.

When working with DAC, you often need to deal with two file types: *.bacpac* and *.dacpac*.

- *.dacpac*

.dacpac contains the schema of a database. Its main purpose is for deploying the database schema to different environments and for updating the database schema.

- *.bacpac*

.bacpac contains both the schema and the data of a database. The schema, which is encoded as JSON data, is the same as that in *.dacpac*. Logically a *.bacpac* is a backup of a database. Its main purpose is database migration.

Example 5.5: Use *.bacpac* and *.dacpac*

Difficulty: ***

In this example, we download and import our SQL Database in Example 5.1 to a local SQL Server.

1. Log in to Microsoft Azure Management Portal. Open the details view of the database in Example 5.1 (if the page you see is not the one in Figure 5.35, click on the cloud icon with the little flash).
2. Click on the **EXPORT** icon on the command bar, as shown in Figure 5.35.
3. On **EXPORT DATABASE** dialog, choose **Create a new storage account** in the **BLOB STORAGE ACCOUNT** dropdown. Enter database server login credentials, and then click the **NEXT** icon to continue, as shown in Figure 5.36.
4. On the next page, create a new storage account. All you need to do is to enter a URL identifying the account, and a container name (only lowercased letters are allowed) to hold the exported data. As we have not introduced Microsoft Azure Storage services yet, you can simply think of a container as a space where you can save files. Uncheck the **ENABLE GEO-REPLICATION** checkbox, and then click the check button, as shown in Figure 5.37.
5. Launch Visual Studio as an administrator. Then, use the **VIEW→Server Explorer** menu to open the Server Explorer.
6. Expand the **Microsoft Azure** node to your storage account (as shown in Figure 5.38). Double click on the container you have created in step 4 to view its contents. Then, right-click the only file in the container and select the **Save As** menu to save the *.bacpac* file to a local folder of your choice.

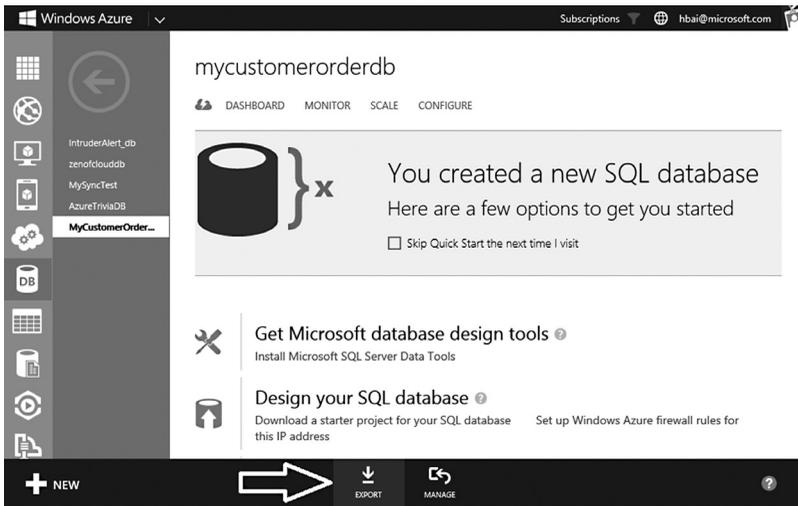


Figure 5.35 Export database.

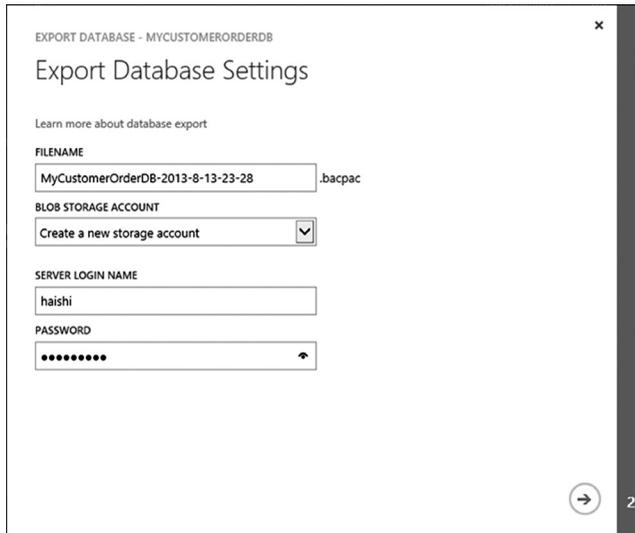


Figure 5.36 Export database to a new storage account.

7. Launch SQL Server Management Studio. Connect to your local SQL Server, such as a local SQL Express database.
8. Right-click on the **Databases** node, and select the **Import Data-tier Application** menu, as shown in Figure 5.39.
9. On **Import Data-tier Application** dialog, click the **Next** button to continue.
10. On the **Import Settings** page, select the **Import from local disk** option, and pick the file you downloaded in step 6. Note that you can actually directly import from a Microsoft Azure Storage account. The only reason we did the manual download step is to avoid manually inputting account information, which we will explain in the next chapter. After the file is chosen, click the **Next** button to continue, as shown in Figure 5.40.

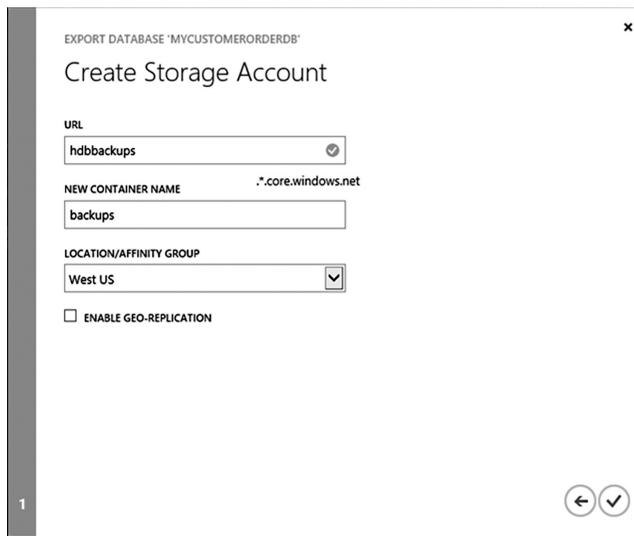


Figure 5.37 Create a new storage account.

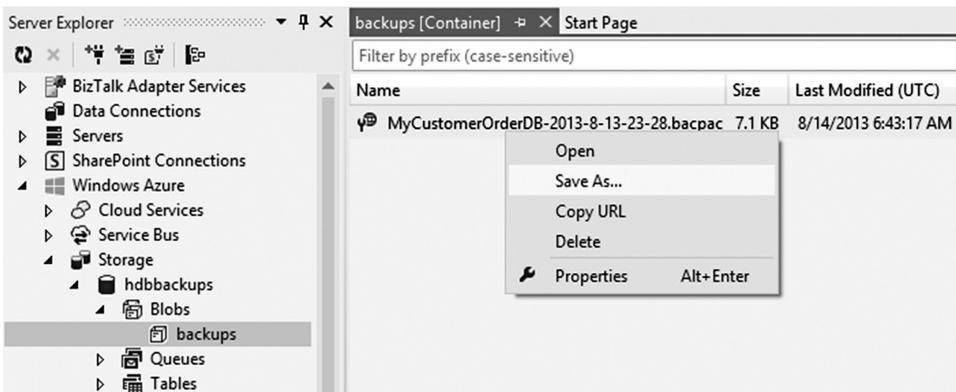


Figure 5.38 Save the .bacpac file to a local folder.

11. On the **Database Settings** page, click the **Next** button to continue.
12. Click the **Finish** button to start import (ironic, isn't it?). After import is done, click the **Close** button to close the wizard.
13. Now you can try out a few queries, such as

```
SELECT * FROM OrderView
```

- to verify that both database schema and data have been successfully copied.
14. You can also deploy your local SQL Server to the SQL Database. In SQL Server Management Studio, right-click on the database, select the **Tasks**→**Deploy Database to Microsoft Azure** menu (SQL Database was called SQL Azure), and then follow the wizard to finish the deployment.

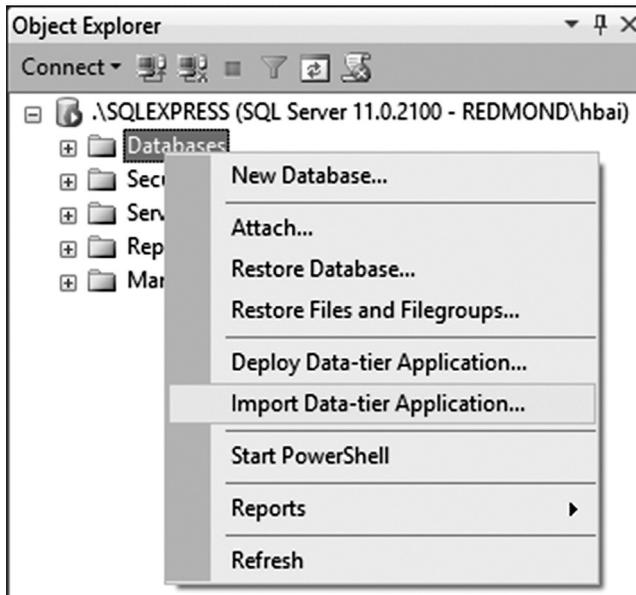


Figure 5.39 Import Data-tier Application menu.

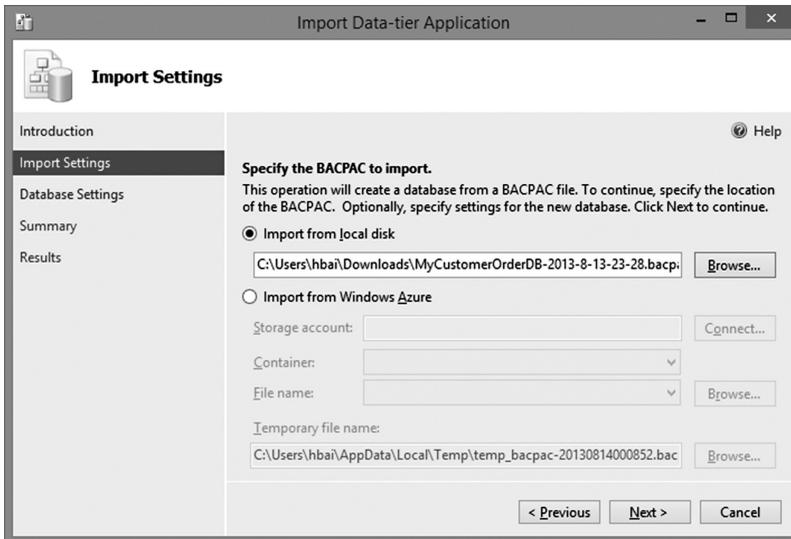


Figure 5.40 Import a local .bacpac file.

5.4.2 Data Sync

Following the previous example gives us two identical databases on the SQL Server and SQL Database. Next, we will learn how to use the SQL Data Sync service to keep the two databases in sync.

Note: At the time of writing this book, SQL Data Sync is still in preview. The released version of the service may differ.

Example 5.6: Data Sync between SQL Server and SQL Database

Difficulty: ****

- Before you send up the Data Sync agent, you need to make sure the target database server has the following prerequisites installed:
 - .Net Framework 4
 - Microsoft SQL Server CLR Types for Microsoft SQL Server 2012 (x86)
- Log in to Microsoft Azure Management Portal. Select the database for which you want to set up the sync. Then, click the **ADD SYNC** icon on the command bar, and select the **Add Sync Agent** menu, as shown in Figure 5.41.
- On **New Sync Agent** dialog, enter a name for the sync agent. Choose a region for the agent (to ensure performance, you should pick the same region as where your SQL Database resides). You can also download and install the local agent from this page by clicking on the **install one here** link, as shown in Figure 5.42 (the download link is <https://go.microsoft.com/fwlink/?LinkID=223590&clcid=0x409>. Note that this is the preview version; you may consult the companion site or use BING to find out the link to the final released version).
- After the agent has been created, click on the **MANAGE KEY** icon on the command bar. Then, on **Manage access key** dialog, click the **Generate** button to generate a new access key. Copy the access key to the clipboard by clicking on the copy button to the right of the **AGENT ACCESS KEY** field, as shown in Figure 5.43.
- Launch **Microsoft SQL Data Sync Agent Preview**. Click on the **Submit Agent Key** button at the top of the screen. Then, on the pop-up dialog, paste in the agent key and click the **OK** button to continue.
- Back in the main window, click on the **Register** button to register the local SQL Server. In the pop-up dialog, enter connection information to the local database, and then click on the **Save** button, as shown in Figure 5.44.
- On Microsoft Azure Management Portal, click on the **ADD SYNC** icon on the command bar again, and select the **New Sync Group** menu, as shown in Figure 5.45.

sql databases

DATABASES SERVERS SYNC PREVIEW

NAME	STATUS	LOCATION	SUBSCRIPTION	SERVER
MyCustomerOrderDB	→ ✓ Online	West US	TE Account - Haishi Bai	dyo4zfrn67

IMPORT EXPORT MANAGE DELETE **ADD SYNC**

New Sync Group
New Sync Agent

Figure 5.41 Add New Sync Agent.

New Sync Agent PREVIEW ✕

Register a client sync agent.
If you have not yet installed a local client sync agent, you can install one here.

NAME

REGION

✓

Figure 5.42 New Sync Agent dialog.

Manage access key PREVIEW ✕

To register the client agent:
 1. Generate the agent access key and copy it to your clipboard.
 2. Start the client agent.
 3. Paste the key into the client sync agent.

AGENT NAME

AGENT ACCESS KEY
 📄 Generate

✓

Figure 5.43 Manage access key.

8. On **Sync group basic settings** dialog, enter a name for the Sync Group, pick its region, and click the next icon to continue, as shown in Figure 5.46.
9. Define a sync hub. A Sync Group can include one hub database and multiple reference databases. On this page, pick the SQL Database as the hub database and enter the corresponding credentials. You can also choose the conflict resolution policy on this page. When conflicts are detected, you can specify whether the hub database should win or the reference (client) database should win. Here, we will use **Hub Wins**, as shown in Figure 5.47.
10. On **Add reference database** dialog, expand the **REFERENCE DATABASE** dropdown box. You will see that the sync agent we just registered shows up in the list. Pick the agent, as shown in Figure 5.48.
11. If the local database uses Windows authentication, you do not need to enter the user name or password. Otherwise, you need to enter the correct credentials to your local database. You can also pick the sync direction, which can be bidirectional, from hub or to hub. Click on the check button to continue, as shown in Figure 5.49.
12. On Microsoft Azure Management Portal, click on the Sync Group to open its details page. Then, switch to the **SYNC RULES** page. On the **SYNC RULES** page, click the

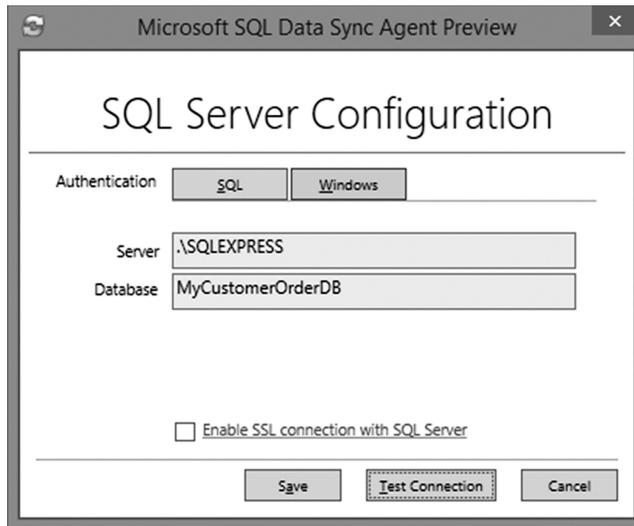


Figure 5.44 Register local database to a Sync Group.

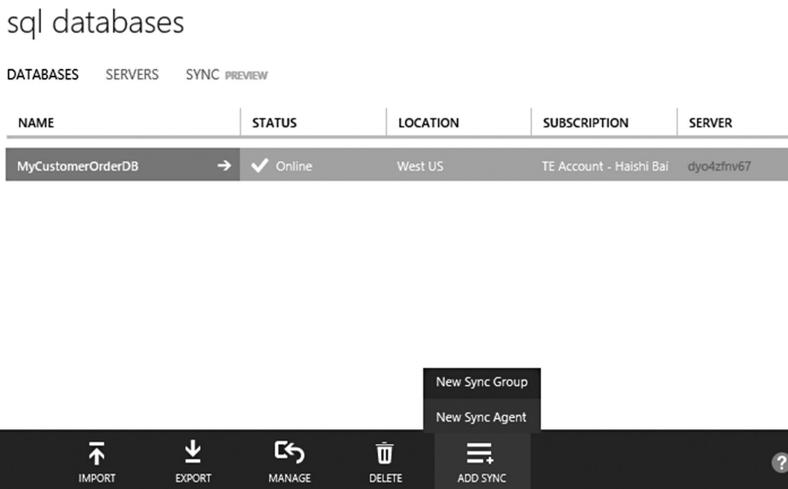


Figure 5.45 Create a New Sync Group.

DEFINE SYNC RULES link. We will pick the data that we want to sync on the **DEFINE DATASET** dialog. First, select the SQL Database (see Figure 5.50).

13. Click the **SELECT** icon on the command bar, and then click the **Select all the columns in all the tables** menu.
14. Click the **SAVE** icon.
15. Go to the **CONFIGURE** page. Set **AUTOMATIC SYNC** to **ON**, and **SYNC FREQUENCY** to 5 min. Click the **SAVE** button to save the configuration, as shown in Figure 5.51.
16. Now our SQL Server and SQL Database are synced every 5 min. To test it out, insert a new record in the local database, for example,

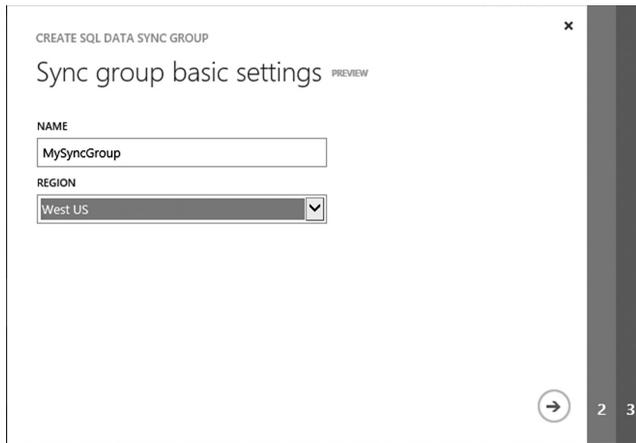


Figure 5.46 Sync Group basic settings.

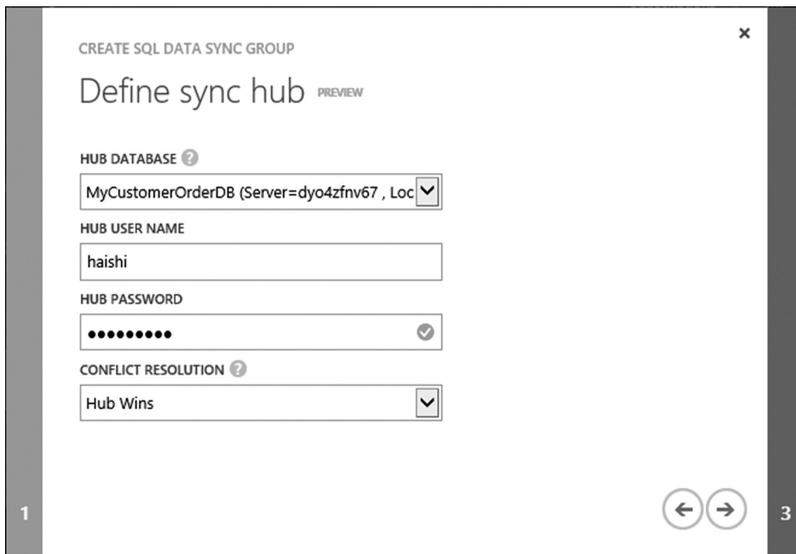


Figure 5.47 Define sync hub data.

```
INSERT Customer(Name, [Address]) VALUES('Sync Customer', '123 Sync Street');
```

17. After several minutes, query the SQL Database to check whether the Customer table on the SQL Database has been updated as well. In addition, you can also check the sync log, to trigger on-demand syncs, on Microsoft Azure Management Portal.

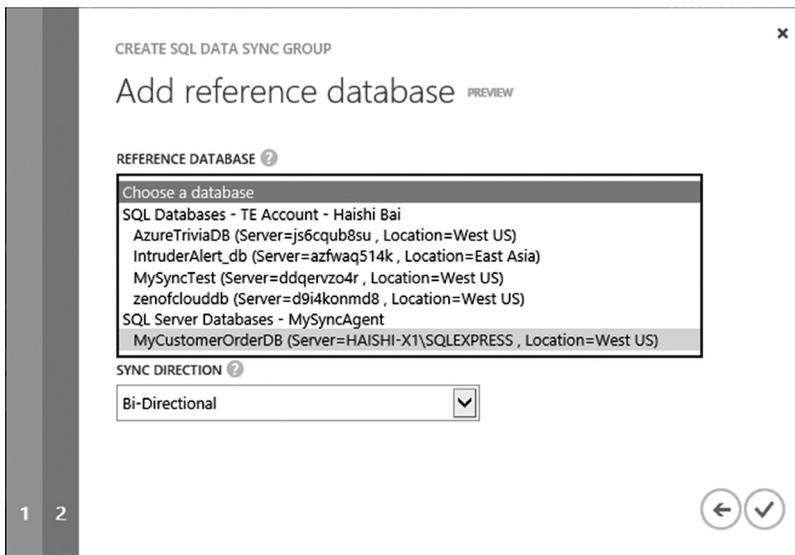


Figure 5.48 Add a reference database.

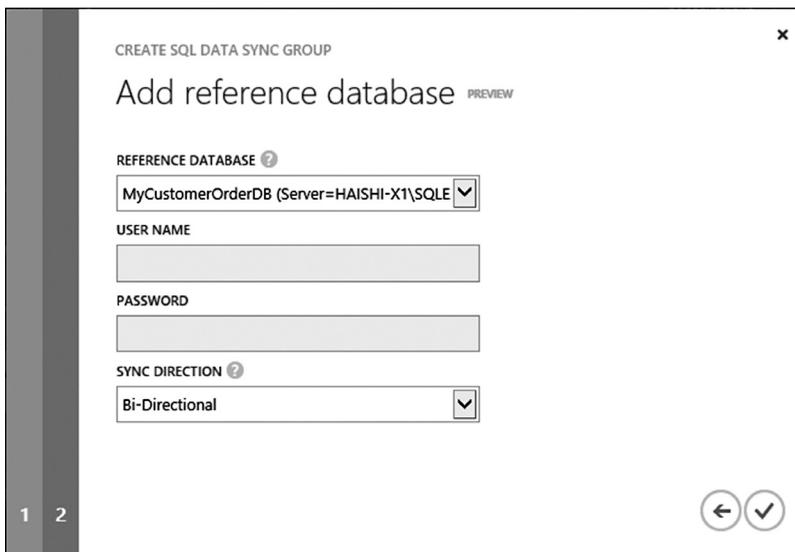


Figure 5.49 Finish adding the reference database.

5.5 Periodically Backup Your SQL Databases

On Microsoft Azure Management Portal, you can set up auto-export rules to periodically export your SQL Database as backups. The exported .bacpac files can be saved on your storage account for a certain number of days. You can restore data from these files when needed, as shown in Figure 5.52.



Figure 5.50 Select database.

mysyncgroup

REFERENCES CONFIGURE SYNC RULES PROPERTIES LOGS

general

AUTOMATIC SYNC

ON OFF

SYNC FREQUENCY

5 Minutes



Figure 5.51 Sync Group configuration.

mycustomerorderdb

 DASHBOARD  MONITOR  SCALE  CONFIGURE

automated export

EXPORT STATUS NONE **AUTOMATIC** PREVIEW ?

STORAGE ACCOUNT ▼

FREQUENCY

Every Days ?

Start date UTC ?

RETENTION

Keep for Days ?

Always keep at least one export. ?

Figure 5.52 Automatic export.

5.6 Use MySQL Database

5.6.1 Microsoft Azure Store

As an open platform, Microsoft Azure provides a Microsoft Azure Store, where service providers and data providers can sell their SaaS services and data. Note that the available services and data may vary in different regions. The process to purchase a new service is very simple:

1. Log in to Microsoft Azure Management Portal.
2. On the command bar, click the **NEW** icon, and then select **STORE**.
3. On **PURCHASE FROM STORE** dialog, you can pick the services or data you want to purchase, and follow the wizard to complete the purchase process. Many service providers provide free trials for you to try out the services before making final purchase decisions. For example, Figure 5.53 shows the ClearDB MySQL service under the **APP SERVICES** category.
4. After a service has been purchased, you can manage it on Microsoft Azure Management Portal under the **ADD-ONS** group. For example, you can click on each service to check its details. You can also use the **MANAGE** icon to open its management UI. Furthermore, you can use the **CONNECTION INFO** icon to check how to connect to the service. You can also use the **UPGRADE** icon to upgrade your service (see Figure 5.54).

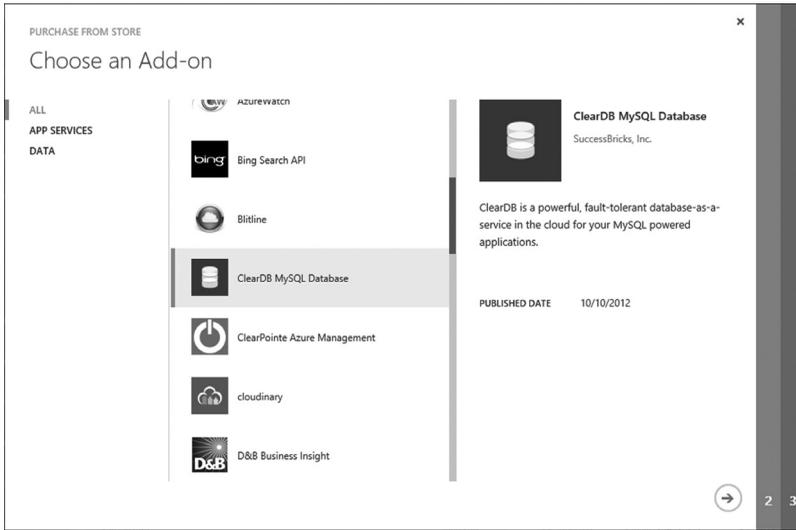


Figure 5.53 Use Microsoft Azure Store.

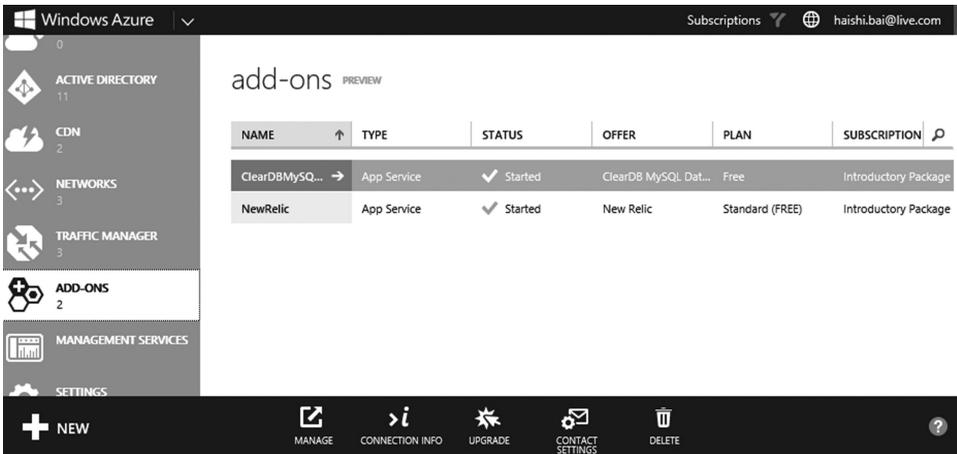


Figure 5.54 Managing add-ons on Microsoft Azure Management Portal.

5.6.2 Purchasing MySQL Service

You can purchase the ClearDB MySQL database following the method in Section 5.6.1. ClearDB MySQL provides several subscription plans, including a free 20 MB, four-connections plan. After you have purchased the service, you can use the **CONNECTION INFO** button on the command bar to view its connection information, as shown in Figure 5.55.

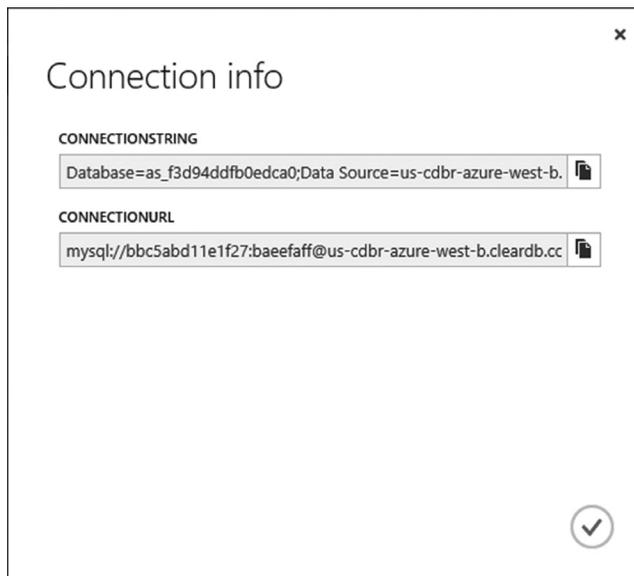


Figure 5.55 Connection information of MySQL.

5.6.3 Other Means to Run MySQL

If the purchased MySQL service does not satisfy the specific requirements of your project, you can build a MySQL environment yourself. Of course, in this case, you are giving up the benefits of SaaS in favor of deep customization. There are several possible options:

- Build MySQL servers on Microsoft Azure Virtual Machines.
- Package MySQL in a cloud service Worker Role, and launch MySQL via startup tasks or custom entry points. Note that because virtual machines running the Worker Roles are stateless, you will need to use data disks external to the cloud service.
- Use a Virtual Network to connect to local MySQL servers.

The detailed steps of the previous configuration exceed the scope of this book. We will introduce Virtual Machines and Virtual Networks in later chapters.

5.7 Summary

In this chapter, we first gave a brief overview of various storage solutions that Microsoft Azure provides. Then we studied the SQL Database. Because of the high compatibility between an SQL Database and an SQL Server, many existing tools, techniques, and codes can be directly applied to the SQL Database. We learned how to use Entity Framework, ASP.NET MVC 4, SQL Server Management Studio, Microsoft SQL Server Data Tools, T-SQL scripts, Dynamic Management

Views, and SQL Database management UI to manage and optimize an SQL Database. We discussed a recommended process of query optimization. Then, we learned how .bacpac and .dacpac can be used to provide database abstractions, backups, and data migrations. We also learned how to use Data Sync Service to sync data among multiple data sources. Finally, we provided a brief introduction of Microsoft Azure Store and introduced several options of running a MySQL service.

Chapter 6

Data Storage *Storage Services*

In addition to relational databases, Microsoft Azure provides several options for saving unstructured data, such as Table storage, BLOB storage, and virtual disks. Before we go into detail of these services, let us first study how to use local storage space on virtual machines running your service roles.

6.1 Local Storage

When defining a cloud service role, you can allocate a local storage space for the role to use. A Local Storage is a special folder on the virtual machine that the role instance can use to save temporary data. Because the virtual machines hosting role instances are stateless, you are not supposed to persist with permanent data on a Local Storage. Although you can specify that data in the Local Storage should be preserved when the role instance is recycled, it is not guaranteed that the data will be permanently saved. For example, during role instance maintenance, Microsoft Azure may migrate your role instances to other virtual machines when there are errors on the hosting machines. Obviously, you will not be able to access the data on the original hosting machines in this case. If you want reliable data storages, you will need to use the database services we introduced in the previous chapter or use the data storage services that we will cover later in this chapter.

You can define one or more storage spaces in the cloud service definition file (.csdef). The minimum size of a Local Storage is 1 MB, and the maximum size of a Local Storage depends on the size of the virtual machine (see the Temporary Storage column in Table 1.1). Now, let us learn how to use a Local Storage via an example.

Example 6.1: Local Storage—Random Data Generator Service

Difficulty: **

In this example, we create a simple data file generation service that generates random data files based on parameters users specify via a Web UI. In this example, we use ASP.NET MVC Web API. If you are not familiar with ASP.NET MVC, you should get some basic knowledge by consulting related documents before returning to this example.

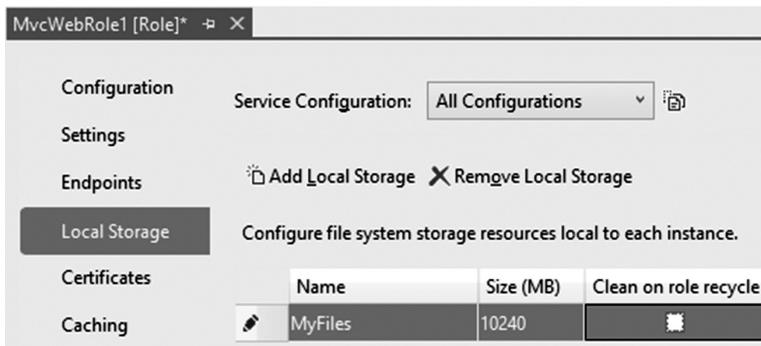


Figure 6.1 Define Local Storage on a role.

1. Launch Visual Studio as an administrator. Create a new cloud service with an ASP.NET MVC Web Role (using the Internet Application template).
2. In **Solution Explorer**, double click the Web Role in the cloud service to open its Properties page.
3. On the left of the screen, click the **Local Storage** tab.
4. Click the **Add Local Storage** link.
5. In the newly added row, enter *MyFiles* as **Name**, and *10240* as **Size**. Then, click Ctrl + S to save the definition file (Figure 6.1).

Note: If you check the *Clean on role recycle* checkbox, the data in the Local Storage will be cleared when the instance is recycled.

6. In the Web Role project, right-click the **Controller** folder, and select the **Add→Controller** menu.
7. On the **Add Controller** dialog, set **Controller name** as *FileController*, and select the **Empty API controller** template. Then, click the **Add** button to add the controller, as shown in Figure 6.2.
8. Implement a GET method on the FileController. The method takes a minimum value (the *min* parameter), a maximum value (the *max* parameter), the number of records (the *count* parameter), and a file name (the *file* parameter), and generates a CSV file containing random numbers of a given number of records within a specified range. Note that this method does not regenerate files with the same name. Line 10 in Code List 6.1 shows how to access the folder of the Local Storage.
9. Modify the **Home\Index.cshtml** file under the **Views** folder. The modified code is as shown in Code List 6.2.
10. Press F5 to launch the application. Click the **Generate Data File** button to download the data file, as shown in Figure 6.3.

6.2 Overview of Microsoft Azure Storage Services

Microsoft Azure Storage services is a series of data storage SaaS provided by Microsoft Azure. You can use either client libraries or their REST-styled APIs to call these services from any platform (such as Windows, Linux, and Mac) using most programming languages (such as C#, Python, and Java) to save and retrieve unstructured data, such as texts, pictures, and videos. We will introduce each of these services in the following sections.

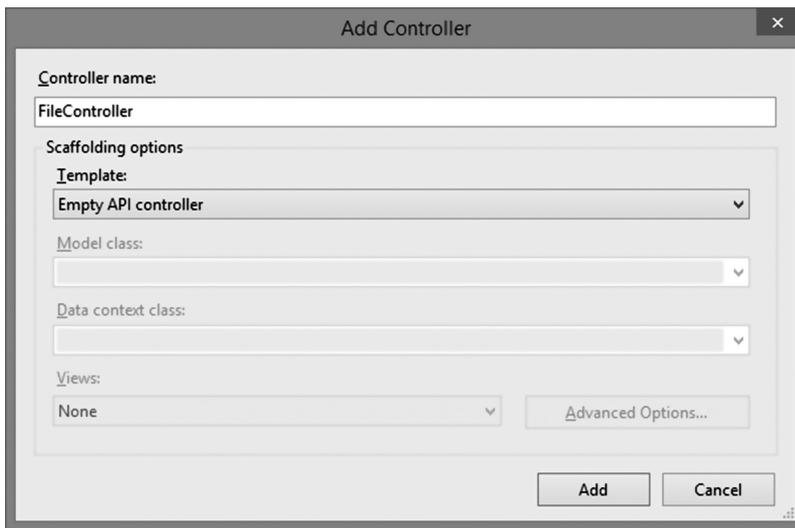


Figure 6.2 Add a new Web API controller.

6.2.1 Microsoft Azure Storage Account

To use Microsoft Azure Storage services, you need to get a storage account. You can create up to five storage accounts under a Microsoft Azure subscription. Each account can be used to save up to 100 TB of data. Before we create an account, let us go over some related concepts:

- **Region, Subregion, and Datacenter**

Microsoft Azure operates datacenters around the globe. These datacenters are divided into several Regions, such as Asia, Europe, and the United States. Each Region is further divided into Subregions. For example, the US region contains four subregions: West, East, North Central, and South Central. Inbound data to Microsoft Azure is free, and data transmitted with the same datacenter is free as well. However, data flowing out from datacenters are billable. You should try to keep your deployed resources together in the same datacenter to avoid unnecessary charges for their communications.

Tip: The first 5 GB of egress data of the billing month is free.

- **Geo Redundant Storage—GRS**

Geo Redundant Storage uses a geo-replication mechanism to replicate user data to another datacenter that is hundreds of miles away from the original datacenter. With GRS, even if a disaster wipes out the whole datacenter, a user's data are still safe in the backup datacenter. GRS is the default option when you provision a new storage account, but you can turn it off to save storage and transmission costs. Geo-redundancy occurs within the same region. For example, user data saved in Europe Region will never be replicated to any datacenters outside the region.

CODE LIST 6.1 GENERATE RANDOM FILES

```

1: using Microsoft.WindowsAzure.ServiceRuntime;
2: using System.IO;
3: using System.Net.Http.Headers;
...
4: public class FileController : ApiController
5: {
6:     private static Random mRandom = new Random(); //Random
       number generator
7:     public HttpResponseMessage Get(int min, int max, long count,
       string file)
8:     {
9:         //get the root folder of a Local Storage
10:        var root = RoleEnvironment.GetLocalResource("MyFiles").
       RootPath;
11:        var fileName = Path.Combine(root, file);
12:        if (!File.Exists(fileName))
13:        {
14:            using (StreamWriter writer =
15:                    new StreamWriter(File.
       Create(fileName, 1024000)))
16:            {
17:                for (var i = 0; i < count; i++)
18:                {
19:                    writer.Write(mRandom.Next(min, max));
20:                    writer.Write(",");
21:                }
22:            }
23:        }
24:        //Return the generated data file
25:        HttpResponseMessage result = new HttpResponseMessage(Http
       StatusCode.OK);
26:        result.Content = new StreamContent(File.Open(fileName,
       FileMode.Open));
27:        result.Content.Headers.ContentDisposition =
28:            new ContentDispositionHeaderValue("attachment");
29:        result.Content.Headers.ContentDisposition.FileName = file;
30:        result.Content.Headers.ContentType =
31:            new MediaTypeHeaderValue("text/plain");
32:        return result;
33:    }
34: }

```

- **Locally Redundant Storage—LRS**

Local Redundancy refers to the fact that all data are saved as three copies within the Subregion to ensure data availability. This is the default behavior and cannot be turned off.

- **Affinity Groups**

Microsoft Azure datacenters host both Microsoft Azure SaaS services (such as storage services) and cloud services deployed by service developers. When Microsoft Azure deploys

CODE LIST 6.2 WEB UI TO CONTROL RANDOM DATA GENERATION

```

@{
    ViewBag.Title = "Home Page";
}
<label>Min Value: </label><input type="text" id="minValue" value="0"
/><br />
<label>Max Value: </label>
<input type="text" id="maxValue" value="@int.MaxValue" />
<br />
<label># of Records: </label><input type="text" id="numOfRec"
value="1000" />
<label>File Name: </label><input type="text" id="fileName"
value="myfile.txt" />
<br />
<input type="button" id="generate" value="Generate Data File" />
@section Scripts{
    <script>
        $('#generate').click(function () {
            window.location.href = '/api/File?min='
                + encodeURIComponent(minValue.value)
                + '&max=' + encodeURIComponent(maxValue.value)
                + '&count=' + encodeURIComponent(numOfRec.value)
                + '&file=' + encodeURIComponent(fileName.value);
        });
    </script>
}

```

your services, it will put your services as close as possible to the SaaS services within the same Affinity Group, in order to minimize the network delays between them.

■ Storage Account Endpoints

Every storage account has three HTTP-based endpoints:

- BLOB service: `http://[storage account].blob.core.windows.net`
- Table service: `http://[storage account].table.core.windows.net`
- Queue service: `http://[storage account].queue.core.windows.net`

Appending the object identifier to these URL generates the URL to an object within the storage service. For example, a BLOB object with the name *blob1* saved in a container with the name *container1* has the following URL:

`http://[storage account].blob.core.windows.net/container1/blob1`

6.2.2 Provisioning a Windows Storage Account

Provisioning a storage account is very simple. In fact, when you deploy a cloud service, if you have not created any storage accounts, the deployment wizard will automatically provision one for you to hold uploaded service packages. To create a new storage account on Microsoft Azure Management Portal, click the **NEW** icon on the command bar, and select **DATA SERVICES**→**STORAGE**→**QUICK CREATE**. Enter a name for the account in the **URL** field,

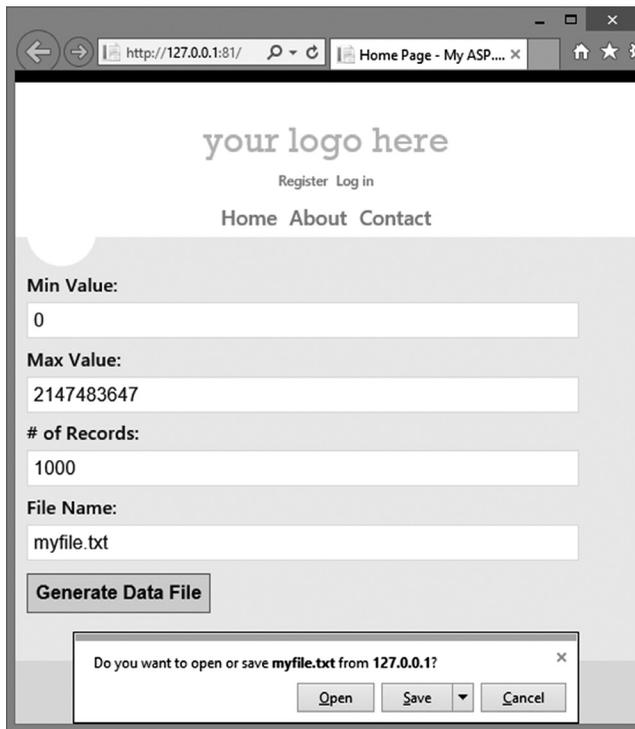


Figure 6.3 Generate random data file.

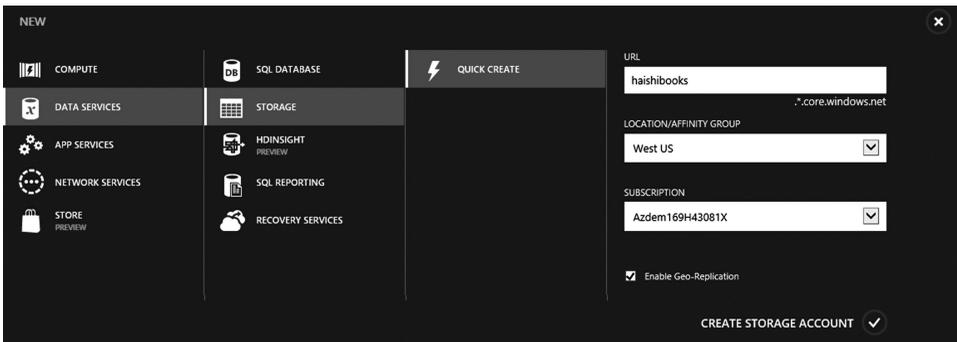


Figure 6.4 Create a new storage account.

and pick a location for the account. Optionally, you can turn off GRS by unchecking the **Enable Geo-Replication** checkbox. Finally, click the **CREATE STORAGE ACCOUNT** link to create the account, as shown in Figure 6.4.

6.2.3 Storage Account Access Keys

Before your service can access data stored on a storage account, the service needs to authenticate to the storage service by providing the correct combination of a storage account name and its

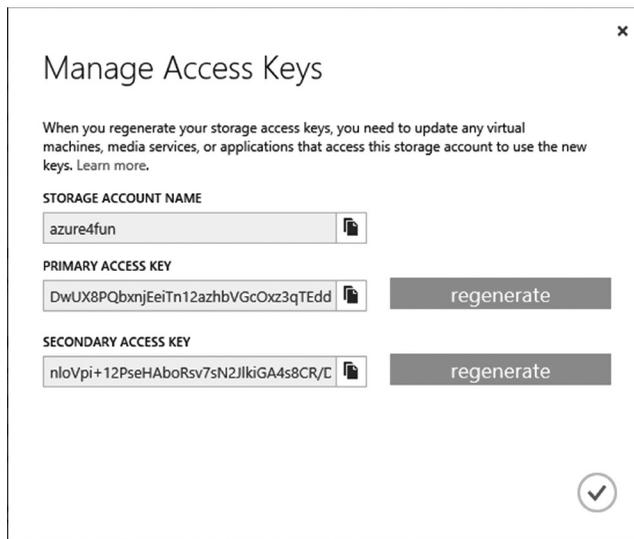


Figure 6.5 Manage access keys of a storage account.

access key. You can query access keys on the storage account list view or the account details view on Microsoft Azure Management Portal. On either view, click the **MANAGE KEYS** icon on the command bar to check or regenerate access keys, as shown in Figure 6.5.

On this dialog, you can check access keys, copy access keys to the clipboard, or regenerate new keys. You may have noticed that there are two access keys, one primary key and one secondary key. What is the purpose of having two keys? This is because when one of the keys is leaked or needs to be updated, you can update your service configuration file to use the other key to keep the service running. Of course, the exact policy of how these two keys are used and updated is up to you. However, when you update access keys, you have to watch out for possible impacts on other services that rely on storage services:

- **Virtual machines**
The disk images of a virtual machine are saved on a storage account. If you regenerate access keys while these virtual machines are running, you will have to redeploy these virtual machines. To avoid redeployments, you should shut down the virtual machines that use the storage account before you update the account's access keys.
- **Media service**
Media service uses storage services to save media files. After you update the access key, you need to sync the key to media service.
- **Cloud services**
Cloud services that use the affected storage accounts need to be updated to use the new keys to reestablish connections. Commonly, connection information to a storage account is presented as a connection string in the service's configuration file:

```
DefaultEndpointsProtocol=https;AccountName=[account
name];AccountKey=[access key]
```

6.3 Using BLOB Storage

BLOB Storage can be used to save large amounts of unstructured data. The size of a single BLOB can be as big as hundreds of GB, and each storage account can hold up to 100 TB of data. Every BLOB has a corresponding URL, which can be accessed by either HTTP or HTTPS.

6.3.1 BLOB Storage Overview

The data structure of the BLOB Storage service is depicted in Figure 6.6.

- Account
 - All access to BLOBs is done using a storage account. Each account contains multiple containers.
- Container
 - Containers are used to group BLOBs. Each container holds any number of BLOBs.
- BLOB
 - BLOB is a file of any size and type. When saved, a BLOB has to be put in a Container. The URL to a BLOB has the following format:

```
http://[storage account].blob.core.windows.net/[container name]/
[BLOB name]
```

There are two types of BLOBs: blocks and pages. Most BLOBs belong to the block type. A block BLOB can be as big as 200 GB. Another type of BLOB is page BLOB. A page BLOB can be as big as 1 TB. We will explain the difference between the two in Section 6.3.2.

Before we learn how to programmatically access the BLOB service, let us learn the basics of managing the BLOB service in Visual Studio.

Example 6.2: Use Visual Studio to manage BLOB Service

Difficulty: *

1. Launch Visual Studio. Select the **VIEW**→**Server Explorer** menu to open Server Explorer.

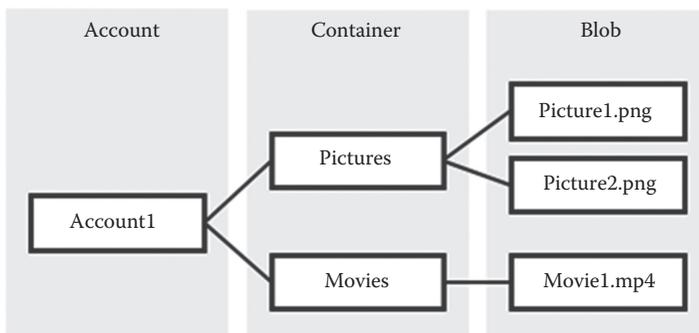


Figure 6.6 BLOB Storage structure.

2. In Server Explorer, expand the storage account node, right click the **Blobs** node, and select the **Create Blob Container** menu, as shown in Figure 6.7.
3. On **Create Blob Container** dialog, enter *pictures* as the container name, and click the **OK** button to create the container (Figure 6.8).
4. In Server Explorer, double click on the new *pictures* node to browse its contents.
5. Click the upload icon (see Figure 6.9), and pick a file to be uploaded to the container, as shown in Figure 6.9.
6. Once the file is uploaded, you can double click the file in Visual Studio to download it, as shown in Figure 6.10.
7. By default, a newly created container is private, which is inaccessible by the public. You can right-click the container in **Server Explorer**, select the **Properties** menu, and then change its **Public Read Access** property to **Blob** to grant anonymous access to the container and its BLOBs.

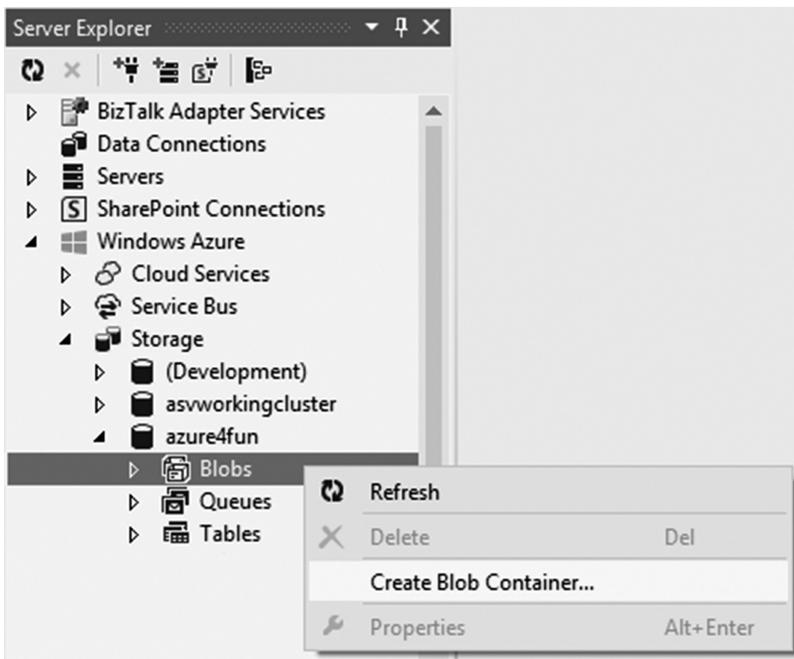


Figure 6.7 Create a Blob Container.

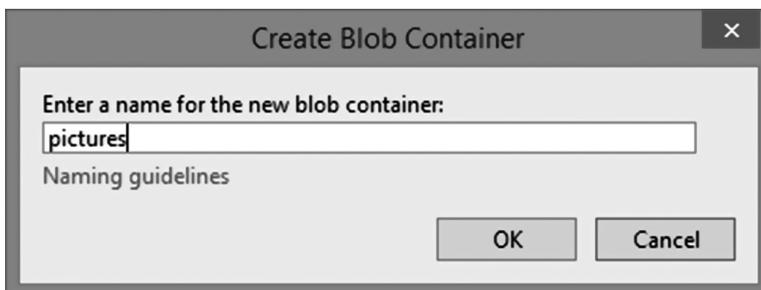


Figure 6.8 Create Blob Container dialog.

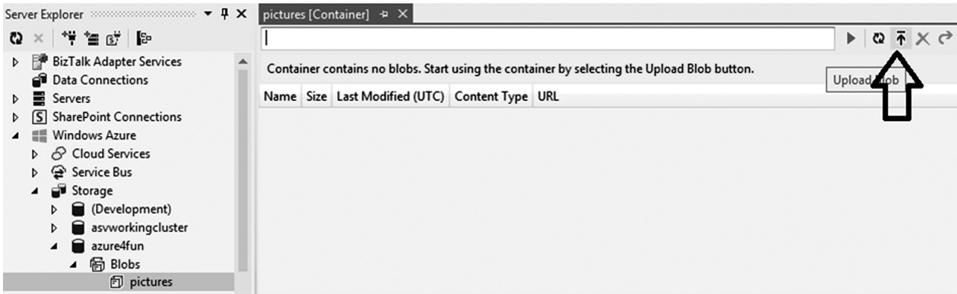


Figure 6.9 Upload a BLOB in Visual Studio.

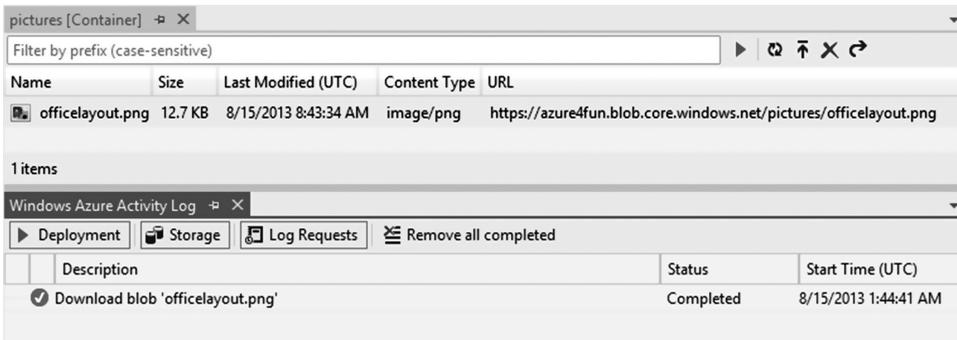


Figure 6.10 Download a BLOB in Visual Studio.

- After the change takes effect, you can copy the URL to the BLOB in Visual Studio (right-click the row in Figure 6.10 and select **Copy URL**), and paste it to a browser to download the file.

Now, let us learn how to use the BLOB service programmatically.

Example 6.3: BLOB Service—online album

Difficulty: ****

In this example, we will create an online album website, which allows users to create albums and upload pictures.

Note: In this example, we will build a relatively complete ASP.NET MVC 4 application. So in addition to accessing the BLOB service, the application also contains other aspects. ASP.NET MVC is still relatively new. Although it is based on the ASP.NET framework, its design philosophy and methodology are quite different from ASP.NET. Although I cannot cover ASP.NET MVC in great detail in this book, I hope to cover some of the basic concepts with this example.

- Launch Visual Studio as an administrator. Create a new cloud service with an ASP.NET MVC 4 Web Role (using Internet Application template).

CODE LIST 6.3 BUSINESS MODELS

```

public class Album
{
    [DisplayName("Name")]
    public string Name { get; set; }
    [DisplayName("Album URL")]
    public string AlbumURL { get; set; }
}
public class AlbumPictures
{
    public string Album { get; set; }
    public List<Picture> Pictures { get; set; }
    public AlbumPictures()
    {
        Pictures = new List<Picture>();
    }
}
public class Picture
{
    public string Album { get; set; }
    [DisplayName("Name")]
    public string Name { get; set; }
    [DisplayName("Picture URL")]
    public string URL { get; set; }
}

```

2. In the Web Role project, right-click the Models folder, and select the **Add→Class** menu. Add an *AlbumModels.cs* file, where we will define three classes: *Album*, *Picture*, and the mapping between them (*AlbumPicture*). These three classes represent the three business models (the “M” in MVC) we want to use in our application. Note that in Code List 6.3, we have decorated several properties with the *DisplayName* attribute, which assists the ASP.NET MVC view scaffolding tool to generate labels for these fields while creating display pages.
3. Rebuild the solution.
4. Add an **Album** folder under the **Views** folder. Under this folder, we will create four views: album list view (*Index.cshtml*), album details view (*Details.cshtml*), page for creating a new album (*Create.cshtml*), and page for uploading a picture (*AddPicture.cshtml*).
5. First, let us work on the album list view (*Index.cshtml*). This page is bound to a list of the album model, as declared in the first line of Code List 6.4. On the top of the page, there is a link to create a new album, under which there is a list of existing albums. Each album corresponds to a container in the BLOB service. The completed code of the view is shown in Code List 6.4.
6. The album details view (*Details.cshtml*) is not complex either, as shown in Code List 6.5.
7. The source code of a page for creating a new album (*Create.cshtml*) is shown in Code List 6.6.
8. Finally, there is the page for uploading a new picture (*AddPicture.cshtml*). The complete source code is shown in Code List 6.7.
9. Add a new Repositories folder, and create an *AlbumRepository* class under the folder. This class encapsulates all BLOB accesses we are going to perform in this application. But we will start with some dummy functions, as shown in Code List 6.8.

CODE LIST 6.4 ALBUM LIST VIEW

```

@model IEnumerable<MvcWebRole1.Models.Album>
@{
    ViewBag.Title = "My Albums";
}
<h2>My Albums</h2>
<p>
    @Html.ActionLink("New Album", "Create")
</p>
@foreach (var item in Model)
{
    <div class="album-div">
        <table>
            <tr>
                <td class="album-title">
                    @Html.DisplayFor(modelItem => item.Name)
                </td>
            </tr>
            <tr>
                <td>
                    
                </td>
            </tr>
            <tr>
                <td class="album-action">
                    @Html.ActionLink("Open", "Details",
                        new { Album = item.Name } ) |
                    @Html.ActionLink("Delete", "Delete",
                        new { Album = item.Name } )
                </td>
            </tr>
        </table>
    </div>
}

```

10. Add a new controller under the Controllers folder. Note that to save a couple of steps, we actually combined an MVC controller and an API controller. As an exercise, you should split the controller into two, with clearly separate responsibilities. The MVC controller is responsible for serving up views only. The API controller provides an access interface to the underlying business logic, which is encapsulated in a repository in this case. The source code of the completed controller is shown in Code List 6.9.
11. Now let us add server CSS style to make the UI a little prettier. Modify the **Site.Css** file under the **Content** folder to add the styles in Code List 6.10.
12. Modify the **RouteConfig.cs** file under the **App_Start** folder to change the default controller from *Home* to *Album*:

```

... defaults: new {controller = "Album" ...

```

CODE LIST 6.5 ALBUM DETAILS VIEW

```

@model MvcWebRole1.Models.AlbumPictures
@{
    ViewBag.Title = "Album Contents";
}
<h2>Album Contents</h2>
<p>
    @Html.ActionLink("Add a Picture", "AddPicture", new { Album =
        Model.Album })
</p>
@foreach (var item in Model.Pictures)
{
    <div class="album-div">
        <table class="album-text">
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.Name)
                </td>
            </tr>
            <tr>
                <td>
                    
                </td>
            </tr>
            <tr>
                <td>
                    @Html.ActionLink("Delete", "DeletePicture",
                        new { Album = item.Album, Name = item.
                            Name })
                </td>
            </tr>
        </table>
    </div>
}
<div>
    @Html.ActionLink("Return to Album List", "Index")
</div>

```

13. Now you can delete the *Index.cshtml* file from the **Views\Home** folder—we do not need it anymore.
14. Rebuild the solution. Now our application is almost ready except for the *AlbumRepository* class, which we will focus on next.
15. First, let us add a *StorageConnectionString* setting to the Web Role, and point it to your storage account, as shown in Figure 6.11.
16. When you created the project, Microsoft Azure ADK already added to your project a reference to *Microsoft.WindowsAzure.Storage*, which contains the client library you need to access for the BLOB storage. First, let us import several namespaces and declare two temporary variables: *mAccount* representing a storage account, and *mClient* representing a BLOB service client.

```
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.Storage.Blob;
...
CloudStorageAccount mAccount;
CloudBlobClient mClient;
```

17. Initialize *mAccount* and *mClient* in the *AlbumRepository*'s constructor.

```
public AlbumRepository()
{
    mAccount = CloudStorageAccount.
        Parse(CloudConfigurationManager.GetSetting("StorageConnection
            String"));
    mClient = mAccount.CreateCloudBlobClient();
}
```

CODE LIST 6.6 PAGE FOR CREATING A NEW ALBUM

```
@model MvcWebRole1.Models.Album
@{
    ViewBag.Title = "Create a New Album";
}
<h2>Create a New Album</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Album</legend>
        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>
        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}
<div>
    @Html.ActionLink("Return to Album List", "Index")
</div>
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

CODE LIST 6.7 PAGE FOR UPLOADING A NEW PICTURE

```

@model MvcWebRole1.Models.Picture
@{
    ViewBag.Title = "Add a New Picture";
}
<h2>Add a New Picture</h2>
@using (Html.BeginForm("AddPicture", "Album", FormMethod.Post,
    new { enctype = "multipart/form-data" }))
{
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Picture</legend>
        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>
        <div class="editor-label">
            @Html.LabelFor(model => model.URL)
        </div>
        <div class="editor-field">
            <input type="file" name="picture" id="picture" />
        </div>
        @Html.HiddenFor(model => model.Album)
        <p>
            <input type="submit" value="Add" />
        </p>
    </fieldset>
}
<div>
    @Html.ActionLink("Return to Album", "Details", new { Album =
        Model.Album })
</div>
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}

```

18. Because this application directly uses containers as albums, the method to list albums is simply to list all containers under the storage account, as shown in line 4 of Code List 6.11.
19. To list all pictures in an album, we enumerate all BLOB files in the container. This is a two-step operation: first to get a reference to a container (line 4 in Code List 6.12) and then to enumerate the BLOBs (line 5).

Note: ListBlobs takes two parameters. The first parameter is a prefix and the second parameter is a flag indicating whether the returned entities should contain hierarchies.

Internally, the BLOBs in a container are organized in a flat structure, but you can use BLOB names to simulate folder hierarchies. For example, two BLOBs with the names *folder1/blob1* and *folder1/blob2* can be considered as two BLOBs under the same (virtual) *folder1* folder. When we set the second parameter to *true*, we will get a simple *CloudBlob* list back. If we set the parameter to *false*, on the other hand, we will get a mixed list of *CloudBlob* and *CloudBlobDirectory*. Then we can use the *CloudBlobDirectory* instance to get the BLOBs it contains.

20. To create a new album is to create a new container. By default, the containers you create (see line 4 in Code List 6.13) are private, which are inaccessible by the public. In lines 5–8, we change the access type of the newly created container to *BlobContainerPublicAccessType.Blob*, which allows anonymous access to BLOBs as well as their metadata, but does not allow access to the container's metadata, or to enumerate BLOBs. The other two access types are *Container*, which grants access to BLOBs and metadata and to enumerate BLOBs) and *Off*, which turns off anonymous access. We will make use of metadata later in this example.

CODE LIST 6.8 DUMMY STORAGE FUNCTIONS

```
public class AlbumRepository
{
    public List<Album> ListAlbums()
    {
        throw new NotImplementedException();
    }
    public AlbumPictures ListPictures(string name)
    {
        throw new NotImplementedException();
    }
    public void CreateAlbum(string name)
    {
        throw new NotImplementedException();
    }
    public void DeleteAlbum(string name)
    {
        throw new NotImplementedException();
    }
    public void DeletePicture(string album, string name)
    {
        throw new NotImplementedException();
    }
    public void AddPicture(string album, string name, Stream file)
    {
        throw new NotImplementedException();
    }
}
```

CODE LIST 6.9 ALBUM CONTROLLER

```

public class AlbumController : Controller
{
    AlbumRepository repository = new AlbumRepository();
    public ActionResult Index()
    {
        return View(repository.ListAlbums());
    }
    public ActionResult Details(string album)
    {
        return View(repository.ListPictures(album));
    }
    public ActionResult Create()
    {
        return View();
    }
    public ActionResult AddPicture(string Album)
    {
        return View(new Picture { Album = Album });
    }
    [HttpPost]
    public ActionResult AddPicture(FormCollection collection)
    {
        repository.AddPicture(collection["Album"],
            collection["Name"],
            Request.Files[0].InputStream);
        return RedirectToAction("Details",
            new { Album = collection["Album"] });
    }
    [HttpPost]
    public ActionResult Create(FormCollection collection)
    {
        repository.CreateAlbum(collection["Name"]);
        return RedirectToAction("Index");
    }
    public ActionResult Delete(string album)
    {
        repository.DeleteAlbum(album);
        return RedirectToAction("Index");
    }
    public ActionResult DeletePicture(string album, string name)
    {
        repository.DeletePicture(album, name);
        return RedirectToAction("Details", new { Album = album });
    }
}

```

CODE LIST 6.10 SITE STYLE

```

.album-picture {
    width:250px;
    height:250px;
    padding:5px;
    margin:3px;
    background-color:darkgray;
}
.album-div {
    display:inline-block;
}
.album-title {
    font-family: Arial, Helvetica, sans-serif;
    font-weight:bold;
    font-size: 32px;
}
.album-action {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 16px;
}

```

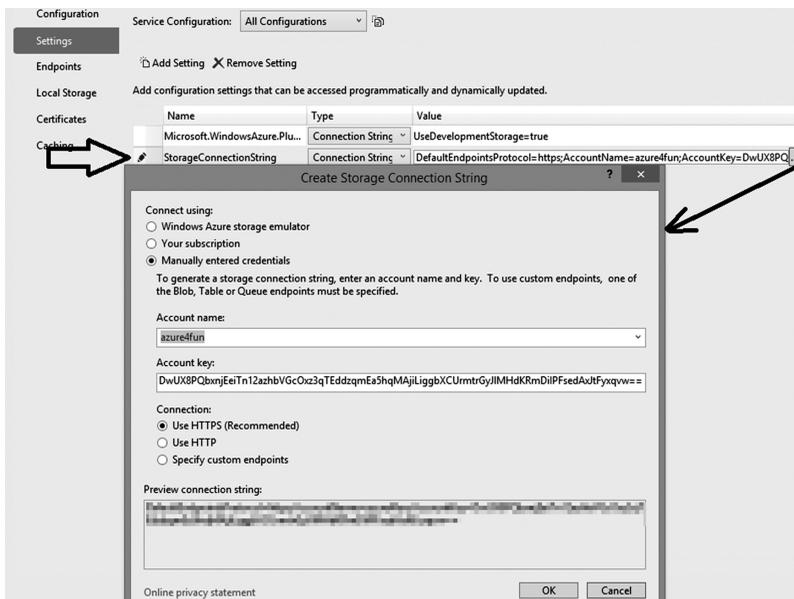


Figure 6.11 Adding connection string to the Web Role.

CODE LIST 6.11 LISTALBUMS METHOD

```

1:public List<Album> ListAlbums()
2:{
3:    List<Album> result = new List<Album>();
4:    var containers = mClient.ListContainers();
5:    foreach (var container in containers)
6:    {
7:        var album = new Album { Name = container.Name };
8:        result.Add(album);
9:    }
10:    return result;
11:}

```

CODE LIST 6.12 LISTPICTURES METHOD

```

1:public AlbumPictures ListPictures(string name)
2:{
3:    AlbumPictures result = new AlbumPictures() { Album = name };
4:    var container = mClient.GetContainerReference(name);
5:    foreach (var blob in container.ListBlobs(null, true))
6:    {
7:        CloudBlockBlob blockBlob = (CloudBlockBlob)blob;
8:        result.Pictures.Add(new Picture
9:            {
10:                Album = name,
11:                Name = blockBlob.Name,
12:                URL = blockBlob.Uri.ToString()
13:            });
14:    }
15:    return result;
16:}

```

CODE LIST 6.13 CREATEALBUM METHOD

```

1:public void CreateAlbum(string name)
2:{
3:    var container = mClient.GetContainerReference(name);
4:    container.CreateIfNotExists();
5:    container.SetPermissions(new BlobContainerPermissions
6:    {
7:        PublicAccess = BlobContainerPublicAccessType.Blob
8:    });
9:}

```

21. The process of adding a picture to an album is the same as uploading a BLOB to a container. Here we are dealing with the simplest case. In the next section of this chapter, we will discuss how to handle different BLOB types and large files.

```
public void AddPicture(string album, string name, Stream file)
{
    var container = mClient.GetContainerReference(album);
    CloudBlockBlob blob = container.GetBlockBlobReference(name);
    blob.UploadFromStream(file);
}
```

Note: We do not use the BLOB downloading method in this example, because we simply point the sources of tags directly to BLOB URLs. If you want to programmatically download a BLOB, you can use the *DownloadToStream* method on a BLOB.

22. Deleting an album (container) or a picture (BLOB) is fairly straightforward, as shown in Code List 6.14. The only thing to point out is the *DeleteSnapshotsOption.IncludeSnapshots* flag in line 10. BLOB storage service allows you to create multiple *snapshots* of a BLOB to preserve multiple versions of it. This flag allows us to delete all associated snapshots while deleting a BLOB.
23. Before we launch the application, let us improve the *ListAlbum* method to display the first image in the container as its cover picture:

```
public List<Album> ListAlbums()
{
    List<Album> result = new List<Album>();
    var containers = mClient.ListContainers();
    foreach (var container in containers)
    {
        var album = new Album { Name = container.Name };
        var blobs = container.ListBlobs(null, false);
        if (blobs != null && blobs.Count() > 0)
            album.AlbumURL = blobs.First().Uri.ToString();
        result.Add(album);
    }
    return result;
}
```

24. If everything is entered correctly, you can press F5 to test out the application (see Figure 6.12). Note that because the names of BLOBs and containers have to be made up by lowercase letters, numbers, and dashes only (and between 3 and 36 letters), you have to pick names following this convention. Of course, this is not very convenient, which we will fix in the next few steps.
25. Both BLOBs and containers have associated *Properties* and *Metadata*. Properties are read-only, but you are free to define custom metadata. We will save descriptive information of albums and pictures in the metadata of corresponding entities.

CODE LIST 6.14 DELETEALBUM METHOD AND DELETEPICTURE METHOD

```
public void DeleteAlbum(string name)
{
    var container = mClient.GetContainerReference(name);
    container.DeleteIfExists();
}
public void DeletePicture(string album, string name)
{
    var container = mClient.GetContainerReference(album);
    var blob = container.GetBlockBlobReference(name);
    blob.DeleteIfExists(DeleteSnapshotsOption.IncludeSnapshots);
}
```

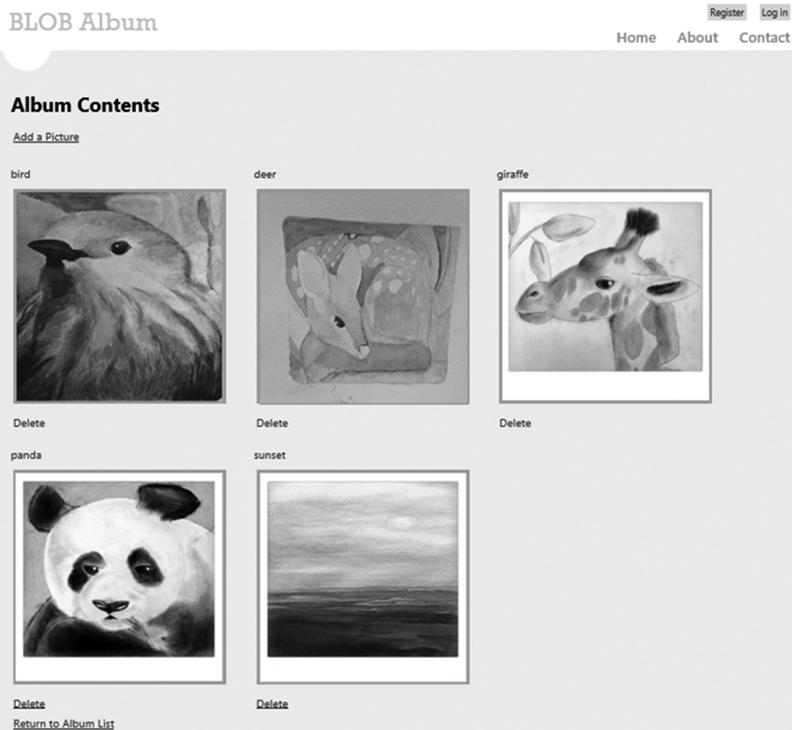


Figure 6.12 Running BLOB Album application.

26. Add a *Description* field to the Album model:

```
[DisplayName("Description")]
public string Description {get; set;}
```

27. Modify the *Index.cshtml* file under the **Views\Album** folder. We will display an album's *Description*, which is not constrained by BLOB naming rules, instead of its *Name*.

```
...
<tr>
    <td class="album-title">
        @Html.DisplayFor(modelItem => item.Description)
    </td>
</tr>
```

28. Modify the *Create.cshtml* page under the **Views\Album** folder to add the new field:

```
<div class="editor-label">
    @Html.LabelFor(model => model.Description)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Description)
    @Html.ValidationMessageFor(model => model.Description)
</div>
```

29. Correspondingly, we need to modify *AlbumController.cs*:

```
[HttpPost]
public ActionResult Create(FormCollection collection)
{
    repository.CreateAlbum(collection["Name"],
        collection["Description"]);
    return RedirectToAction("Index");
}
```

30. We also need to modify the *CreateAlbum* method of the *AlbumRepository* class. Lines 5–7 in Code List 6.15 save the album's description to its metadata. Note that once you finish updating the metadata, you need to invoke *SetMetadata* to commit the changes.
31. Finally, modify the *ListAlbum* method to read album descriptions from metadata (see lines 7–14 in Code List 6.16).
32. The modified album creation page is shown in Figure 6.13.
33. The modified album list view is shown in Figure 6.14.

6.3.2 Block BLOB and Page BLOB

BLOB storage service supports two BLOB types: block BLOB and page BLOB.

■ Block BLOB

When uploading a BLOB, you can upload a file up to 64M in size (the client library by default allows an upload size of up to 32 MB, but you can change this setting by modifying the *SingleBlobUploadThresholdInBytes* attribute). Larger files are split into blocks, each smaller than 4 MB. A BLOB can be split into as many as 50,000 such blocks. After all blocks have been uploaded, you need to upload a block list so that the storage service can

CODE LIST 6.15 MODIFIED CREATEALBUM METHOD

```

1:public void CreateAlbum(string name, string description)
2:{
3:    var container = mClient.GetContainerReference(name);
4:    container.CreateIfNotExists();
5:    container.Metadata["description"] =
6:        Convert.ToBase64String(Encoding.UTF8.
            GetBytes(description));
7:    container.SetMetadata();
8:    container.SetPermissions(new BlobContainerPermissions
9:        {
10:        PublicAccess = BlobContainerPublicAccessType.Container
11:        });
12:}

```

CODE LIST 6.16 MODIFIED LISTALBUMS METHOD

```

1:public List<Album> ListAlbums()
2:{
3:    List<Album> result = new List<Album>();
4:    var containers = mClient.ListContainers();
5:    foreach (var container in containers)
6:    {
7:        container.FetchAttributes();
8:        var album = new Album
9:        {
10:            Name = container.Name,
11:            Description =
12:                (container.Metadata.ContainsKey("description") ?
13:                Encoding.UTF8.GetString
14:                (Convert.FromBase64String(
15:                    container.Metadata["description"]))
16:                : container.Name)
17:        };
18:        var blobs = container.ListBlobs(null, false);
19:        if (blobs != null && blobs.Count() > 0)
20:            album.AlbumURL = blobs.First().Uri.ToString();
21:        result.Add(album);
22:    }
23:    return result;
24:}

```

assemble these blocks back into a complete BLOB. You can concurrently upload multiple blocks, and they do not need to be uploaded in order. All uploaded blocks will be in an *uncommitted* state till you commit the operation. To upload a BLOB in blocks programmatically, you can use the *PutBlock* method on the *CloudBlockBlob* class to submit blocks, and use the *PutBlockList* method to submit the block list.

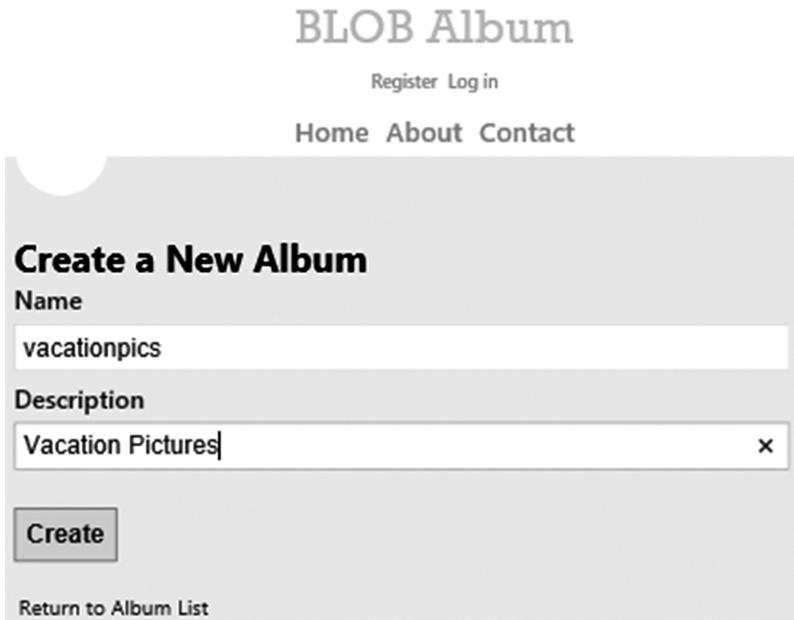


Figure 6.13 Modified album creation page.

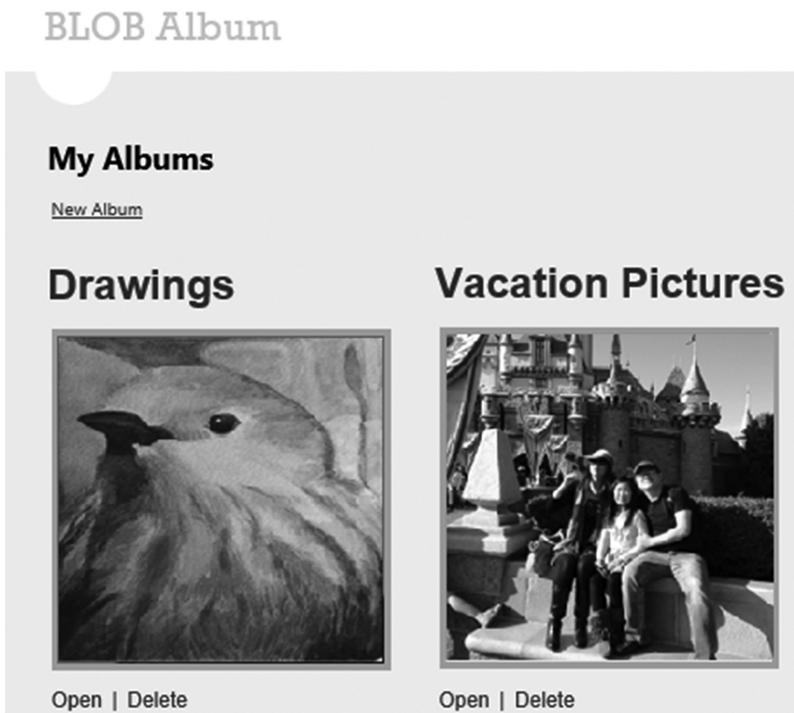


Figure 6.14 Modified album list view.

■ Page BLOB

A Page BLOB is made up of 512B pages for random read/write access. When you create a page blob, you need to specify the maximum size the BLOB can use. All read/write operations are performed within the 512B boundaries. Different from a block BLOB, all your updates are immediately committed without additional commit operations. The maximum size of a page BLOB is 1 TB.

6.3.3 ETag and Snapshots

Entity Tag (ETag) is a common mechanism for optimistic concurrency control. Every committed BLOB has an associated ETag, which can be used to represent the version of the BLOB. For example, when you write back a BLOB, you can compare the BLOB's current ETag and the ETag your BLOB has to detect if the BLOB has been modified before you commit your changes. Using ETag for concurrency does not require holding a lock on the BLOB for a long time. It is a popular way for concurrency control on cloud because it is simple and it provides higher data availability compared to other methods. Code List 6.17 provides a simple list of how to read an ETag.

You can create multiple snapshots for a BLOB. The snapshots are read-only. You can create, delete, and read snapshots, but you cannot update them. A snapshot is a *CloudBlob* instance itself. Once you have create a snapshot for a BLOB, you can use the snapshot's *Snapshot* property, which is a *DateTime*-typed value, to uniquely identify this snapshot. You can use snapshots to save multiple versions of a BLOB. You can also use a snapshot to restore a BLOB to an older version. Code List 6.18 is a simple example of creating a snapshot.

6.3.4 REST API

Microsoft Azure storage services, as well as most of SaaS services provided by Microsoft Azure, provide REST API for the clients to use. For example, you can send an HTTP/1.1 GET request to the address `https://[storage account name].blob.core.windows.net/?comp=list` to enumerate BLOB containers under a storage account. Unsurprisingly, your requests have to contain the required headers. For detailed header requirements, you may consult the MSDN document at the address:

CODE LIST 6.17 READING ETAG

```
CloudBlockBlob blockBlob;
...
blockBlob.FetchAttributes();
var etag = blockBlob.Properties.ETag;
```

CODE LIST 6.18 CREATING A SNAPSHOT

```
CloudBlob blob;
...
CloudBlob snapshot = blob.CreateSnapshot();
DateTime timestamp = (DateTime)snapshot.Attributes.Snapshot;
```



Figure 6.15 Submit a REST API query via Fiddler.

```
<?xml version="1.0" encoding="utf-8"?><EnumerationResults AccountName="https://azure4fun.blob.core.windows.net/"><Containers><Container><Name>cacheclusterconfigs</Name><Uri>https://azure4fun.blob.core.windows.net/cacheclusterconfigs</Uri><Properties><Last-Modified>Tue, 30 Apr 2013 09:09:19 GMT</Last-Modified><Etag>DxB0D1399882FE14</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>code</Name><Uri>https://azure4fun.blob.core.windows.net/code</Uri><Properties><Last-Modified>Mon, 21 Jan 2013 18:29:38 GMT</Last-Modified><Etag>DxB0CF510578E99411</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>denary</Name><Uri>https://azure4fun.blob.core.windows.net/denary</Uri><Properties><Last-Modified>Sun, 28 Apr 2013 09:30:18 GMT</Last-Modified><Etag>DxB0D120A21A20D7D</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>frames</Name><Uri>https://azure4fun.blob.core.windows.net/frames</Uri><Properties><Last-Modified>Mon, 28 Jan 2013 09:08:18 GMT</Last-Modified><Etag>DxB0CFB4E25D2E048</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>learnazure-high-en</Name><Uri>https://azure4fun.blob.core.windows.net/learnazure-high-en</Uri><Properties><Last-Modified>Mon, 10 Sep 2012 07:50:43 GMT</Last-Modified><Etag>DxB0CF5D4010792F48</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>learnazure-low-en</Name><Uri>https://azure4fun.blob.core.windows.net/learnazure-low-en</Uri><Properties><Last-Modified>Mon, 10 Sep 2012 07:05:51 GMT</Last-Modified><Etag>DxB0CF5D39D82E2CAD</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>mam</Name><Uri>https://azure4fun.blob.core.windows.net/mam</Uri><Properties><Last-Modified>Wed, 17 Oct 2012 22:18:21 GMT</Last-Modified><Etag>DxB0CF7AC738507E9</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>salinity</Name><Uri>https://azure4fun.blob.core.windows.net/salinity</Uri><Properties><Last-Modified>Sun, 28 Apr 2013 09:30:07 GMT</Last-Modified><Etag>DxB0D120A1B2BD8ED</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>temperature</Name><Uri>https://azure4fun.blob.core.windows.net/temperature</Uri><Properties><Last-Modified>Sat, 27 Apr 2013 21:57:58 GMT</Last-Modified><Etag>DxB0D11A96A45CFE2</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>videoshare</Name><Uri>https://azure4fun.blob.core.windows.net/videoshare</Uri><Properties><Last-Modified>Mon, 07 Jan 2013 19:33:14 GMT</Last-Modified><Etag>DxB0CFBB24F82A72</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>vshare</Name><Uri>https://azure4fun.blob.core.windows.net/vshare</Uri><Properties><Last-Modified>Wed, 27 Feb 2013 10:16:17 GMT</Last-Modified><Etag>DxB0CFE2E9D51F0A4</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>vsd</Name><Uri>https://azure4fun.blob.core.windows.net/vsd</Uri><Properties><Last-Modified>Wed, 13 Mar 2013 03:34:24 GMT</Last-Modified><Etag>DxB0CFEDB2C9A9BC17</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container><Container><Name>vsd-control</Name><Uri>https://azure4fun.blob.core.windows.net/vsd-control</Uri><Properties><Last-Modified>Fri, 22 Mar 2013 16:01:09 GMT</Last-Modified><Etag>DxB0CF52D9A3AA906</Etag><LeaseStatus>unlocked</LeaseStatus><LeaseState>available</LeaseState></Properties></Container></Containers></EnumerationResults>
```

Figure 6.16 Sample query result.

<http://msdn.microsoft.com/en-us/library/windowsazure/dd135733.aspx>. Figure 6.15 shows a sample query submitted via Fiddler.

This query returns an XML file, which contains all the containers under the storage account, as shown in Figure 6.16.

Note: You will not be able to resubmit the previous query, because the *Authorization* header is calculated with the submission time factored in. You may visit

<http://msdn.microsoft.com/en-us/library/windowsazure/dd135733.aspx> for a complete list of REST API.

6.3.5 Shared Access Signature and Stored Access Policies

When you share your BLOB data with other people, you need to give them access to your resources. Instead of granting anonymous access to the resources, you can create Shared Access Signatures (SAS) to allow users to access the specific resources within a limited period of time. The format of SAS is a URL, which allows users to access the corresponding BLOB resource with specified constraints. For example, if you want to allow a customer to view and sign a document within a given time, you can create an SAS for the document and share the URL with the customer. The customer can use the URL to open the document only within the specified time period; otherwise, the access will be denied. Because using an SAS URL does not require extra authentication, you need to ensure that the URL is shared only with designated users.

Obviously, it is a tedious job to maintain a large number of SASs. Instead, you can use Stored Access Policy to manage SASs in bulk. For example, you can change the access window, or deny access to a group of SASs, using a Stored Access Policy.

Code List 6.19 is a simple example of generating an SAS. Lines 4–8 define an immediately effective access policy (by omitting the *ShareAccessStartTime* parameter) that expires in 5 h.

CODE LIST 6.19 GENERATING AN SAS

```

1 CloudBlobContainer container;
2 ...
3 BlobContainerPermissions permissions = new
  BlobContainerPermissions();
4 permissions.SharedAccessPolicies.Add("testpolicy", new
  SharedAccessBlobPolicy
5 {
6     SharedAccessExpiryTime = DateTime.UtcNow.AddHours(5),
7     Permissions = SharedAccessBlobPermissions.Read
8     | SharedAccessBlobPermissions.List});
9 permissions.PublicAccess = BlobContainerPublicAccessType.Off;
10 container.SetPermissions(permissions);
11 var sasToken = container.GetSharedAccessSignature
12     (permissions.SharedAccessPolicies["testpolicy"]);

```

The policy allows SAS holders to perform *Read* and *List* operations on the container within the time limit. Lines 11 and 12 indicate how to generate the SAS. Then, you can share the result *sasToken* string with your customers.

Note: About UTC Time

All servers and services on Microsoft Azure use UTC time. Moreover, many services check if the timestamp on the request is close enough to the server time when handling a request. When you submit a request, ensure that you are using UTC time. Line 6 in Code List 6.19 shows how to use UTC time. Because there will be a time drift between the client machine and the server, the code does not specify a start time to ensure the policy is effective immediately.

Once the SAS is received, the customer can use it to create a new *StorageCredentials* instance (line 1 in Code List 6.20) to gain access to the resource. As shown in Code List 6.20, because the SAS grants *List* operation, the customer can enumerate the BLOBs in the container (line 4).

CODE LIST 6.20 USE AN SAS

```

1 StorageCredentials credentials = new StorageCredentials(sasToken);
2 CloudBlobContainer clientContainer = new CloudBlobContainer(new
3     Uri("https://[storage account name].blob.core.windows.net/"
4     + name), credentials);
5 var blobs = clientContainer.ListBlobs();
6 int count = blobs.Count();

```

CODE LIST 6.21 USE BLOB LEASE

```

1 var leaseID = blob.AcquireLease(TimeSpan.FromSeconds(10), null);
2 blob.Delete(DeleteSnapshotsOption.IncludeSnapshots,
3     new AccessCondition { LeaseId = leaseID });

```

6.3.6 BLOB Update, Copy, and Lease

In addition to adding and deleting BLOBs, you can update or copy existing BLOBs (within the same storage account or across different storage accounts). Before you operate on a BLOB, you can create a lease, which provides the holder exclusive write and delete access to the BLOB. The lease can be between 15 s and 1 min, or be permanent. Before the lease expires, the lease holder can renew the lease to extend the exclusive access window if needed. Lease state changes are more complex than what we just described. Interested readers may consult <http://msdn.microsoft.com/en-us/library/windowsazure/ee691972.aspx> for more details.

Code List 6.21 is a simple example of using a lease. Line 1 creates a 10-s lease. The operation returns a lease id, which identifies the lease. If a BLOB has a lease on it, you have to provide the correct lease id before you can operate on the BLOB (as shown in line 3).

6.3.7 Error Handling

In order to simplify the code, we skipped all error handling in the previous code. Most user errors can be avoided by proper input validations—for instance, to check if the album name follows the naming convention of a BLOB container. However, you may also notice that some operations may fail without apparent reasons, but succeed if retried later. This phenomenon is most likely caused by transient errors. Transient errors are caused by some temporary conditions, and they will go away once the triggering conditions disappear. We will introduce transient error handling in Chapter 10.

6.4 Using Table Storage

Table storage service is ideal for saving large amounts of unstructured, unrelated data. For example, you can use table storage to save millions of contact cards, billions of geo coordinates, etc.

6.4.1 Table Storage Overview

The data structure of the table storage service is depicted in Figure 6.17.

- **Account**
All access to tables are done using a storage account.
- **Table**
A table is a collection of entities. Different from a relational database, table storage does not have a fixed schema. In other words, you can save entities with different fields in the same table.
- **Entity**
Entity is a collection of properties, similar to a record in a relational database. The maximum size of an entity is 1 MB.

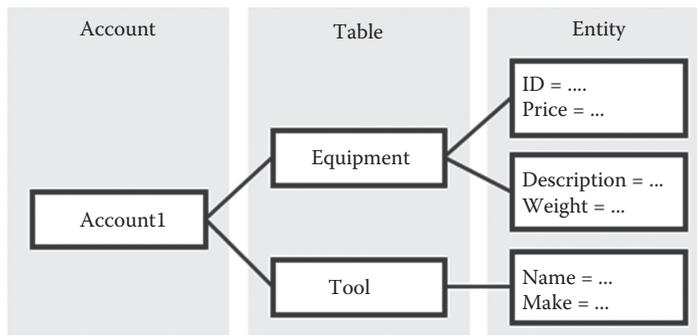


Figure 6.17 Table storage service.

- Property

A property is a key-value pair. Each entity can have up to 252 properties. Each entity has three system-reserved properties: Partition Key, Row Key, and Timestamp. Entities residing in the same partition can be inserted or queried with an atomic operation. A Row Key has to be unique within a partition.

6.4.2 Optimizing Data Partition

Similar to many other NoSQL databases, Microsoft Azure Table storage also saves data as key-value pairs. However, the data keys in Microsoft Azure Table storage consist of two parts: Partition Key and Row Key. The Partition Key and the Row Key are strings with length of less than 1K, or an empty string (but cannot be null). Data keys are sorted alphabetically, first by Partition Keys and then by Row Keys, forming a clustered index. Figure 6.18 shows four entities belonging to two partitions (partitions 11 and 12). In partition 11, because the Row Keys are sorted alphabetically, key 111 appears before key 20. Microsoft Azure may put partitions on different partition servers to balance overall system loads. Each partition server can hold multiple partitions and can return about 500 entities per second (actual throughputs may vary). How the data are partitioned may have significant impacts on system performance. Here we will cover several points that are worth noting:

- Use smaller partitions

All entities within a partition are saved on the same partition server. When a partition is hit heavily, it may overload the hosting partition server. In this case, Microsoft Azure will relocate some of the partitions on this server to some other servers to balance system loads. So, generally speaking, you should use smaller partitions to reduce the probability of hitting the same partition heavily. At the same time, smaller partitions are easier to migrate to other partition servers so your partition can “escape” from a busy server quickly.

- Use batch operations

Within the same partition, you can group operations on up to 100 entities into a batch, which is processed as a single ACID (Atomicity, Consistency, Isolation, and Durability) transaction. Batched operations not only give you better performance, but also save your money because a batch is billed as one transaction.

- Range partitions

If each of the entities in your table has a unique partition id (in other words, if each portion contains only one entity), and these partition keys are sorted in an ascending

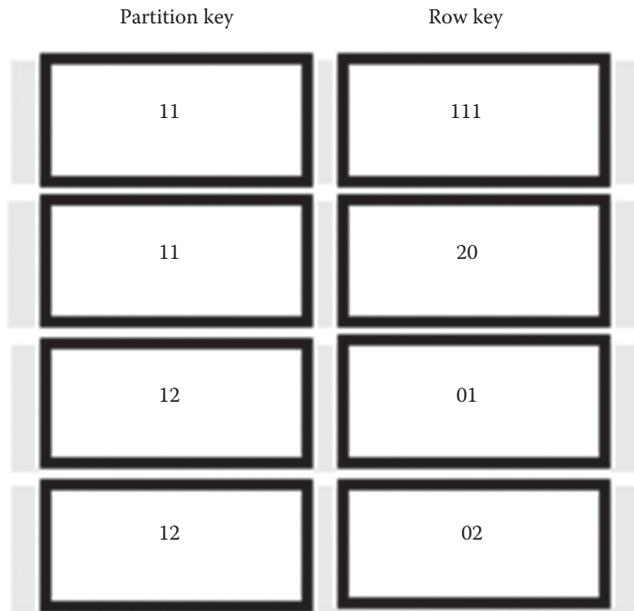


Figure 6.18 Table storage index structure.

or descending order, the table storage service may automatically group these partitions to range partitions to improve query speed. If your insert operations are always to be appended or prepended to this ordered list, all insert operations will always be handled by the tail or the head partition server. On the other hand, if the partition keys of your insert operations are randomly distributed, the workload is better balanced across multiple partition servers.

Of course, there is no fixed formula for choosing the best partition strategy. Different data, different query modes, and different update patterns all have impacts on how partitions should be chosen. In your projects, you should choose partitions based on your specific scenarios and test your choices by performance tests in order to determine the best partition strategy.

Note: Table storage does not support custom indexes.

Using table storage service is very similar to using BLOB storage service. Now let us learn how to manage tables in Visual Studio Server Explorer.

Example 6.4: Use Visual Studio to manage table service

Difficulty: *

1. Launch Visual Studio. Select the **VIEW**→**Server Explorer** menu to open Server Explorer.
2. In Server Explorer, expand the storage account node, right click the **Tables** node, and select the **Create Table** menu.
3. On **Create Table** dialog, enter a table name and click the **OK** button to create the table.

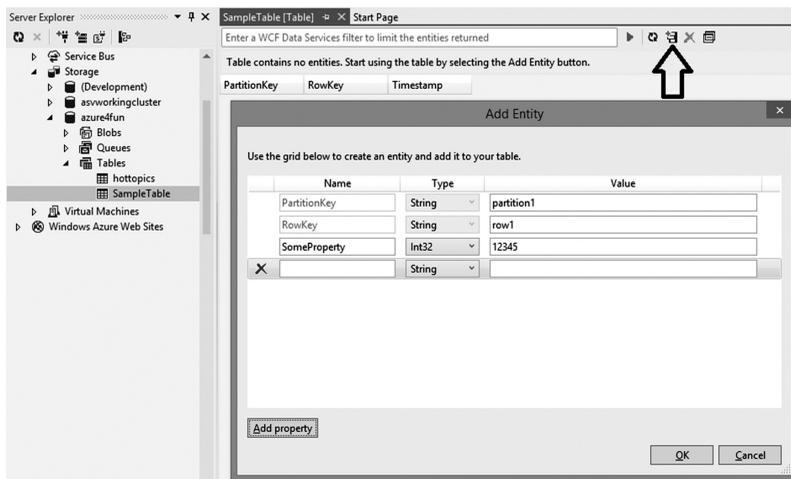


Figure 6.19 Manage table entities.

4. Double click on the newly created table to open its details view, in which you can create, modify, and delete entities. Click on the **Add Entity** icon (see Figure 6.19) to add an entity; double click on an entity to edit it; and click on the delete button to delete selected entities. The UI also supports queries written in WCF Data Services query syntax (similar to T-SQL). If you are not familiar with the syntax, you can use **Query Builder** to build a query.

In the next example, we will learn how to programmatically operate tables.

Example 6.5: Table storage example—the animal game

Difficulty: ***

In this example, we build an online game—the animal game, which is a variant of the classic Twenty Questions game. In this game, the player thinks of an animal, and the server attempts to guess the animal by asking the player less than 20 yes–no questions. The program has built-in learning capability. If it fails to guess an animal, it will ask the player to provide a question that can be used to identify the animal. Although the program initially only knows about three animals—dove, lion, and goldfish—in theory it can identify as many as 2^{20} (1,048,576) kinds of animals when it is fully educated. Hence, it would know more animals than most of us do at the end.

The essence of this program is to build and query a binary tree. The initial knowledge tree is shown in Figure 6.20. Evidently, this is a very small tree, so the server will fail to guess the animal at the beginning. For example, when the server knows the animal can fly, it will immediately guess it is a dove. If the player tells the server it has made a wrong guess, it will ask the player to enter the name of the animal, and a question that can distinguish the animal from a dove.

Assume that the player thinks of a seagull. Then the player can enter the question “Is it a sea bird?” for the server to learn how to tell apart doves and seagulls. After one round of learning, the updated knowledge tree is depicted in Figure 6.21.

How do we preserve this tree? In theory, when the tree is fully populated, we will have over two million nodes (internal nodes for questions and leaf nodes for animals)—this should not be a problem for table storage service. The question, then, is how to partition the data. Obviously, we do not want to put all entities in a single partition. Instead, we will use single-instance partitions and use partition keys made up by 1s and 0s, with 1s indicating left branches and 0s indicating right branches. Table 6.1 shows how the knowledge tree in Figure 6.21 is saved in table storage.

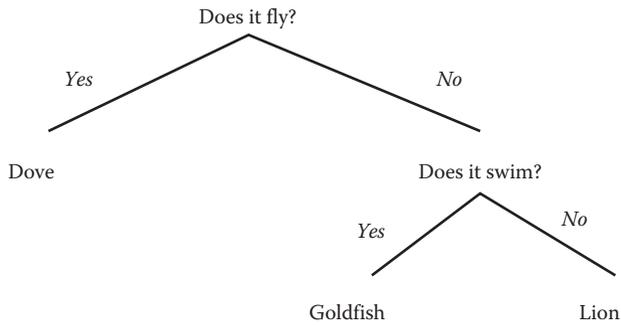


Figure 6.20 Initial knowledge tree.

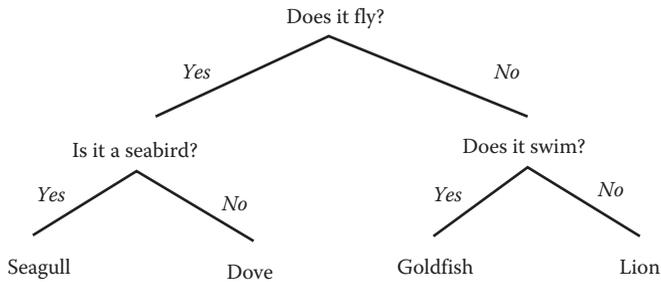


Figure 6.21 Updated knowledge tree.

Table 6.1 Entities for the Knowledge Tree

Partition	Row	Text	Is Answer?
1	—	Does it fly?	0
11	—	Is it a sea bird?	0
111	—	Seagull	1
110	—	Dove	1
10	—	Does it swim?	0
101	—	Goldfish	1
100	—	Lion	1

Now let us start to implement the application. Here we will start with an empty ASP.NET MVC 4 Web Role to build a Single Page Application (SPA).

1. Launch Visual Studio as an administrator. Create a new Microsoft Azure cloud service with an ASP.NET MVC 4 Web Role (using an empty template).

- Because the empty template does not include jQuery, let us first add a reference to jQuery by using NuGet packages. Right-click the Web Role and select the **Manage NuGet Packages** menu. On **Manage NuGet Packages** dialog, click on the **Install** button to the right of *jQuery* to install jQuery. If you do not see jQuery in the list, you can use the search box at the upper-right corner of the dialog to search for it.

Note: About NuGet

NuGet is a new way to manage assembly references. In addition to adding references to assemblies, NuGet can also add other resources, such as JavaScript files, into the project. Some NuGet packages also transform an application's configuration file even source code to bootstrap the development process. NuGet has gained great popularity in the past years. Many ISVs as well as individual contributors have published over 117,000 packages. All Microsoft Azure client libraries are published as NuGet packages as well. So, you should familiarize yourself with how to use NuGet packages. Removing NuGet packages is also very simple. All you need to do is to open the **Manage NuGet Packages** dialog again and click on the **Uninstall** buttons beside the packages you want to remove. In addition, you can update your referenced packages to new versions at any time. Finally, for those who like command lines, you can also use command-line commands to manage NuGet packages (Figure 6.22).

- Add a new **QuestionEntity** class under the **Models** folder. Table storage service requires entities to inherit *TableEntity* class. All the entities you want to save in a table need to inherit this class, which defines a partition key (PartitionKey), a row key (RowKey), a timestamp (Timestamp), and an ETag. Our subclass defines the additional *Text* property and the *IsAnswer* property.

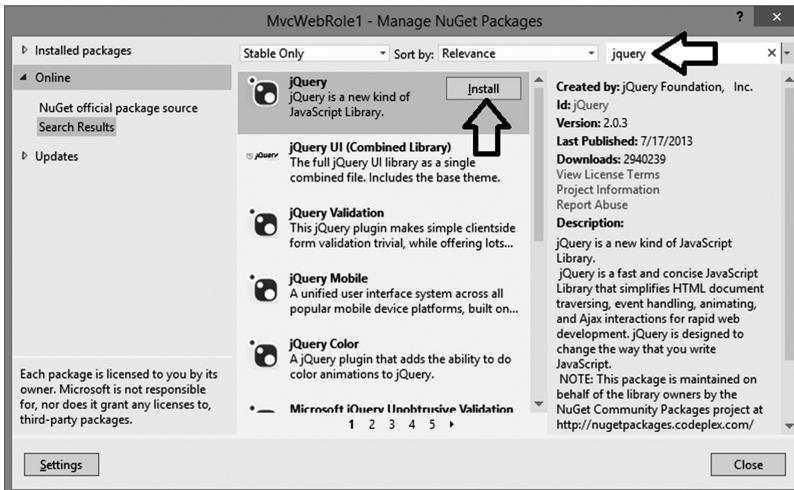


Figure 6.22 Manage NuGet packages.

```

using Microsoft.WindowsAzure.Storage.Table;
...
public class QuestionEntity: TableEntity
{
    public QuestionEntity(string questionId, string text, bool
        isAnswer)
    {
        this.PartitionKey = questionId;
        this.RowKey = "";
        this.Text = text;
        this.IsAnswer = isAnswer;
    }
    public QuestionEntity() { }
    public string Text { get; set; }
    public bool IsAnswer { get; set; }
}

```

4. Add a new empty API Controller named **QuestionController** under the **Controllers** folder. The controller has the knowledge tree in Figure 6.20 predefined as a starting point. Its three GET methods return the root node (line 12 in Code List 6.22), return a child node (line 16), and update the current node (line 24), respectively. When updating a node, the question entered by the user replaces the node. The original node becomes the right child of the question node. The answer to the question is added as the left child of the question node. A node's identifier is made by appending either a "1" or a "0" to its parent's identifier. For example, the two children of node "101" are "1011" (representing a true answer) and "1010" (representing a false answer).
5. Add a new Empty MVC Controller named **HomeController** to the **Controllers** folder. The sole responsibility of this controller is to return the main view:

```

public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}

```

6. Create a **Home** folder under the **Views** folder. Then, add an **Index.cshtml** view under the Home folder. Because this is an SPA, the UI contains all required display elements as well as AJAX calls to the API controller. The complete code is shown in Code List 6.23.
7. Now you can press F5 to test the application. Because currently the *QuestionController* saves its knowledge tree in memory, the application forgets everything when it is restarted. Next, we will store the knowledge tree in table storage.
8. You need a storage account to use table storage. Use the same method as in step 15 of Example 6.3 to add a *StorageConnectionString* to your Web Role. Then, import the following namespaces to the *QuestionController* class:

```

using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Table;
using Microsoft.WindowsAzure;

```

CODE LIST 6.22 QUESTIONCONTROLLER API CONTROLLER

```

1: public class QuestionController : ApiController
2: {
3:     private static Dictionary<string, QuestionEntity> mQuestions =
4:         new Dictionary<string, QuestionEntity>
5:         {
6:             {"1", new QuestionEntity("1", "Does it fly?",
7:                                     false)},
8:             {"11", new QuestionEntity("11", "Dove", true)},
9:             {"10", new QuestionEntity("10", "Does it
10:                swim?", false)},
11:             {"101", new QuestionEntity("101", "Goldfish",
12:                                        true)},
13:             {"100", new QuestionEntity("100", "Lion",
14:                                        true)}
15:         };
16:     public QuestionEntity Get()
17:     {
18:         return mQuestions["1"];
19:     }
20:     public QuestionEntity Get(string id, bool positive)
21:     {
22:         string index = id + (positive ? "1" : "0");
23:         if (mQuestions.ContainsKey(index))
24:             return mQuestions[index];
25:         else
26:             return null;
27:     }
28:     public QuestionEntity Get(string id, string newquestion,
29:                               string animal)
30:     {
31:         var negativeId = id + "0";
32:         var positiveId = id + "1";
33:         mQuestions.Add(negativeId,
34:                       new QuestionEntity(negativeId, mQuestions[id].
35:                                           Text, true));
36:         mQuestions[id].Text = newquestion;
37:         mQuestions[id].IsAnswer = false;
38:         mQuestions.Add(positiveId,
39:                       new QuestionEntity(positiveId, animal, true));
40:         return mQuestions[id];
41:     }
42: }

```

CODE LIST 6.23 INDEX VIEW

```

<style>
  div {
    margin: 20px;
    padding: 10px;
    background-color: lightgray;
    display: block;
  }
  h2 {
    margin: 20px;
  }

  h3 {
    margin: 20px;
  }

  input {
    margin: 5px;
  }
</style>
<h2>The Animal Game</h2>
<!-- UI for asking questions -->
<div id="guessPanel">
  <h3 id="question"></h3>
  <input type="radio" name="choices" value="yes" />Yes
  <input type="radio" name="choices" value="no" />No
  <input type="button" id="submit" value="Confirm" />
</div>
<!-- UI for making a guess -->
<div id="confirmPanel" style="display: none;">
  <h3>I guess the animal on your mind is: <span id="guess"></span>. Am I right?</h3>
  <input type="button" id="yesButton" value="Yes" />
  <input type="button" id="noButton" value="No" />
</div>
<!-- UI for learning -->
<div id="learnPanel" style="display: none;">
  <h3>You won! Please teach me how to improve.</h3>
  The animal you think about is:
  <br />
  <input type="text" id="animal" /><br />
  Please enter a question to separate the animal from <span
  id="guessText"></span>.<br />
  <input type="text" id="newQuestion" /><br />
  <input type="button" id="teach" value="Submit" />
  <input type="button" id="restart2" class="restart"
  value="Restart" />
</div>

```

```

<!-- UI for thank you message -->
<div id="thankyouPanel" style="display: none;">
  <h3>Thank you!</h3>
  <input type="button" id="restart1" class="restart"
    value="Restart" />
</div>
<script src="~/Scripts/jquery-2.0.3.min.js"></script>
<script>
  var currentQuestion;
  var lastPanel = $('#guessPanel'); //currently displayed panel
  $(function () {
    $.getJSON('/api/Question', function (json) {
      setcurrentQuestion(json);
    });
  });
  function flipDiv(div2) {
    lastPanel.slideUp();
    lastPanel = $('#' + div2);
    lastPanel.slideDown();
  }
  $('#submit').click(function () {
    var positive = false;
    if ($('#input[name=choices]:checked').val() == 'yes')
      positive = true;
    $.getJSON('/api/Question?id=' + currentQuestion.PartitionKey
      + '&positive=' + positive,
      function (json) {
        if (json == null) //didn't find knowledge
          node
            flipDiv('learnPanel');
        else if (json.IsAnswer) { //found an answer,
          make a guess
            flipDiv('confirmPanel');
            setcurrentQuestion(json);
        }
        else { //didn't find a answer, ask a
          question
            flipDiv('guessPanel');
            setcurrentQuestion(json);
        }
      }
    );
  });
  $('#yesButton').click(function () { //user confirms the answer,
    game ends.
    flipDiv('thankyouPanel');
  });

```

```

$('#noButton').click(function () { //user denies the answer,
    start learning.
    flipDiv('learnPanel');
});
$('#teach').click(function () { //record the question and the
    answer user has provided.
    $.getJSON('/api/Question?id=' + currentQuestion.PartitionKey
        + '&newquestion='
        + encodeURIComponent($('#newQuestion').val())
        + '&animal='
        + encodeURIComponent($('#animal').val()),
        function (json) {
            flipDiv('guessPanel');
            setcurrentQuestion(json);
        });
});
$('.restart').click(function () { //reset game
    $.getJSON('/api/Question', function (json) {
        flipDiv('guessPanel');
        setcurrentQuestion(json);
    });
});
function setcurrentQuestion(data) { //update UI with current
    question
    currentQuestion = data;
    $('#question').text(currentQuestion.Text);
    $('#guess').text(currentQuestion.Text);
    $('#guessText').text(currentQuestion.Text);
}
</script>

```

9. Similar to Example 6.3, add several private variables and a constructor to the *QuestionController* class.

```

CloudStorageAccount mAccount;
CloudTableClient mClient;
const string tableName = "animals";
public QuestionController()
{
    mAccount = CloudStorageAccount.Parse(
        CloudConfigurationManager.GetSetting("StorageConnection
            String"));
    mClient = mAccount.CreateCloudTableClient();
}

```

10. Next, let us change the original Dictionary to a List. We save these data for table initialization:

```
private List<QuestionEntity> mQuestions = new List<QuestionEntity>
{
    new QuestionEntity("1", "Does it fly?", false),
    new QuestionEntity("11", "Dove", true),
    new QuestionEntity("10", "Does it swim?", false),
    new QuestionEntity("101", "Goldfish", true),
    new QuestionEntity("100", "Lion", true)
};
```

11. Add a new *getTableReference* private method to the *QuestionController* class. This method gets a reference to a table and initializes the table with the initial knowledge tree. Line 3 in Code List 6.24 checks if a table exists. Line 6 creates a table if the table does not exist. Lines 10–12 show the process of inserting a new entity—first by creating an Insert operation (TableOperation) and then executing it. This is also the general process of executing any operation (insert, update, and delete) on a table.
12. Add a new *getEntityById* private method to the *QuestionController* class. The method reads an entity with a specified partition key. The method takes only one parameter because we only use partition keys in this example (Code List 6.25).
13. Finally, we can modify the three GET methods. To update an entity, you need to retrieve it first (line 14 of Code List 6.26), update it (lines 19 and 20), and then use a Replace operation to update it in the table (lines 21 and 22).

CODE LIST 6.24 GETTABLEREFERENCE METHOD

```
1 private CloudTable getTableReference()
2 {
3     var table = mClient.GetTableReference(tableName);
4     if (!table.Exists())
5     {
6         if (table.CreateIfNotExists())
7         {
8             foreach (var question in mQuestions)
9             {
10                TableOperation insertOperation =
11                    TableOperation.Insert(question);
12                table.Execute(insertOperation);
13            }
14        }
15    }
16    return table;
17 }
```

CODE LIST 6.25 GETENTITYBYID METHOD

```

private QuestionEntity getEntityById(string id)
{
    TableOperation retrieve =
        TableOperation.Retrieve<QuestionEntity>(id, "");
    TableResult retrieveResult =
        getTableReference().Execute(retrieve);
    if (retrieveResult.Result != null)
        return (QuestionEntity)retrieveResult.Result;
    else
        return null;
}

```

CODE LIST 6.26 GET METHODS BASED ON TABLE STORAGE

```

1 public QuestionEntity Get()
2 {
3     return getEntityById("1");
4 }
5 public QuestionEntity Get(string id, bool positive)
6 {
7     string index = id + (positive ? "1" : "0");
8     return getEntityById(index);
9 }
10 public QuestionEntity Get(string id, string newquestion, string
    animal)
11 {
12     var negativeId = id + "0";
13     var positiveId = id + "1";
14     var entity = getEntityById(id);
15     var table = getTableReference();
16     TableOperation insertOperation = TableOperation.Insert(
17         new QuestionEntity(negativeId, entity.Text, true));
18     table.Execute(insertOperation);
19     entity.Text = newquestion;
20     entity.IsAnswer = false;
21     TableOperation updateOperation = TableOperation.
        Replace(entity);
22     table.Execute(updateOperation);
23     insertOperation = TableOperation.Insert(
24         new QuestionEntity(positiveId, animal, true));
25     table.Execute(insertOperation);
26     return entity;
27 }

```

14. Figure 6.23 shows several screens of the game.
15. Figure 6.24 shows the entities in the *animal* table after several rounds of the game.

In the previous example, we used the *Insert* operation and the *Replace* operation. Other operations include *Merge*, *InsertOrMerge*, *InsertOrReplace*, and *Delete*. In addition, at the table level, you can use the *DeleteIfExists* method on the *CloudTable* class to delete a table.



Figure 6.23 Application screens.

PartitionKey	RowKey	Timestamp	Text	IsAnswer
1		8/17/2013 9:07:...	Does it fly?	False
10		8/17/2013 9:07:...	Does it swim?	False
100		8/17/2013 9:11:...	Does it eat grass?	False
1000		8/17/2013 9:11:...	Lion	True
1001		8/17/2013 9:11:...	Deer	True
101		8/17/2013 9:07:...	Goldfish	True
11		8/17/2013 9:07:...	Is it a sea bird?	False
110		8/17/2013 9:07:...	Dove	True
111		8/17/2013 9:07:...	Seagull	True

Figure 6.24 Entities in table storage.

6.4.3 Query Table Data

In the previous example, we queried for a specific entity. You can use *TableQuery* to perform more complex queries. Here are some typical examples:

- Query all entities in a partition

```
TableQuery<QuestionEntity> query =
    new TableQuery<QuestionEntity>()
        .Where(TableQuery.GenerateFilterCondition(
            "PartitionKey", QueryComparisons.Equal, "1"));
foreach (var entity in table.ExecuteQuery(query))
{
    var rowKey = entity.RowKey;
    var text = entity.Text;
    ...
}
```

- Query some entities in a partition

Code List 6.27 is an example to query some entities in a partition. This code queries all entities with Row Keys less than “11” in partition “1.” Of course, if you run this query against the table in Example 6.3, you can get only one record (because each entity has its own partition). Generally speaking, the query returns a list of entities (line 9). With a series of filters (lines 4 and 7) and Boolean operators (line 6), you can build very complex queries.

6.4.4 Other Operations

- InsertOrReplace—Insert the entity if it does not exist, or replace the existing entity.
- InsertOrMerge—Insert the entity if it does not exist, or merge the entity into the existing version.
- Merge—Merge entities. The result of a Merge operation is a union of two entities. The merged entity has all the properties of both entities. If the two entities have the same property, the value from the newer entity is used. You need to specify an ETag during the

CODE LIST 6.27 QUERY SOME ENTITIES IN A PARTITION

```
TableQuery<QuestionEntity> query =
    new TableQuery<QuestionEntity>()
        .Where(TableQuery.CombineFilters(
            TableQuery.GenerateFilterCondition("PartitionKey",
                QueryComparisons.Equal, "1"),
            TableOperators.And,
            TableQuery.GenerateFilterCondition("RowKey",
                QueryComparisons.LessThan, "11")));
foreach (var entity in table.ExecuteQuery(query))
{
    ...
}
```

Merge operation. As we introduced earlier, ETag is used for optimistic concurrency control. When you merge an entity, the ETag you provide to the Merge operation has to match with the entity's ETag. To force an unconditional merge, you can set ETag to "*" as shown in the following code:

```
TableOperation op = TableOperation.Merge
    (new QuestionEntity("1", "Does it fly?", false) {ETag = "*"});
table.Execute(op);
```

6.4.5 Batch Operations

In this example, we did not use batched operations, because each of our entities has its own partition, while batch operations only apply to entities within a same partition. Batch operations can group up to 100 table operations into a single transaction. Table storage service ensures that all operations in a batch either succeed together or fail together to protect data integrity. In addition, because all operations are handled as a single transaction, they can be executed faster, and they are charged as a single operation. At the time of writing this book, Microsoft Azure charges \$0.01 per 100,000 operations. Batch operations have the following limitations:

- You can batch update, delete, insert, merge, and replace operations.
- You cannot batch a query operation with insert, update, or delete operations.
- A batch can contain up to 100 operations. An entity can only appear once in a transaction.
- All operations have to be in the same partition.
- The maximum workload of a batch is 4 MB.

Code List 6.28 is an example of a simple batch.

6.4.6 Dynamic Table Entities

We have already shown that a table can hold entities with different properties. In some cases, instead of retrieving whole entities, we may only want to retrieve a couple of properties on those entities. Table storage service provides a *DynamicTableEntity* class for this purpose. In addition to a *PartitionKey*, a *RowKey*, and a *Timestamp*, the class defines an *IDictionary*, which can be populated with the only properties you need. In Code List 6.29, the code queries the *animal* table in Example 6.5, but it only retrieves and returns the text property on entities.

CODE LIST 6.28 A SIMPLE BATCH

```
TableBatchOperation batch = new TableBatchOperation();
batch.Insert(new SomeEntity("111", "Frog"));
batch.Insert(new SomeEntity("222", "Fox"));
table.ExecuteBatch(batch);
```

CODE LIST 6.29 DYNAMIC TABLE ENTITIES

```

var table = getTableReference();
TableQuery query = new TableQuery().Select(new string[] { "Text" });
foreach (var entity in table.ExecuteQuery(query))
{
    EntityProperty text;
    if (!entity.Properties.TryGetValue("Text", out text))
        throw new ArgumentException("Property not found!");
}

```

6.4.7 Shared Access Signatures

Similar to BLOB storage service, table storage service also supports Shared Access Signatures (SAS). In addition to creating SASs at the table level, you can create SASs based on certain partitions, or certain rows in a partition. In other words, you can actually grant different access rights to different entities in the same table. For example, for a table holding information from different customers, you can create a SAS for each of the customer partitions so that each customer can only access his or her own data.

6.5 Use Queue Storage

Queue storage service provides queues for sending a large number of messages with sizes smaller than 64 KB. Queue is an important tool to implement asynchronous patterns. Instead of directly communicating with each other, a job creator and a job process pass jobs via a queue. This design decouples the two parties so that they can operate independently—they can have different throughputs, they do not need to be online at the same time, and they do not need to know about each other. This kind of decoupling makes the system easily maintainable and extensible.

6.5.1 Queue Storage Overview

The data structure of Queue storage service is relatively simple, as shown in Figure 6.25.

- Account
All accesses to Queues are done using a storage account.
- Queue
A Queue stores a series of Messages. A Message has to be saved in a queue.
- Message
A Message can be in any format, up to 64 KB in size.

Example 6.6: Use Visual Studio to manage Queue Service

Difficulty: *

1. Launch Visual Studio. Select the **VIEW**→**Server Explorer** menu to open Server Explorer.
2. In Server Explorer, expand the storage account node, right-click the **Queues** node, and select the **Create Queue** menu.

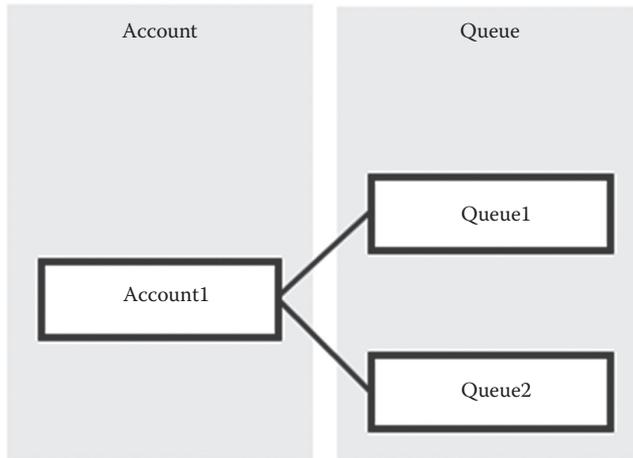


Figure 6.25 Queue storage.

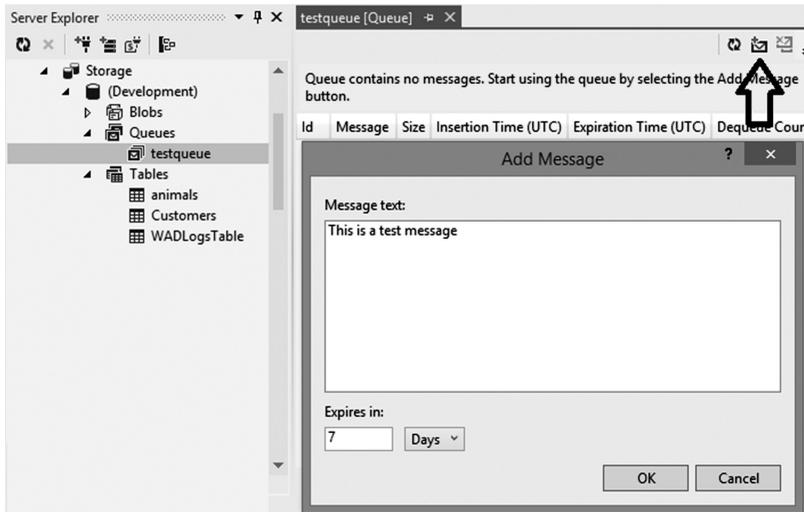


Figure 6.26 Manage Queues in Visual Studio.

3. On **Create Queue** dialog, enter a queue name and click the **OK** button to create the queue.
4. Double click the queue node to open its details view, where you can enqueue messages, dequeue messages, or clear the queue, as shown in Figure 6.26.

6.5.2 Programmatically Operate Queues

Because using Queue storage service is similar to using two other storage services, we will not provide a complete example here, but only list out some code snippets for typical queue operations.

■ Create a Queue

```
using Microsoft.WindowsAzure.Storage.Queue;
...
CloudQueueClient client = mAccount.CreateCloudQueueClient();
CloudQueue queue = client.GetQueueReference("jobqueue");
queue.CreateIfNotExists();
```

■ Add a message

```
CloudQueueMessage message = new CloudQueueMessage("test message");
queue.AddMessage(message);
```

■ Peek a message

```
CloudQueueMessage message = queue.PeekMessage();
var str = message.AsString;
```

■ Retrieve a message

You can use the *GetMessage* method to retrieve a message from a queue. The *GetMessage* method does not directly remove the message from the queue, but places a lock on the message so that the message becomes invisible to other callers. The lock holder has to explicitly call the *DeleteMessage* method to remove the message, or renew the lock within 30 s, or the message will be unlocked so it can be retrieved by other callers.

```
CloudQueueMessage message = queue.GetMessage();
var str = message.AsString;
queue.DeleteMessage(message);
```

■ Update a message

After a message is retrieved, you can use the lock window to update the message. You can update the message content (line 2) as well as specify when updates are available to other callers (line 4).

```
1: CloudQueueMessage message = queue.GetMessage();
2: message.SetMessageContent("new content");
3: queue.UpdateMessage(message,
4:     TimeSpan.FromSeconds(0.0), //make updates immediately visible
    to others
5:     MessageUpdateFields.Content | MessageUpdateFields.Visibility);
```

■ Batch process a group of messages

You can use *GetMessages* to retrieve a group of messages. For example, the following code reads 10 messages and places a 1 min lock on them and then processes the messages one by one.

```
foreach (CloudQueueMessage message in
    queue.GetMessages(10, TimeSpan.FromMinutes(1)))
{
    ...
    queue.DeleteMessage(message);
}
```

■ Read queue length

You can use *ApproximateMessageCount* property to estimate the current queue length. This is an estimation because the property is updated when the *FetchAttributes* method is called, and the queue length may have changed between the following two lines:

```
queue.FetchAttributes();
int? count = queue.ApproximateMessageCount;
```

■ Delete a queue

```
queue.Delete();
```

6.6 Monitor Storage Accounts

You can use Microsoft Azure Management Portal to monitor your storage accounts. By default, Microsoft Azure does not collect metrics data from storage accounts, as shown in Figure 6.27. You need to explicitly enable monitoring on the account's **CONFIGURE** page.

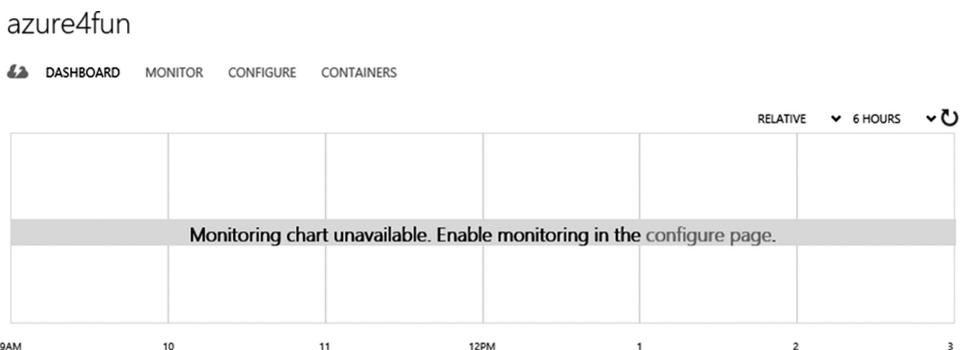


Figure 6.27 Monitoring is turned off by default.

6.6.1 Configure Storage Service Monitoring

On a storage account’s **CONFIGURE** page, you can modify monitoring and logging settings. You can choose from two monitoring levels: Minimal or Verbose.

- Minimal

By default, minimal level collects the metrics on success percentage, availability, and total requests. However, you can customize metrics to be collected by the **ADD METRICS** icon on the command bar.
- Verbose

In addition to collecting the metrics under the minimal level, you can collect the metrics on storage operations under the Verbose level. The configure page is shown in Figure 6.28.

In addition, you can collect logs from storage services. The collected logs are saved in a BLOB container. You can also specify the retention policy of monitoring data. By specifying 0 days as retention policy you may save logs permanently. Obviously, saving more log data in storage services accrues higher costs.

After monitoring has been enabled, you can monitor the performance of your storage accounts on Microsoft Azure Management Portal. Just like monitoring other services, you can choose which data series (up to 6) are to be plotted, and add/remove metrics to be monitored, as shown in Figure 6.29.



Figure 6.28 Monitoring configuration page.



Figure 6.29 Storage service monitoring page.

6.6.2 Cost of Service Monitoring

Microsoft Azure does not collect storage service metrics data because retaining the data generates additional costs. In addition, transactions on monitoring and log data, such as creating log BLOB and adding data entities to tables, are not free either (though the cost is really low). So, when you configure monitoring settings, you need to understand the cost implications. If you have turned on retention policy, the delete operations performed by the system are free. But pruning data manually is charged as regular operations.

6.7 Summary

In this example, we studied Microsoft Azure BLOB storage, Table storage, and Queue storage services through a series of examples. The rich set of NoSQL storage services provided by Microsoft Azure can satisfy most requirements of large-volume data storage and processing. All storage services provide corresponding client libraries as well as REST APIs. You can also control access to storage entities by access keys, Shared Access Signatures, and stored data policies. To improve performance and reduce costs, you can group multiple operations in batches, optimize partitions of data, use paralleled uploads/downloads, and use dynamic entities to retrieve subsets of entity properties. Moreover, you can monitor the performance of your storage accounts on Microsoft Azure Management Portal as well as collect logs for diagnostics purposes.

Chapter 7

Virtual Machines and Virtual Networks

In previous chapters, our definitions of websites and cloud services are logical definitions. These definitions are not bound to specific virtual or physical machines. In other words, the applications are totally separated from underlying infrastructure. It is only when these applications are deployed on Microsoft Azure that they are mapped to specific virtual machines. This kind of separation is an essential requirement for applications to fully leverage the benefits of PaaS. So, websites and cloud services are preferred ways to build new cloud services. However, some projects may have additional requirements that cannot be satisfied by websites or cloud services, such as using Linux operation systems, joining virtual machines to on-premise networks, and customizing virtual machines in depth. To satisfy these requirements, Microsoft Azure provides Infrastructure as a Service (IaaS), which allows you to work on an infrastructure level by directly managing virtual machines and virtual networks.

7.1 Microsoft Azure IaaS

You can use Microsoft Azure IaaS to create Windows-based or Linux-based virtual machines. Instead of going through the lengthy process of acquiring and managing physical servers, you can provision a new virtual machine on Microsoft Azure in minutes. In addition, Microsoft Azure provides 99.95% SLA to multi-instance virtual machines and allows you to manage these virtual machines using familiar tools such as Microsoft System Center.

Now let us learn the basic processes of virtual machine management on Microsoft Azure by looking at several examples.

Example 7.1: Hello, Windows Virtual Machines!

Difficulty: *

In this example, we will use Microsoft Azure Management Portal to create a new Windows-based virtual machine.

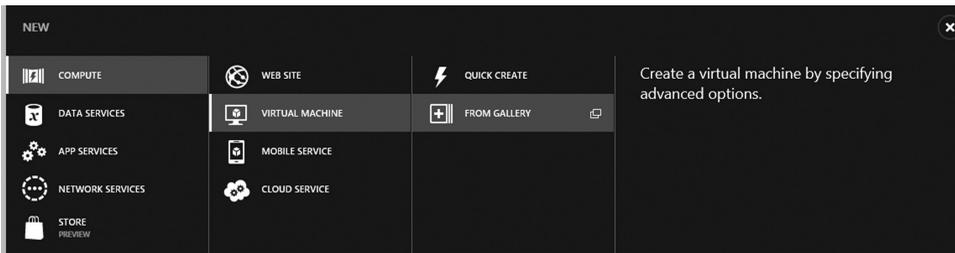


Figure 7.1 Creating a new virtual machine.

1. Log in to Microsoft Azure Management Portal.
2. On the command bar, click the **NEW** icon, and then select **COMPUTE**→**VIRTUAL MACHINE**→**FROM GALLERY**, as shown in Figure 7.1.
3. On **CREATE A VIRTUAL MACHINE** dialog, select the **Windows Server 2012 R2 Datacenter** image (see Figure 7.2), and then click the next button.
4. On the **Virtual machine configuration** page, select **VERSION RELEASE DATA** (you would usually select the latest version). Pick a virtual machine size from various **BASIC** sizes or **STANDARD** sizes. Then, enter the virtual machine name, as well as the administrator credentials, and click the next button to continue (Figure 7.3).
5. On the next page, you can create a new cloud service for the virtual machine or join the virtual machine to an existing cloud service. In the previous chapters, we mentioned that a cloud service is a container that can hold web roles and worker roles. You can also put virtual machines in a cloud service. We will come back to multi-instance virtual machines later in this chapter. Pick a region where you want the virtual machine to be hosted (we will discuss Virtual Networks later). Create a new storage account to save disk images used by the virtual machine. Leave **AVAILABILITY SET** to empty (more on this later). Then, configure endpoints to be defined on the virtual machine. By default, Microsoft Azure defines a remote desktop endpoint as well as a remote PowerShell endpoint for virtual machine management. You can define additional endpoints just like defining endpoints on web roles or worker roles. For

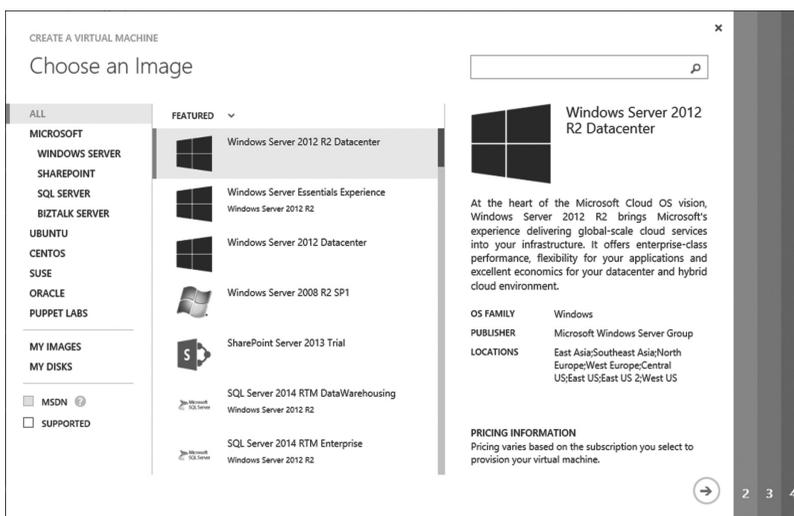


Figure 7.2 Virtual machine image gallery.

CREATE A VIRTUAL MACHINE

Virtual machine configuration

VERSION RELEASE DATE [?]
3/17/2014

VIRTUAL MACHINE NAME [?]
new2012srv

TIER
BASIC STANDARD

SIZE
A1 (1 core, 1.75 GB memory)

NEW USER NAME
haishi

NEW PASSWORD CONFIRM

Windows Server 2012 R2 Datacenter

At the heart of the Microsoft Cloud OS vision, Windows Server 2012 R2 brings Microsoft's experience delivering global-scale cloud services into your infrastructure. It offers enterprise-class performance, flexibility for your applications and excellent economics for your datacenter and hybrid cloud environment.

OS FAMILY
Windows

PUBLISHER
Microsoft Windows Server Group

LOCATIONS
East Asia;Southeast Asia;North Europe;West Europe;Central US;East US;East US 2;West US

PRICING INFORMATION
Pricing varies based on the subscription you select to provision your virtual machine.

1 3 4

Figure 7.3 Virtual machine configuration.

CREATE A VIRTUAL MACHINE

Virtual machine configuration

CLOUD SERVICE [?]
Create a new cloud service

CLOUD SERVICE DNS NAME
new2012srv .cloudapp.net

REGION/AFFINITY GROUP/VIRTUAL NETWORK [?]
East US

STORAGE ACCOUNT
Use an automatically generated storage account

AVAILABILITY SET [?]
(None)

ENDPOINTS [?]

NAME	PROTOCOL	PUBLIC PORT	PRIVATE PORT
Remote Desktop	TCP	AUTO	3389
PowerShell	TCP	5986	5986

ENTER OR SELECT A VALUE

Windows Server 2012 R2 Datacenter

At the heart of the Microsoft Cloud OS vision, Windows Server 2012 R2 brings Microsoft's experience delivering global-scale cloud services into your infrastructure. It offers enterprise-class performance, flexibility for your applications and excellent economics for your datacenter and hybrid cloud environment.

OS FAMILY
Windows

PUBLISHER
Microsoft Windows Server Group

LOCATIONS
East Asia;Southeast Asia;North Europe;West Europe;Central US;East US;East US 2;West US

PRICING INFORMATION
Pricing varies based on the subscription you select to provision your virtual machine.

1 2 4

Figure 7.4 Add the virtual machine to a cloud service.

example, if your virtual machine runs a public-accessible web server, you need to define an HTTP-based endpoint at port 80 to allow HTTP traffic. Click the next button to continue, as shown in Figure 7.4.

- On the last page, you can select to install one or more virtual machine extensions. Microsoft Azure virtual machine extensions can be dynamically injected to standard images to further customize your virtual machines. Finally, click the check button to provision the virtual machine (Figure 7.5).
- A couple of minutes later, your virtual machine is provisioned and is ready to use. On the command bar, click the **CONNECT** icon to connect to the virtual machine, as shown in Figure 7.6.

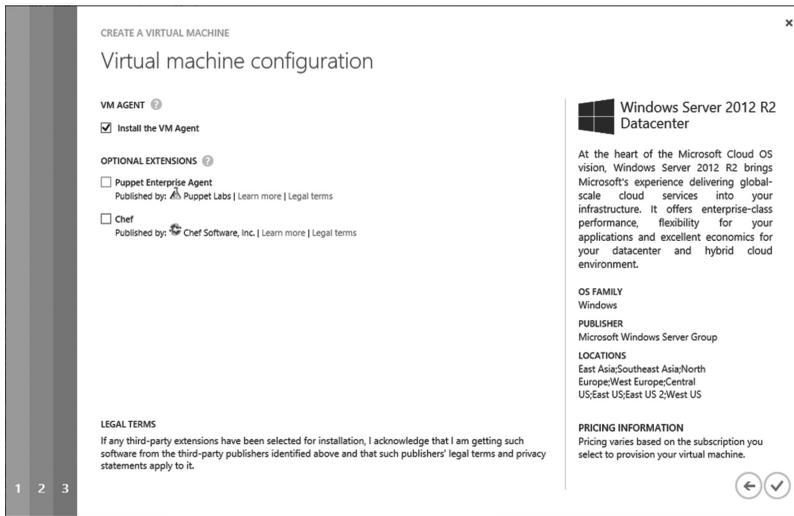


Figure 7.5 Endpoint configuration.

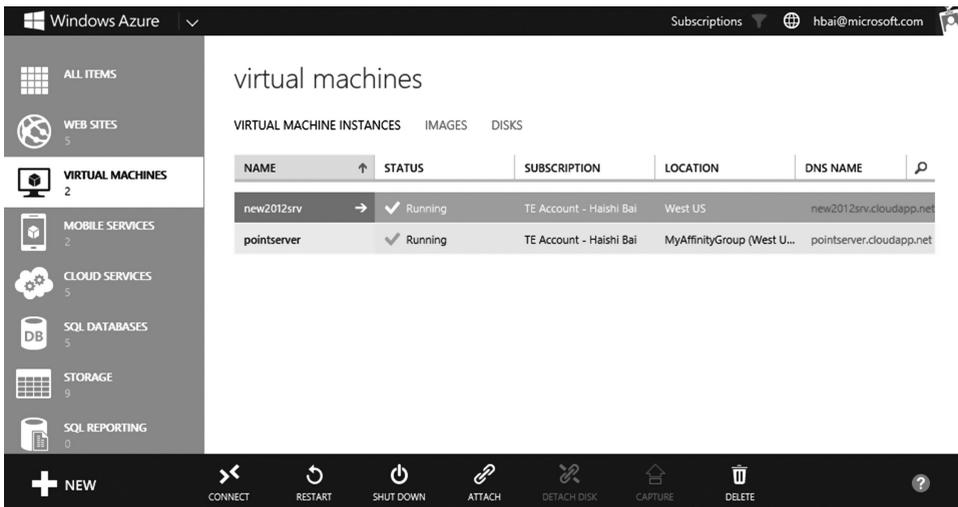


Figure 7.6 The new virtual machine on Microsoft Azure Management Portal.

8. Once you see the prompt for opening the **.rdp** file, click the **Open** button to continue (Figure 7.7).
9. On the **Remote Desktop Connection** dialog, click the **Connect** button to continue (this is because the **.rdp** file is not signed), as shown in Figure 7.8.
10. On the **Windows Security** dialog, enter the credential you specified when creating the virtual machine, and then click the **OK** button to continue, as shown in Figure 7.9.
11. Because the certificate of the virtual machine is self-signed, you will see a certificate warning. Click **Yes** to continue, as shown in Figure 7.10.
12. Now you have successfully connected to the remote desktop to manage the virtual machine, as shown in Figure 7.11.



Figure 7.7 Open .rdp file.



Figure 7.8 Remote desktop connection prompt.



Figure 7.9 Log in to the remote desktop.



Figure 7.10 Certificate prompt.

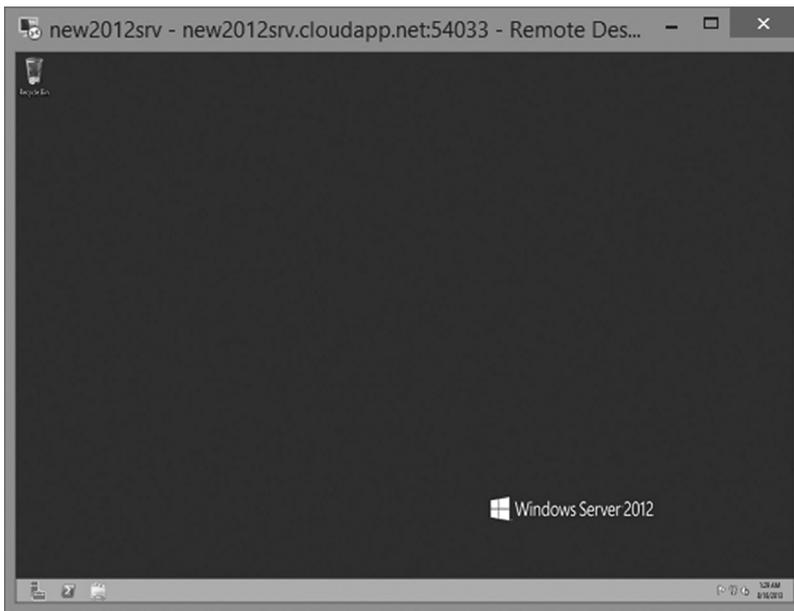


Figure 7.11 Remote desktop window.

On the virtual machine's details page, you can observe that Microsoft Azure has created a virtual disk that holds the operation system. Your virtual machine is running on this virtual disk. The next section will introduce you to using additional data virtual disks.

Example 7.2: Hello, Linux Virtual Machines!

Difficulty: *

In this example, we will use Microsoft Azure Management Portal to create a new Linux-based virtual machine. Some steps will be omitted for simplicity. You can refer to the screenshots in Example 7.1 if needed.

1. Log in to Microsoft Azure Management Portal.
2. On the command bar, click the **NEW** icon, and then select **COMPUTE**→**VIRTUAL MACHINE**→**FROM GALLERY**.
3. On **CREATE A VIRTUAL MACHINE** dialog, select the **OpenLogic CentOS 6.3** image, and then click the next button.
4. On **CREATE A VIRTUAL MACHINE** dialog, enter a name for the virtual machine. Select the virtual machine size as **Extra Small**. Uncheck the **UPLOAD COMPATIBLE SSH KEY FOR AUTHENTICATION** checkbox, and check the **PROVIDE A PASSWORD** checkbox. Enter a password and then click the next button to continue, as shown in Figure 7.12.
5. Follow steps 5 and 6 in Example 7.1 to complete creating the virtual machine.

Note: By clicking the check button to create the virtual machine, you acknowledge that you are getting this software from OpenLogic and that OpenLogic's legal terms apply to it. Microsoft does not provide rights for third-party software.

CREATE A VIRTUAL MACHINE

Virtual machine configuration

VIRTUAL MACHINE NAME ⓘ

SIZE

NEW USER NAME

AUTHENTICATION ⓘ

UPLOAD COMPATIBLE SSH KEY FOR AUTHENTICATION

PROVIDE A PASSWORD

NEW PASSWORD **CONFIRM**

OpenLogic CentOS 6.3

This distribution of Linux is based on CentOS version 6.3 and is provided by OpenLogic. It contains an installation of the Basic Server packages.

PUBLISHER OpenLogic

OS FAMILY Linux

LOCATION East Asia/Southeast Asia/North Europe/West Europe/Central US/East US/East US 2/West US

PRICING INFORMATION
Pricing varies based on the subscription you select to provision your virtual machine.

1 ← → 3 4

Figure 7.12 Virtual machine configuration.

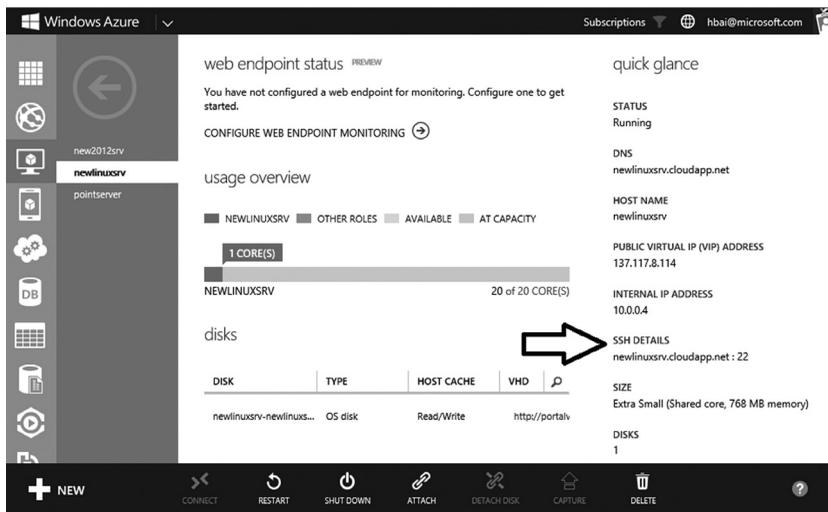


Figure 7.13 SSH information on the portal.

- After the virtual machine has been provisioned, you can use an SSH client, such as PuTTY or OpenSSH, to connect to the virtual machine. Here we will use PuTTY.

Note: You can download PuTTY, which is a free implementation of Telnet and SSH authored by Simon Tatha, from its official site <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>.

- You can find SSH details on Microsoft Azure Management Portal, as shown in Figure 7.13.
- Enter the information provided earlier to PuTTY, and click the **Open** button to continue, as shown in Figure 7.14.
- Then, on the terminal window, you can log in to your new Linux virtual machine, as shown in Figure 7.15.

7.2 Disk Images and Virtual Disks

In Section 1.3.1, a list of currently supported virtual machine templates is provided. These templates are standardized system disk images. When you use one of these images to create a virtual machine, the image is mapped into a virtual disk. You can also create system virtual disks locally, upload the disk file (in *.vhd* format) to Microsoft Azure, and then use the disk file to create a new virtual machine. In addition, you can capture an image of an existing virtual machine to the image gallery, and use it to create new virtual machines later.

Now, let us go through a couple of examples of using disk images and virtual disks.

Example 7.3: Use a virtual data disk

Difficulty: **

In this example, we will add two data disks to the virtual machine we have created in Example 7.1. We will add the first data disk via Microsoft Azure Management Portal and the second disk by uploading a local disk file.

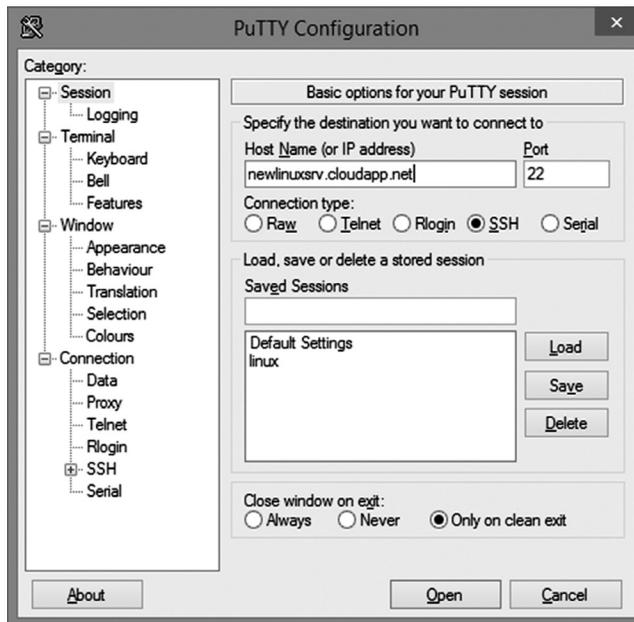


Figure 7.14 Enter SSH connection information to PuTTY.

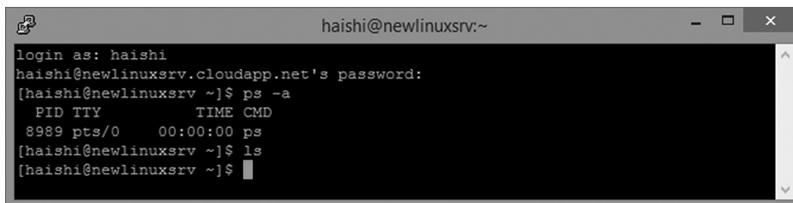


Figure 7.15 Remote terminal window to the Linux machine.

1. Log in to Microsoft Azure Management Portal.
2. On the virtual machine list view, or a virtual machine's DASHBOARD view, click on the **ATTACH** icon on the command bar, and select **Attach empty disk** menu. On **Attach an empty disk to the virtual machine** dialog, enter a disk size (in this case, 10G). Set host cache preference to NONE. Then, click the check button to complete the operation, as shown in Figure 7.16.
3. Once the disk has been attached, you can see this disk in the disk list of the virtual machine (see Figure 7.17). But before you can use the disk, you need to log in to the virtual machine and initialize it.
4. Log in to the virtual machine via remote desktop. Then, in **Server Manager**, switch to **File and Storage Services** view. You can see the new data disk in the disk list (see Figure 7.18). Right-click the disk and select the **Initialize** menu.
5. After initialization has finished, right-click the disk and select the **New Volume** menu.
6. On the **New Volume Wizard** dialog, keep clicking the **Next** buttons to accept all defaults, and finally click the **Create** button to create the disk volume. Once the disk volume is created, the disk can be used normally.
7. Next, we will create a second data disk by uploading a *.vhd* file. In the following steps, we will create a new *.vhd* file on a Windows 8 machine.

Attach an empty disk to the virtual machine

VIRTUAL MACHINE NAME

STORAGE LOCATION

FILE NAME

SIZE (GB)

HOST CACHE PREFERENCE
 NONE READ ONLY READ/WRITE ?

Figure 7.16 Attach an empty data disk to a virtual machine.

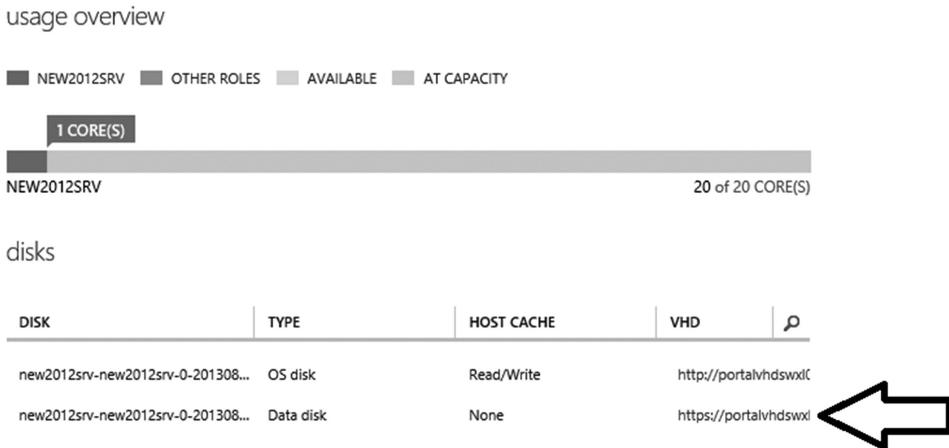


Figure 7.17 Disk list of a virtual machine.

8. Open **Control Panel**→**Administrative Tools**→**Computer Management**.
9. Right-click the **Storage**→**Disk Management** node, and select the **Create VHD** menu, as shown in Figure 7.19.
10. On **Create and Attach Virtual Hard Disk** dialog, enter a location for the *.vhd* file (you can omit the *.vhd* extension. For example, when *c:\haishi\newdisk* is entered in the **Location**

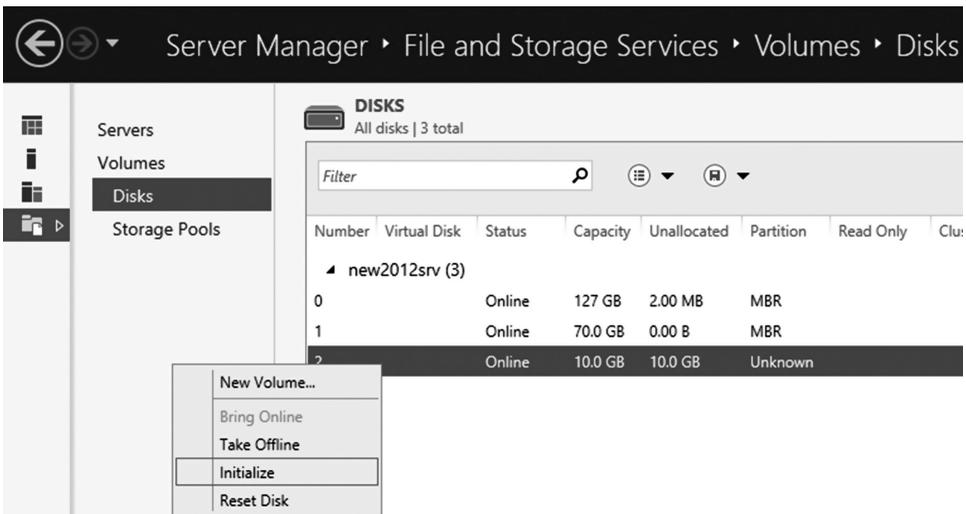


Figure 7.18 Initialize disk in server manager.

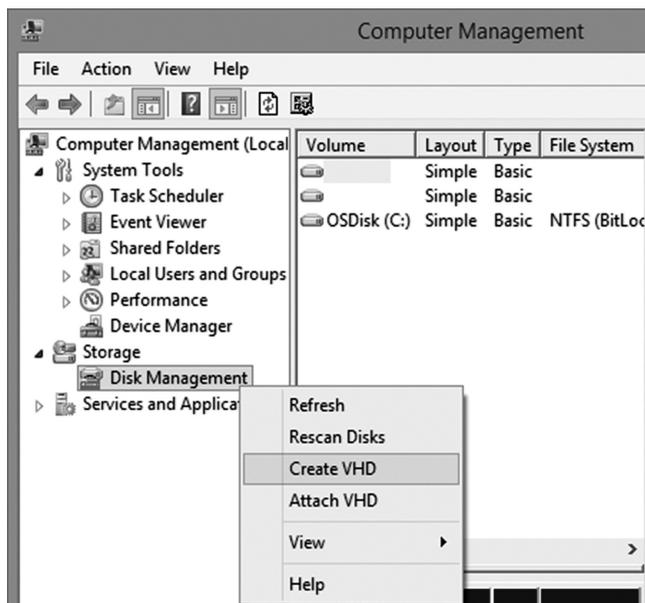


Figure 7.19 Create a new VHD file.

field, the generated *.vhd* file will be *c:\baishi\newdisk.vhd*). Enter a size for the new disk (in this case, 200 MB). Then, click the **OK** button, as shown in Figure 7.20.

- Right-click the newly created virtual disk and select the **Initialize Disk** menu, as shown in Figure 7.21.
- Once initialization is done. Right-click the disk and select the **New Simple Volume** menu. Accept all default values on the wizard and create the volume.

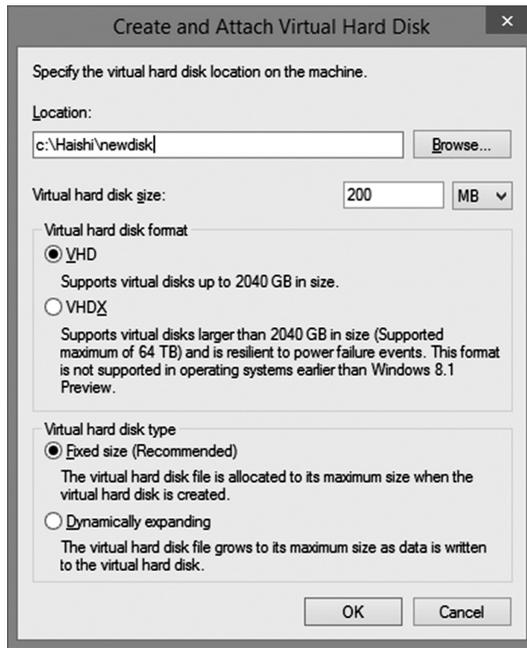


Figure 7.20 Create and attach virtual hard disk.

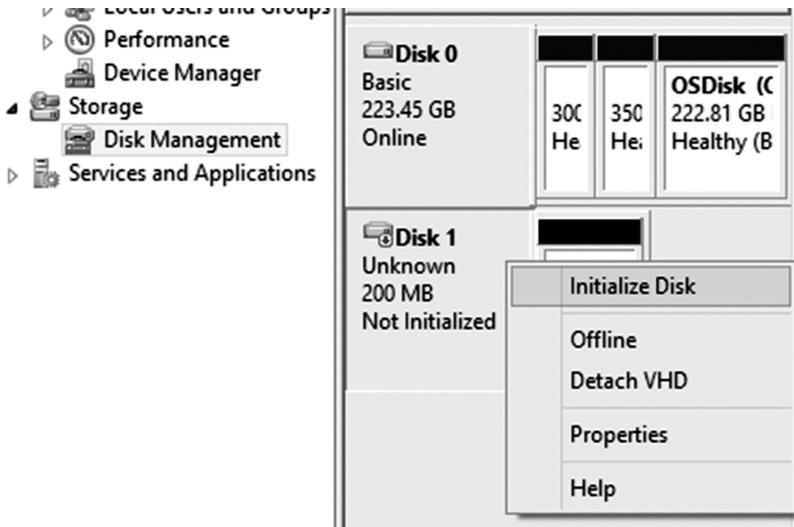


Figure 7.21 Initialize local virtual disk.

13. Back on the **Computer Management** window, right-click on the virtual disk again and select the **Detach VHD** menu. On **Detach Virtual Hard Disk** dialog, click the **OK** button to continue.
14. Now the *.vhd* file is ready to be uploaded to Microsoft Azure. Here I use an open-source tool AzCopy, which is a command-line tool you can download from GitHub (<https://github.com/downloads/%20WindowsAzure/azure-sdk-downloads/AzCopy.zip>).

```
C:\Haishi\Tools>azcopy c:\haishi http://portalvhdswx10q40h0m0w.blob.core.windows
.net/vhds newdisk.vhd /destKey:
Finished transfer: newdisk.vhd
Transfer summary:
-----
Total files transferred: 1
Transfer successfully: 1
Transfer failed: 0
```

Figure 7.22 Use AzCopy to upload *.vhd*.

- Use AzCopy to upload the *.vhd* file. Note that you need to use */blobtype:page* to specify the BLOB is a page BLOB. You can get the address of *[BLOB URL]* from the STORAGE LOCATION field in Figure 7.16. See Figure 7.22 for a sample result.

```
azcopy [local folder] [BLOB URL] [file name]/destKey: [blob account
access key]/V/blobtype:page
```

- Back in Microsoft Azure Management Portal, open virtual machine list view, and then switch to **DISKS** view. On the command bar, click the **CREATE** icon. On **Create a disk from VHD** dialog, enter a disk name, and then click on the folder icon besides the **VHD URL** field.
- Choose the storage account you have used in step 15. Locate the *.vhd* file you just uploaded. Select the file, and then click the **Open** button. Back on **Create a disk from VHD** dialog, click the check button to create the disk (Figure 7.23).
- Switch back to the virtual machine list view, click the **ATTACH** icon on the command bar, and select **Attach disk** menu. On **Attach a disk to the virtual machine** dialog, pick the disk you just created, and click the check button to continue.
- Once the disk has been attached, you can log in to the machine to manage the newly attached data disk.

NAME	LAST MODIFIED	SIZE
2euprieb.ouf201305062315250967.v...	5/6/2013 4:15:25 PM	127 GB
haishi2012-mystestserver-2013-05-06...	5/6/2013 3:56:14 PM	127 GB
haishilinux-mylinux-2013-05-06.vhd	5/6/2013 6:29:39 PM	30 GB
mynew2012-mynew2012-2013-05-0...	8/4/2013 10:42:04 PM	127 GB
mytestserver-mystestserver-0506-1.vhd	5/6/2013 3:56:14 PM	10 GB
new2012srv-new2012srv-0817-1.vhd	8/17/2013 10:23:17 PM	10 GB
new2012srv-new2012srv-2013-08-1...	8/17/2013 11:39:23 PM	127 GB
newdisk.vhd	8/17/2013 11:31:13 PM	200 MB
newlinuxsrv-newlinuxsrv-2013-08-18...	8/17/2013 11:39:18 PM	30 GB
pointserver-pointserver-2013-05-07...	8/17/2013 11:39:14 PM	127 GB
revivedwindow-otherserver-2013-0...	5/7/2013 1:44:17 AM	127 GB
vagrantlinux-vagrantlinux-2013-07-2...	8/17/2013 5:29:31 PM	29.3 GB

File name: [*.*vhd]

Figure 7.23 Choose the *.vhd* file in a storage account.

Example 7.4: Create and use images

Difficulty: **

In this example, we will capture an image of the earlier virtual machine. Before you can capture an image of a system disk, you need to run a *sysprep* tool on it (more on the tool soon). Then, you will be able to safely capture an image of a system disk. You do not need to run *sysprep* on data disks.

1. Log in to the virtual machine through remote desktop.
2. Run the **Command Prompt** as an administrator.
3. Go to `c:\Windows\System32\sysprep` folder, and run `sysprep.exe`.

Note: About sysprep.exe

To put it simply, sysprep is a Windows tool to reset virtual machine settings so that when the virtual machine is rebooted, it will enter the configure mode to set up new product license, computer name, user name, etc. Images in the gallery already have the sysprep tool included.

4. On the **System Preparation Tool** window, select **Enter System Out-of-Box Experience (OOBE)**. Check the **Generalize** checkbox. Then, set **Shutdown Options** to **Shutdown**. Click the **OK** button to complete this step, as shown in Figure 7.24.
5. When the tool finishes, the virtual machine is shut down. Now, you can click the **CAPTURE** icon on the command bar to capture an image of your system, as shown in Figure 7.25.
6. On **Capture an image from a virtual machine** dialog, enter a name for the image to be captured. Check the **I have run Sysprep on the virtual machine** checkbox. Then click the check button to complete the operation (the original virtual will be deleted once the image has been captured), as shown in Figure 7.26.
7. Once the image has been captured, you can use the image from MY IMAGES category when creating a new virtual machine, as shown in Figure 7.27.

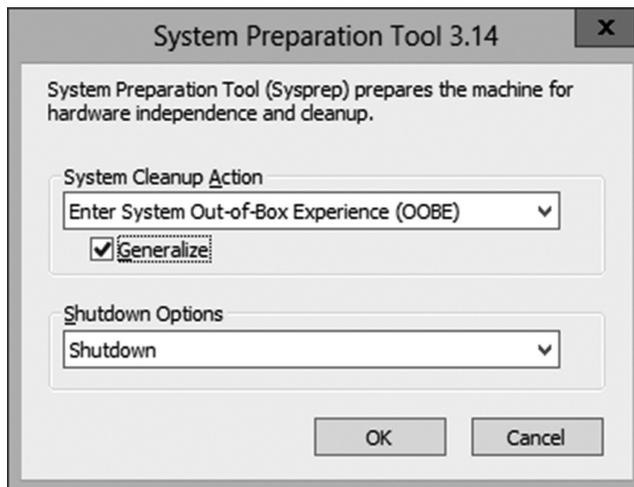


Figure 7.24 System preparation tool window.

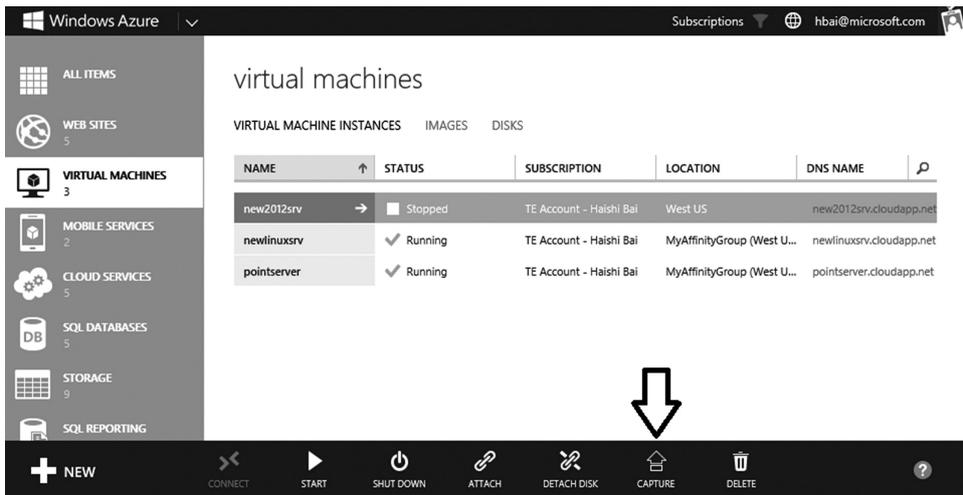


Figure 7.25 The virtual machine is stopped after sysprep.

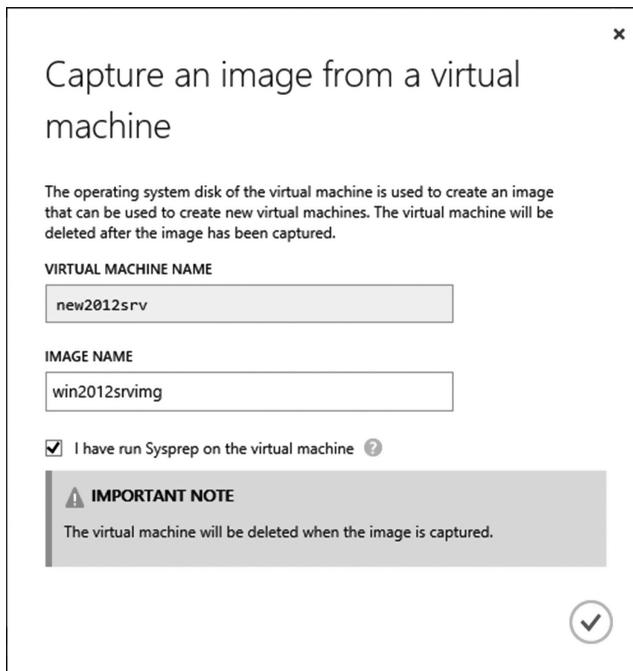


Figure 7.26 Capture a new image from the virtual machine.

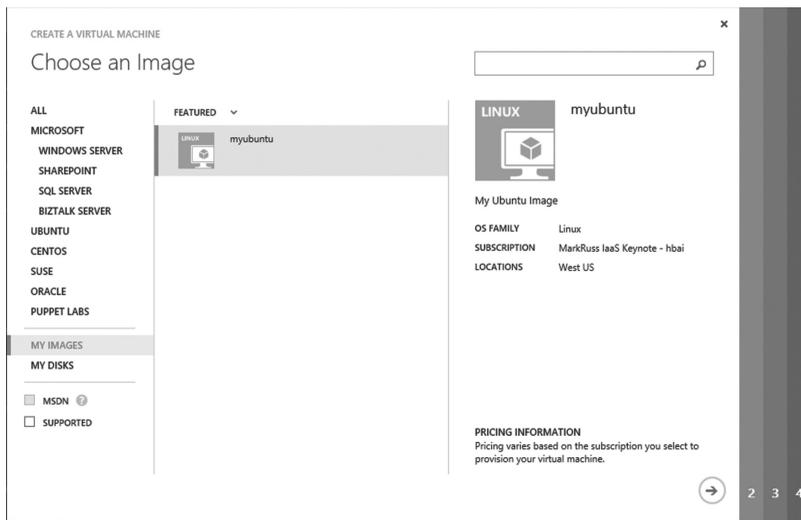


Figure 7.27 Use the captured image to create a virtual machine.

7.3 Virtual Machine Communications

If you view a virtual machine as another role in a cloud service, then it is easy to understand that communications between virtual machines are quite similar to those in web role/worker role. In the following sections, we will introduce endpoints and load balancing of virtual machines.

7.3.1 Virtual Machine Endpoints

Virtual machines within the same cloud service or on the same Virtual Network can directly communicate with each other. In addition, you can define more endpoints on virtual machines just like you define endpoints on web roles or worker roles. For example, if you want to run a web server on a virtual machine, you can create a TCP Input Endpoint on port 80.

You can examine and edit endpoints on the virtual machine's details page. Next we will learn how to use endpoints on a virtual machine by an example.

Example 7.5: Running Node.js on a Linux virtual machine

Difficulty: ****

In this example, we will install and configure Node.js on the Linux virtual machine we have created in Example 7.2. We will compile the Node.js source code and install it. This is obviously not the most convenient way, but it is a good practice for Windows developers to get more familiarized with the Linux system.

1. Log in to Microsoft Azure Management Portal. Open the **ENDPOINT** view of the Linux virtual machine. Define a new standalone TCP Input Endpoint on port 80, as shown in Figure 7.28.
2. Use PuTTY to log in to the Linux virtual machine.
3. The CentOS image provided by Microsoft Azure does not have a kernel package, which we will install first. Use *vi* to edit the configuration file:

newlinuxsrv

DASHBOARD MONITOR ENDPOINTS CONFIGURE

NAME	↑	PROTOCOL	PUBLIC PORT	PRIVATE PORT	LOAD-BALANCED SET NA...
HTTP		TCP	80	80	-
SSH		TCP	22	22	-

Figure 7.28 Define a TCP endpoint on the virtual machine.

```
sudo vi/etc/ym.conf
```

Edit the file and comment out this line (by prefixing the line with a “#”):

```
exclude=kernel*
```

Then execute the command

```
sudo yum install kernel-headers
```

4. Now, we can use Git to download and compile Node.js:

```
sudo -i
yum install gcc-c++ make git
cd/usr/local/src/
git clone git://github.com/joyent/node.git
cd node
./configure
make
make install
```

5. Compilation takes a while. Once it is done, go to the `/usr/local/src/node/out/Releases/` folder.

6. Use `vi` to create a `Server.js` file:

```
var http = require("http");
http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/html"});
  response.write("<h1>Hello from Node.js on Azure Linux VM</h1>");
  response.end();
}).listen(80);
```

7. Save the `Server.js` file. Then use

```
./node Server.js
```

to launch the Node.js server.

8. Now, you can open a browser and use the address `http://[virtual machine name].cloudapp.net/` to access the web page served by Node.js, as shown in Figure 7.29.



Figure 7.29 Web page served by the Node.js server.

7.3.2 Virtual Machines under the Same Cloud Service

We have learned through Example 7.1 that when you create a new virtual machine, the virtual machine is put into a cloud service. You can also join multiple virtual machines to the same cloud service. The virtual machines within the same cloud service can directly communicate with each other. In addition, you can put multiple virtual machines behind a load balancer to provide high availability. In this section we will use an example to learn how virtual machines communicate with each other. And then we will implement a load balancing scenario in another example. We will discuss high availability further in Chapter 9.

Example 7.6: Virtual machine communications within a Cloud Service

Difficulty: ***

In this example, we will provision a new Windows Server 2012 server and join it to the same cloud service to which the virtual machine from Example 7.1 belongs. Two virtual machines within the same cloud service will be able to communicate with each other, share files, and address each other over the private network.

1. Log in to Microsoft Azure Management Portal. Create a new Microsoft Azure 2012 virtual machine. The steps are almost identical to those in Example 7.1, except for step 5. Instead of creating a new cloud service, you should join the virtual machine to the same cloud service in Example 7.1 (see Figure 7.30). In addition, when you enter the user credentials, make sure

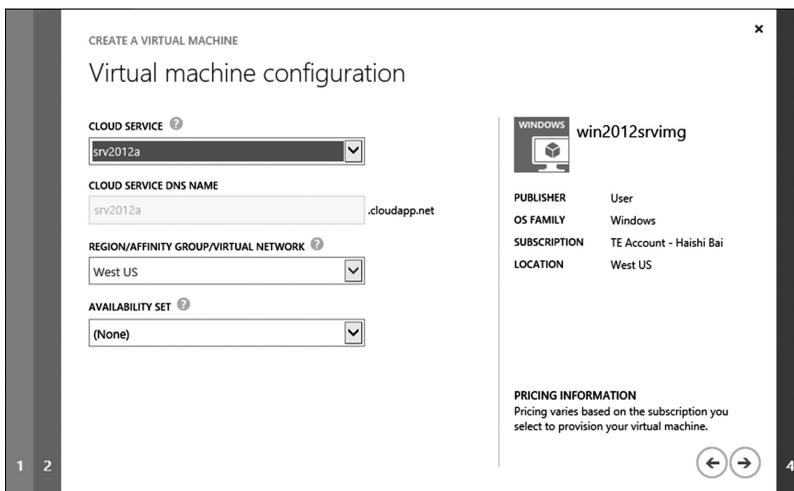


Figure 7.30 Join a virtual machine to an existing cloud service.

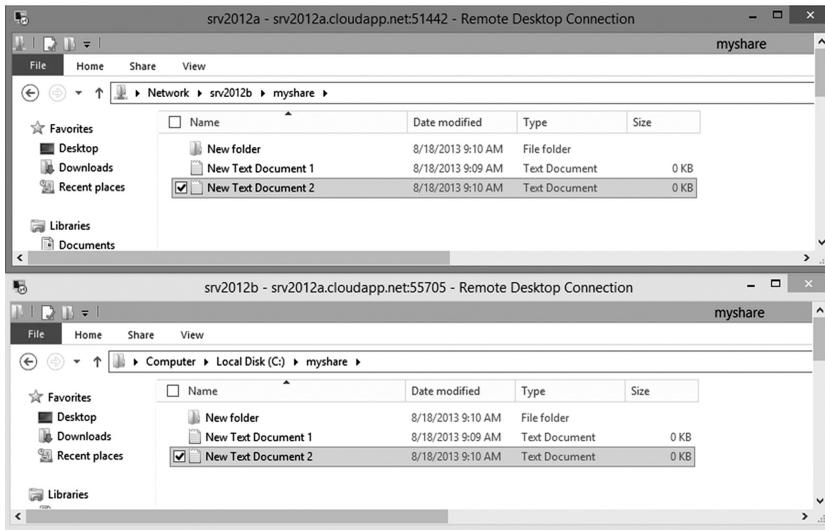


Figure 7.31 Share files between two virtual machines.

to use the exact user name and password as you have entered in Example 7.1—because the machines are not under a domain, we enter the same credentials on to both machines to allow them to share files with each other. It is not mandatory to use the same credentials on both machines. We use the same credentials simply to make file sharing easier.

2. Once both machines have been created, log in to both machines through remote desktop. Create a new folder on one of the machines and share the folder with the user. Then, on the other virtual machine, access the folder by the address `\\[virtual machine name]\[shared folder name]` (note the machine name does not include the *cloudapp.net* postfix). In Figure 7.31, you can see a *myshare* folder being shared between the two virtual machines.

Example 7.7: Virtual machine load balancing

Difficulty: ***

In this example, we will enable IIS service on both virtual machines in Example 7.6, and then put the two IIS servers behind a load balancer.

1. Log in to one of the virtual machines. Launch **Server Manager**. Then click **Add roles and features** (see Figure 7.32).
2. On **Add Roles and Features Wizard** dialog, click the **Next** buttons till you reach the **Server Roles** tab. Check *Web Server (IIS)*, and then click the **Next** button again, as shown in Figure 7.33.
3. Accept all defaults till you reach the last screen. Then click the **Install** button. After installation is done, click the **Close** button to close the wizard.
4. Create a **default.htm** file under the `c:\inetpub\wwwroot` folder:

```
<h1>Web page served by Server A</h1>
```

5. On the virtual machine, browse to `http://localhost` to make sure the web page is working correctly.

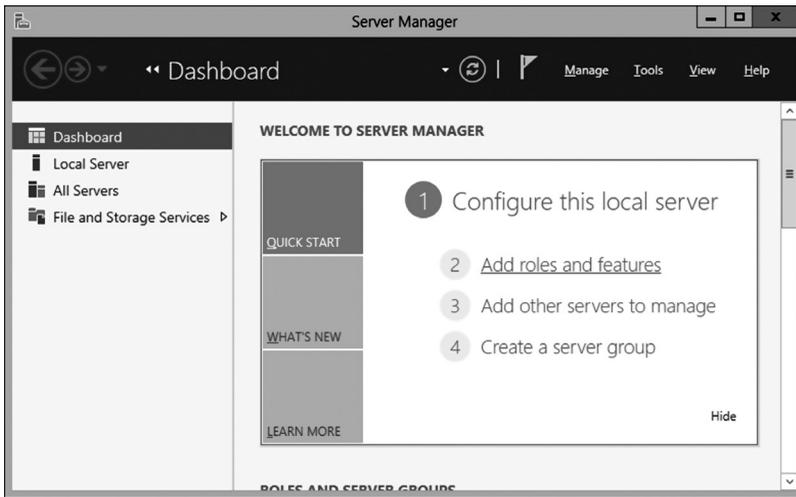


Figure 7.32 Add roles and features link in Server Manager.

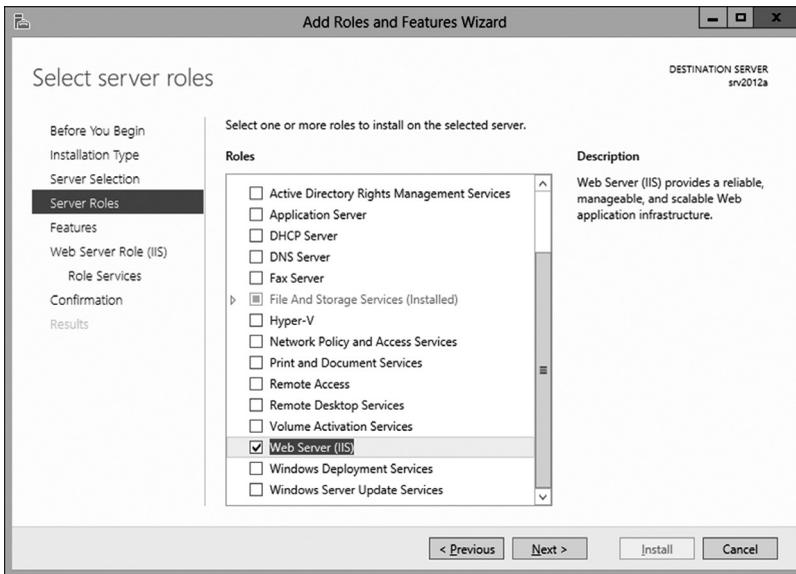


Figure 7.33 Add Roles and Features Wizard.

- Repeat the previous steps on the second virtual machine. The only difference is that the content of *default.htm* on the second machine is slightly different so we can observe which server is serving the page:

```
<h1>Web page served by Server B</h1>
```

- Log in to Microsoft Azure Management Portal. Pick one of the virtual machines and add a TCP endpoint on port 80. Make sure **CREATE A LOAD-BALANCED SET** is checked, and then click the next button to continue, as shown in Figure 7.34.

EDIT ENDPOINT

Specify the details of the endpoint

NAME
HTTP

PROTOCOL
TCP

PUBLIC PORT
80

PRIVATE PORT
80

CREATE A LOAD-BALANCED SET ?

→ 2

Figure 7.34 Add a new TCP endpoint.

8. In the next step, enter a name for the load-balanced set, and click the check button to complete the operation (Figure 7.35).
9. On the second virtual machine, create a new endpoint in the earlier load-balanced set, as shown in Figure 7.36.
10. On the next screen, set a name for the endpoint. Note in this case that both the public port and the private port have been locked from editing. Click the check button to finish the operation, as shown in Figure 7.37.

EDIT ENDPOINT

Configure the load-balanced set

Endpoints that are load-balanced across multiple virtual machines are added to a load-balanced set.

LOAD-BALANCED SET NAME
balancedweb

PROBE PROTOCOL
TCP

PROBE PORT
80

PROBE INTERVAL
15 SECONDS

NUMBER OF PROBES ?
2

← ✓

1

Figure 7.35 Creating a load-balanced set.

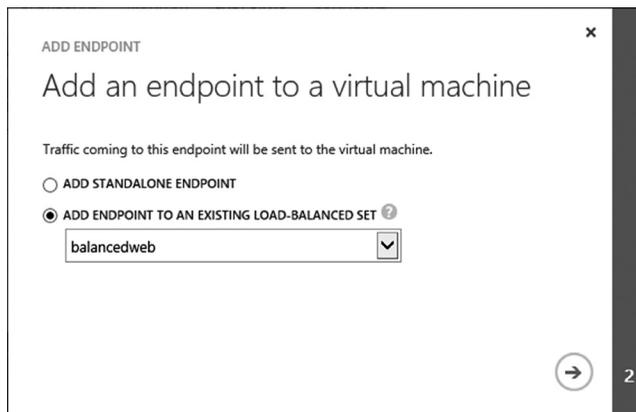


Figure 7.36 Add an endpoint to an existing load-balanced set.

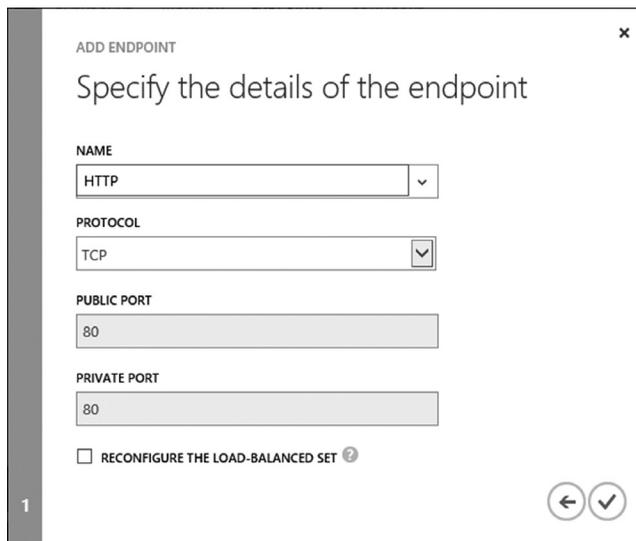


Figure 7.37 Endpoint details.

11. Now, the two virtual machines are joined to the same load balancer. When we access the address `http://[cloud service name].cloud.app.net`, the requests will be distributed to these two machines. Because the `default.htm` files are different on the machines, when you keep refreshing the browser by Ctrl+F5 (to avoid caching), you will observe the page coming from different servers, as shown in Figure 7.38.

7.4 Virtual Networks

Microsoft Azure Virtual Network is designed primarily for three scenarios:

- Create a virtual network on Microsoft Azure. You can allocate a private IPv4 address space (10.x, 172.x, 192.x) for virtual machines to communicate over it.

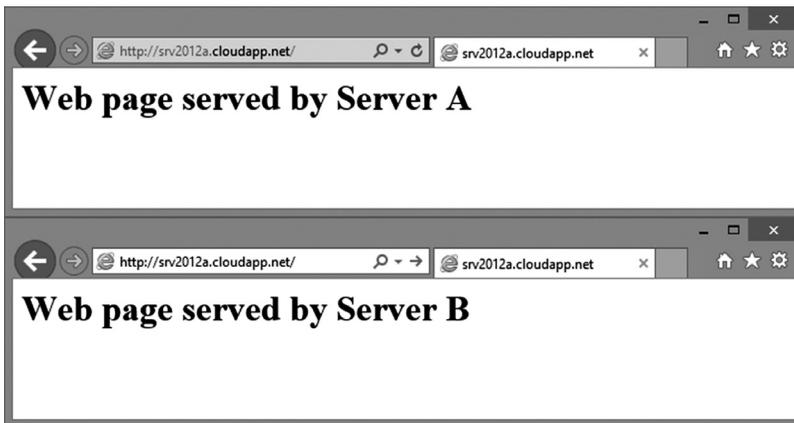


Figure 7.38 Load-balanced virtual machines.

- Extend your local network to Microsoft Azure. You can establish a site-to-site VPN connection to a Microsoft Azure Virtual Network through an IPsec VPN device or a VPN software.
- Link a single computer to a Microsoft Azure Virtual Network through a point-to-site VPN connection.

7.4.1 Virtual Networks Overview

Virtual machines joined on a Virtual Network have stable private IPs. Although technically they still use DHCP, their IP addresses do not change during their lifecycles because the IP leases are permanent. Stable IP addresses allow you to set up services that require static IP addresses, such as domain servers. In addition, you can also create and use your own DNS servers on Virtual Networks.

Virtual Networks also allow you to create Hybrid Cloud solutions across multiple cloud data centers and on-premise networks. Many enterprises are running mission-critical systems on their on-premise data centers. The complexity, risk, and cost of migrating all these systems to cloud are often unacceptable. Hybrid Cloud allows enterprises to integrate existing legacy systems to new services on cloud, providing a smooth path for cloud adoption with controlled risks, paced schedules, and progressive returns.

Example 7.8: Create a virtual network

Difficulty: **

In this example, we will use Microsoft Azure Management Portal to create a new virtual network, and join a virtual machine to it.

1. Log in to Microsoft Azure Management Portal.
2. On the command bar, click the **NEW** icon. Then, select **NETWORK SERVICES**→**VIRTUAL NETWORK**→**CUSTOM CREATE**.
3. On **Virtual Network Details** dialog, enter a name for the virtual network. Then, select **Create a new affinity group** in the **AFFINITY GROUP** field. Pick a region and a name for the affinity group (see Section 6.2.1 for details on affinity groups), and then click the next button to continue, as shown in Figure 7.39.

Figure 7.39 Virtual network details dialog.

Figure 7.40 DNS server and VPN connection.

4. On the DNS Servers and VPN Connectivity page, click the next button to continue (we will not use either custom DNS or VPN connections in this example), as shown in Figure 7.40.
5. On the **Virtual Network Address Spaces** page, you can define IPv4 address spaces and subnets. Here we will take all defaults and directly click the check icon to complete the operation, as shown in Figure 7.41.
6. Create a new Windows Server 2012 virtual machine. On the **Virtual machine configuration** page, pick the virtual network you have just created. Select a subnet (see Figure 7.42). When the virtual machine is created, it will be joined to the virtual network, and be assigned an IP address within the virtual network address spaces.
7. After the virtual machine is created, you can observe its IP address on its **DASHBOARD** page. Figure 7.43 shows that the virtual machine that was created got assigned to IP address 10.0.0.4.

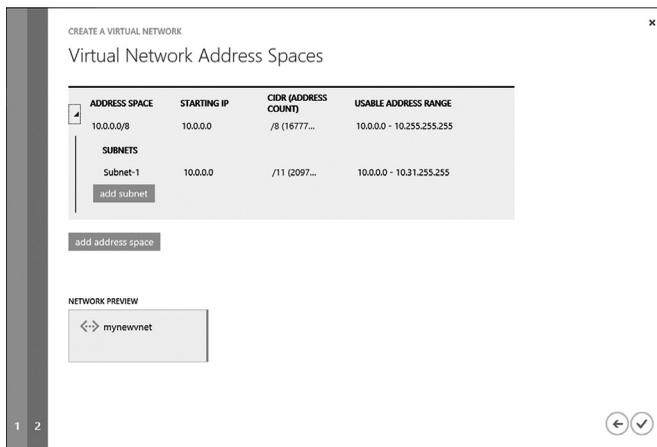


Figure 7.41 Define Virtual Network Address Spaces.

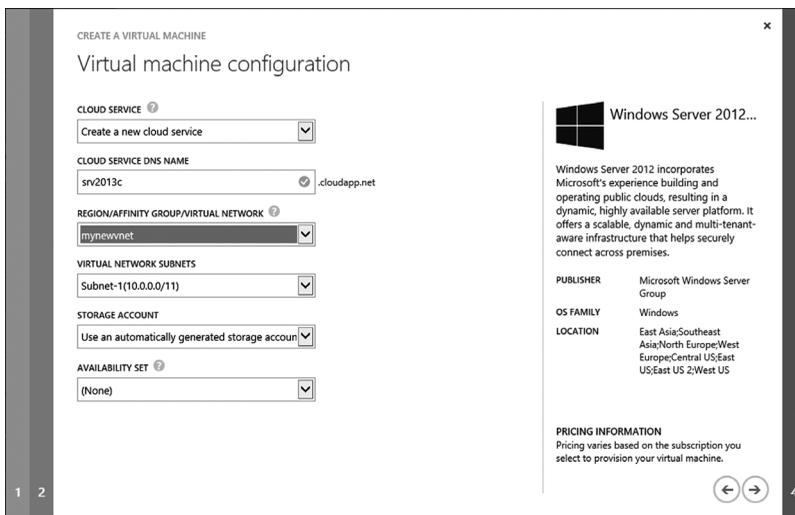


Figure 7.42 Join a virtual machine to a virtual network.

7.4.2 Point-to-Site Virtual Network

Configuring a point-to-site virtual network needs three steps:

1. Create the virtual network and a virtual gateway.
2. Acquire and upload a digital certificate for client authentication.
3. Download and install the VPN client.

Now let us go through the steps in Example 7.9.

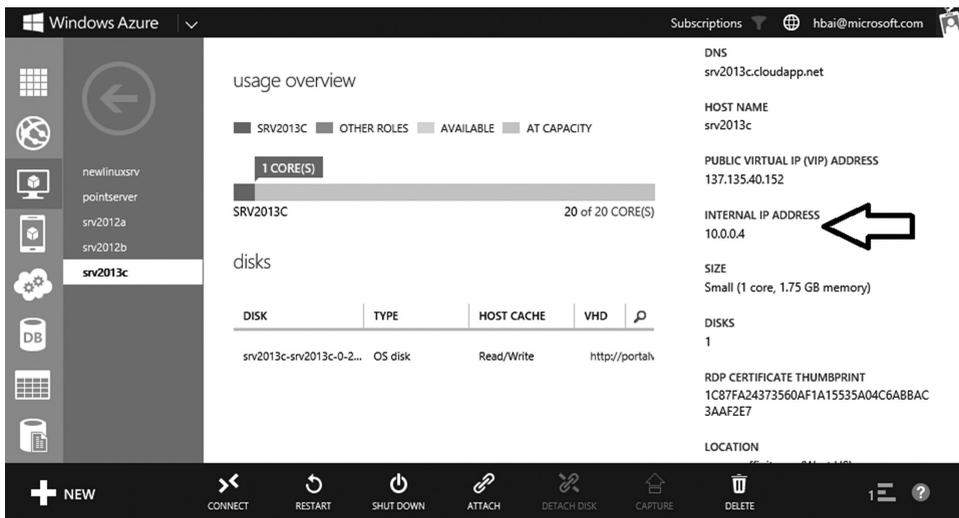


Figure 7.43 Internal IP address of a virtual machine.

Example 7.9: Point-to-site virtual network—share between local machine and cloud

Difficulty: ****

In this example, we will create a new virtual network and join a virtual machine to it. Then, we will configure a point-to-site virtual network and share files between your local machine and the virtual machine on the virtual network.

1. Log in to Microsoft Azure Management Portal.
2. On the command bar, click the **NEW** icon. Then, select **NETWORK SERVICES**→**VIRTUAL NETWORK**→**CUSTOM CREATE**.
3. On the **Virtual Network Details** page, enter *pointtosite* as the virtual network's name. Pick or create an affinity group, and click the next button to continue.
4. On the **DNS Servers and VPN Connectivity** page, check the **Configure point-to-site VPN** checkbox, and then click the next button to continue, as shown in Figure 7.44.
5. On **Point-to-Site Connectivity** dialog, click the next button to continue, as shown in Figure 7.45.
6. On **Virtual Network Address Spaces** dialog, click the **add gateway subnet** button, and then click the **check** button to complete the operation, as shown in Figure 7.46.
7. Create a new virtual machine. When you enter user credentials, be sure to enter the same user name and password as the credentials on your local machine (see Figure 7.21). This is to allow you to share folders and files between your local machine and your remote virtual machine on the virtual network. Join the new virtual machine to the earlier virtual network.
8. After the virtual network has been created, go to its **DASHBAORD** page, and click the **CREATE GATEWAY** icon on the command bar to create the dynamic gateway.
9. Now let us create the digital certificate for client authentication. Here we will use two self-signed certificates: one as the root certificate and the other for client authentication. We will need to upload the root certificate to Microsoft Azure. Launch **Developer Command Prompt for VS2012** as an administrator.
10. Go to the folder where you want to keep the certificate file. In this example, we will use the *c:\haishi* folder.

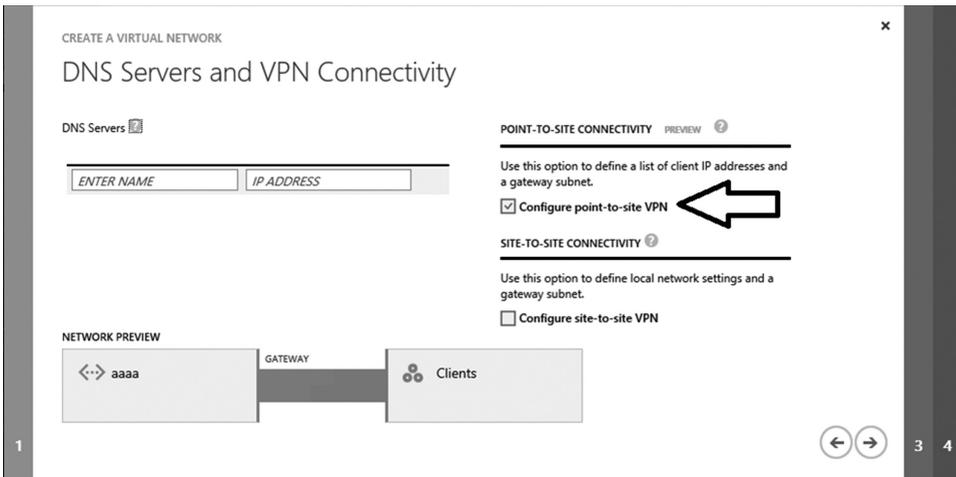


Figure 7.44 DNS Servers and VPN Connectivity dialog.

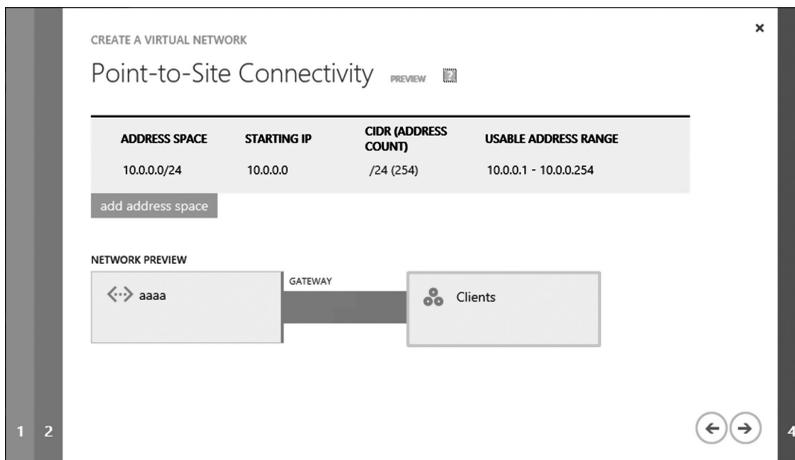


Figure 7.45 Point-to-site connection.

- Use the following command to create the root certificate:

```
Makecert -sky exchange -r -n "CN=MyFakeRoot" -pe -a sha1 -len 2048 -ss My
```

- Use the following command to create the client certificate:

```
makecert -n "CN=MyLaptop" -pe -sky exchange -m 96 -ss My -in "MyFakeRoot" -is my -a sha1
```

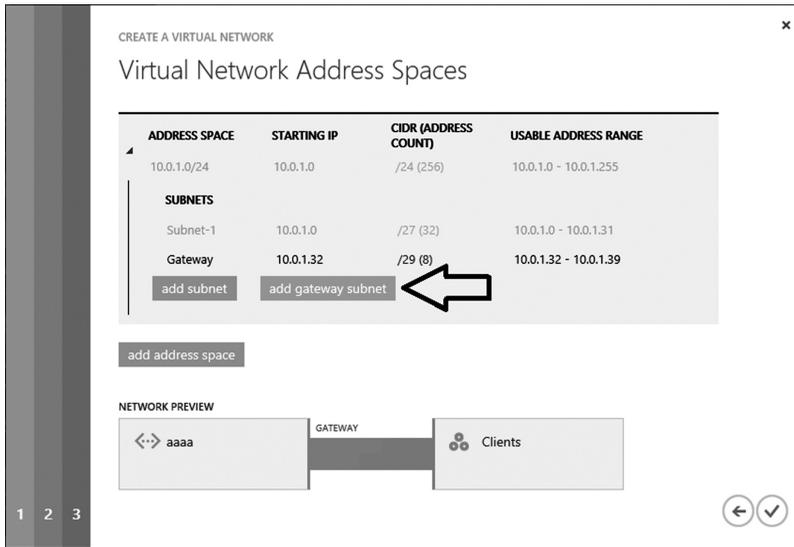


Figure 7.46 Adding gateway subnet.

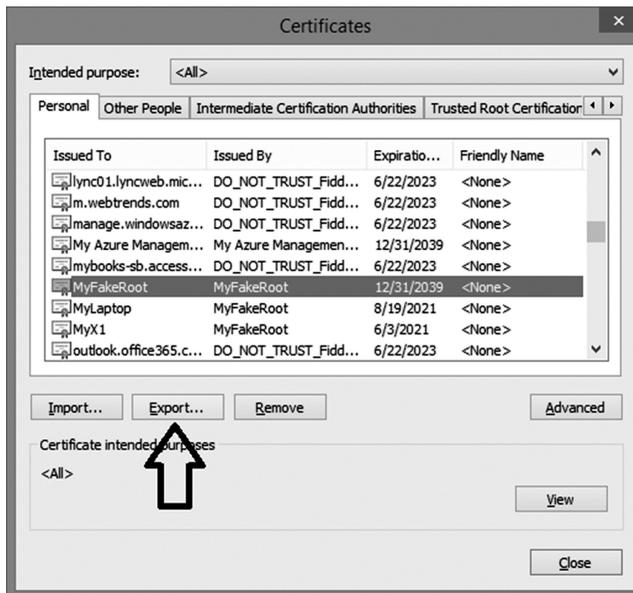


Figure 7.47 Export root certificate.

- On the command prompt window, type in `certmgr [enter]` to run `certmgr.exe`. Export `MyFakeRoot` to a `MyFakeRoot.cer` file (see Figure 7.47; note that you should not export the private key on the **Certificate Export Wizard**). Note that if you are configuring the connection for another client, you will need to install the client certificate (with private key) on the target machine.
- Back on Microsoft Azure Management Portal. On the virtual machine's **CERTIFICATES** page, click the **UPLOAD** icon on the command bar to upload the root certificate, as shown in Figure 7.48.

pointtosite

DASHBOARD CONFIGURE CERTIFICATES

A point-to-site VPN has been configured, but it is missing a root certificate. Upload a root certificate.

UPLOAD A ROOT CERTIFICATE →



Figure 7.48 Upload root certificate.

15. After the certificate has been uploaded, return to the virtual machine's **DASHBOARD** page. Click the **Download the 64-bit client VPN Package** link (for 64-bit client) or the **Download the 32-bit client VPN Package** link (for 32-bit client). Install the downloaded package. Because the package is not signed, you will need to ignore the security warning to finish installation.
16. After the VPN client has been installed, you can see the VPN network in the list of your Windows networks (see Figure 7.49). Click to connect to the network.

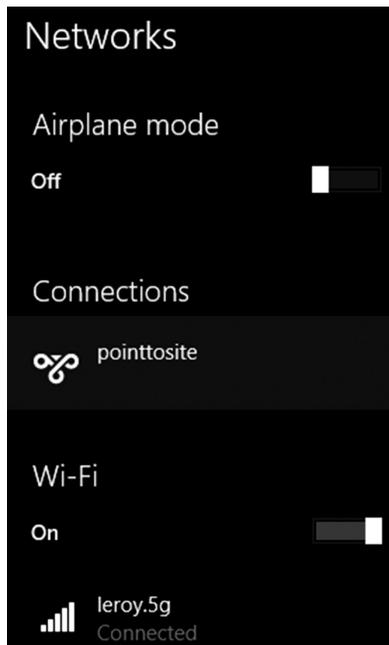


Figure 7.49 The VPN on Windows.

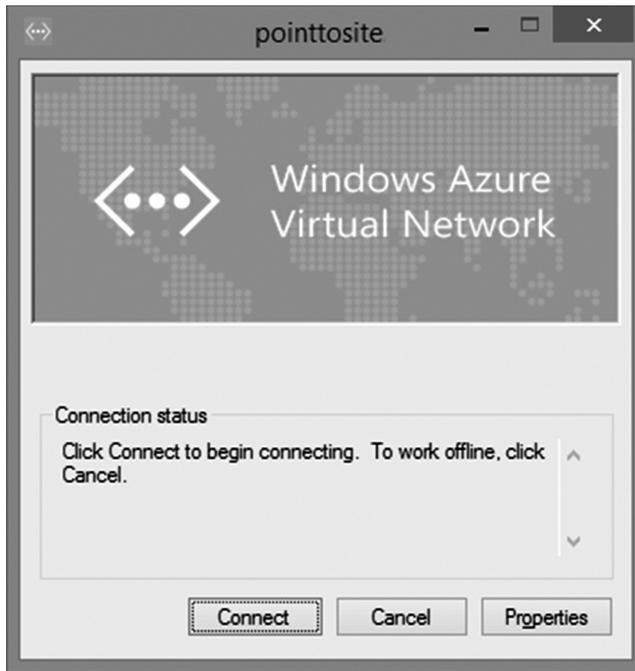


Figure 7.50 VPN client window.

17. On the VPN client window, click the **Connect** button to connect to the VPN, as shown in Figure 7.50.
18. Once the VPN is connected, you can use `ipconfig/all` to query connection information, as shown in Figure 7.51.
19. Log in to the virtual machine. Create a new folder named *Share* and share it with the current user.
20. On the virtual machine's **DASHBOARD** page, note down the machine's internal IP address, which in this case is 10.0.1.4.

```
C:\Users\hbai>ipconfig/all
PPP adapter pointtosite:

Connection-specific DNS Suffix . . . : 
Description . . . . . : pointtosite
Physical Address . . . . . : 
DHCP Enabled . . . . . : No
Autoconfiguration Enabled . . . . . : Yes
IPv4 Address . . . . . : 10.0.0.2(Preferred)
Subnet Mask . . . . . : 255.255.255.255
Default Gateway . . . . . : 
NetBIOS over Tcpip. . . . . : Enabled

Wireless LAN adapter Local Area Connection* 2:
Media State . . . . . : Media disconnected
```

Figure 7.51 Connection information by ipconfig.

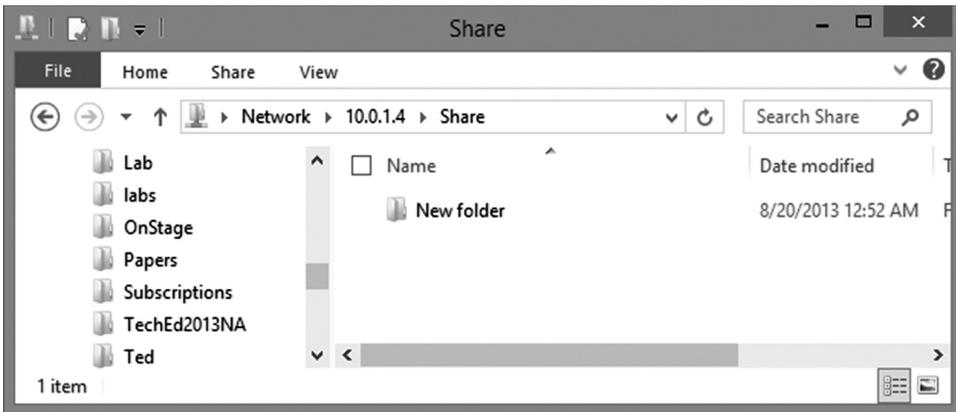


Figure 7.52 Use shared folder.

21. On the local machine, open File Explorer and open `\\10.0.1.4\Share` to access the shared folder. Figure 7.52 shows that I have created a new folder on the remote virtual machine under the shared folder.

7.4.3 Site-to-Site Virtual Network

In earlier versions of Microsoft Azure, a site-to-site connection required specific VPN hardware support from selected vendors. Now you can use the pure software virtual network solution from Microsoft Azure to create a site-to-site virtual network. Considering this is not a common activity carried out by developers, we will skip this part. Interested users may consult related MSDN documents.

7.4.4 ExpressRoute

Many components in a hybrid cloud solution are not originally designed to communicate via public Internet. As large enterprises implement hybrid solutions, they may face various problems dealing with such components in terms of security and throughputs, etc. Azure ExpressRoute allows these enterprises to establish dedicated connections between their data centers and Azure so that they can have faster, more reliable, and more secured connections among their on-premises resources and cloud resources. Interested readers can consult Microsoft Azure documentation at <http://azure.microsoft.com/en-us/services/expressroute/>.

7.5 Summary

In this chapter, we studied Microsoft Azure Virtual Machines as well as Microsoft Azure Virtual Networks. Through a series of examples, we learned detailed steps of creating both Windows-based and Linux-based virtual machines. We also discussed endpoints on virtual machines. We then configured a Node.js server on a virtual machine. We put two IIS servers behind a load balancer for load balancing as well as high availability. Finally, we introduced Microsoft Azure Virtual Network by creating a virtual network and a point-to-site VPN connection.

CLOUD SOLUTIONS



A cloud-based solution has to be designed for cloud before it can thrive on cloud. In this section of the book, we first examine some of the most popular architectures running on local data centers today. We discuss both opportunities and challenges you may face when you attempt to migrate these applications to cloud. Then, we discuss several key design aspects you need to consider when designing for cloud: high-availability, high-reliability, performance, and security. Cloud presents a different paradigm of how successful applications are designed, operated, and maintained. So, designing for cloud requires a different mindset that may take time to set in. This section of the book will cover some of the key mindset changes, such as scaling out and embracing errors, which can help you a long way in designing new cloud-based solutions.

Chapter 8

Cloud Solution Architecture

In this part of the book, we will discuss how to design cloud solutions. The so-called cloud solutions are solutions that leverage services and infrastructure supports provided by cloud platforms to solve practical problems. Cloud services are common building blocks of cloud solutions, but they are not all. A cloud solution consists of all related resources such as services, clients, virtual machines, external services, legacy systems, and their integration. In this chapter, we will discuss several common cloud solution architectures. Although these architectures share their names with those on on-premise systems, there are some significant differences between the two.

8.1 Client/Server

Client/server is a very common architecture. Under client/server architecture, the server governs and orchestrates all business processes. To ensure system performance and availability, the server is often deployed on a server cluster. Coordinated by the server, clients can share data and work together to carry out complex, distributed workflows. Clients are responsible for providing rich user experiences, operating local devices, and taking some compute and storage workloads off the server. The architecture diagram of a client/server system is depicted in Figure 8.1.

8.1.1 Characteristics of Client/Server Architecture

Because client/server has many favorable characteristics, it is a widely used architecture in spite of a few shortcomings.

8.1.1.1 Benefits

- Rich user experiences

Because client applications run on local machines, they can leverage local compute and storage resources to provide an optimized user experience. For example, client applications can use local GPUs to provide more dynamic, more expressive user interfaces. Clients can also access local devices such as cameras, printers, and various sensors. In addition, user

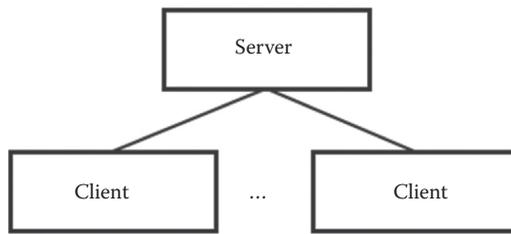


Figure 8.1 Client/server architecture.

interfaces built using client-native gadgets provide consistent, predictable user experience, allowing users to get started quickly. Predictability is a very important aspect for providing pleasant user experiences. For example, if a UI element looks like a button but behaves like an input box, it will surely cause confusions and complaints. To a certain extent, this is why all mobile application stores (such as Apple Store, Windows Phone Apps+Games Store, and Android stores) provide UI specifications and guidelines for developers to implement consistent UIs.

Without the constraints of network bandwidth and service throttling, client applications can have frequent interactions with users. For example, a client application can utilize all local CPU cores to handle complex tasks, such as image processing and data analysis, and provide immediate feedback as the tasks progress.

In addition, client applications can be deeply integrated with client environments, providing new ways of engagement, such as voice control, rich gestures, specialized input devices, and many other innovative ways of user interactions. Clients can be unattended as well. For example, a client application can be configured as a background service or a daemon that automatically responds to user actions or system events. Another example is that various plug-ins made by cloud storage providers allow users to upload/download files directly from Windows Explorer.

Modern browsers are constantly evolving. With features such as GPU-accelerated rendering, offline caches, and HTML5 supports, modern browsers can provide richer user experiences. Some argue that client applications will be obsolete. However, the unique capabilities of working with local devices and providing deep engagements are hard to be totally replaced by browsers, which, for security concerns, will always have certain constraints accessing local resources.

■ **Support offline mode**

One benefit of the client/server architecture is to allow users to use the system in both online mode and offline mode. When a client becomes offline, it can still provide the necessary features to keep the business going. For example, a client can still take orders when it is offline and submit the orders when the connection to the server is restored. For many mission-critical systems, even the slightest interruptions in workflow are unimaginable. A client that can provide continuous access to business functions regardless of network conditions is invaluable to many businesses.

Of course, supporting offline mode has intrinsic complexity. Client applications have to ensure the security of local data as well as the communications with the server. Moreover, because data and workflow states are scattered in multiple machines, states of these machines have to be synchronized, and conflicts have to be reconciled.

- **Stickiness**

The stickiness of a service refers to a user's tendency to keep using the same service instead of switching to competitors. Because client applications are installed on local systems, users naturally have a sense of ownership. Compared to a subscription relationship, the ownership relationship has a better bond with the user, helping to create a stable, long-term relationship. This sense of ownership is most obvious when it comes to mobile devices. Because mobile devices follow users into their real-life contexts, their personal attachment to users is unprecedented. Many successful social networks, media services, and content services provide free mobile clients in order to harvest the benefits of this close relationship between users and their devices.

8.1.1.2 Shortcomings

Because clients need to be installed on all machines that access the server, deployment and maintenance of client/server systems are complex. Configurations of client machines have endless permutations. It is impractical to guarantee a client software will work on all those machines. In addition, software provided by different vendors may conflict with each other, or have conflicting requirements. Upgrading all the clients is also a tedious job—clients may have to be reconfigured; different versions of a client may conflict; data might need to be migrated across client versions, etc. Some larger enterprises have to set dedicated resources just to keep all the systems up to date. All these complexities have been long criticized, and constant attempts have been made to alleviate the problem in the past decades. Techniques such as installers, scripts, automatic updates, and security policies are all targeted at streamlining client application management. At the same time, the complexity has also driven people to consider other architectural choices, such as browser/server and SaaS.

8.1.2 Client/Server Architecture on Cloud

Architecturally, client/server systems on cloud are quite similar to client/server systems on-premise. However, there are many hidden differences at the technical level, architecture level, and business level.

- **Differences at technical level**

At the technical level, the biggest difference is the running environment. On-premise client/server systems are deployed to local data centers. Servers and clients reside on the same private network, which provides secure, stable, and efficient connections among them. Under this kind of setting, clients and servers can afford frequent exchanges of large amounts of data. Systems that follow this communication pattern are sometimes called chatty systems. Cloud-based client/server systems live in a totally different environment, where clients and servers communicate over the open Internet. This environment raises additional requirements for secure data transfer, simplified communication patterns, and reduced network utilization. In addition, server resources on cloud are shared by all tenants. Cloud platforms impose strict constraints on resource utilization to ensure fair usage of its resource pools. Because you are charged for the amount of resources you use, inefficient system designs have direct financial consequences. So, system efficiency is very important for cloud-based client/server systems to avoid throttling and unnecessary costs.

■ Differences at architecture level

Cloud-based client/server systems often have different deployment topology with on-premise client/server systems. Traditionally, many client/server systems use dedicated servers for different customers. In other words, a server instance is only responsible for requests from a designated customer. This kind of architecture is also called single-tenant architecture. Although cloud-based client/server systems can use single-tenant architecture, a more efficient choice is to use multitenant architecture. Under multitenant architecture, a cluster of servers share the responsibility of handling requests from all customers so that the server resources can be utilized more efficiently. For example, under a single-tenant architecture, a service provider deploys three dedicated servers for three different customers. Among the three customers, one customer generates significantly higher workloads, making his server extremely busy. On the contrary, the other two servers remain idle because of the low activities of their corresponding customers. Under a multitenant architecture, because the three servers share the workload from all customers, you can use the available resources on all three servers to satisfy the needs of the busy customer. Multitenancy is radically different from single-tenancy. We will discuss multitenancy in more detail in the next section.

■ Differences at business level

Although this book focuses on technologies, it is worth devoting several paragraphs to differences at the business level. Understanding the differences is crucial in deciding a company's, especially an ISV's, cloud strategy, not to overlook its importance in determining scopes and goals of cloud projects.

First, an ISV needs to understand the difference between the licensing mode and the subscription mode. In the traditional licensing mode, a software customer is both a service provider and a service consumer. The customer is responsible for managing the hosting infrastructure to keep the service running. Major maintenance, such as system-level upgrades, is often delegated to the ISV by maintenance contracts. Deploying a large-scale system requires significant investment in both time and money. This serious commitment creates a tight relationship between the ISV and the customer. Some ISVs also provide deep customization services to tailor the service to satisfy the customer's particular needs, which leads to an even closer relationship with the customer. The subscription mode is very different. A customer who subscribes to a service is just a service consumer. Without the need to invest time and money to build up the supporting infrastructure, a cloud service customer can implement a new system quickly without significant upfront investments. What often happens is that a potential customer will subscribe to a trial version and evaluate the service without any involvement of the ISV. On the one hand, the subscription mode increases the risk of a customer to pick a new service. On the other hand, the relationship between a customer and an ISV is not as close as in the traditional licensing mode. An ISV has to ensure that the service design is intuitive enough so that a potential customer can navigate through the system without much assistance. And the ISV has to figure out ways to continuously engage with the customer to keep an active relationship.

Second, the cost of implementing a cloud-based system is distributed to the whole lifespan of a subscription. This means that it takes a much longer time for an ISV to realize the whole sales amount. If the ISV's sales capability has not changed, the same number of deals will result in a much smaller income of the year. Moreover, because of the looser relationships with the customers, the risk of losing a customer is higher as well. Both these factors create problems in cash flow, sales force morale, customer relationships, and virtually all areas of the company. Although we have lived through the actual cases of how ISVs succeed

or fail to make such leap, we are not experts in making the business model switch from a low-volume–high-margin business to a low-margin–high-volume business. So we shall not discuss this topic any further. The only point we are raising is that readers should be aware that such challenges exist not only at the technical level but also at the business level when the switch is made to subscription mode. The challenges should not be taken lightly because they may generate serious consequences.

8.1.3 Multitenant System Design

A multitenant system uses a shared resource pool to satisfy the needs of multiple tenants. In the traditional dedicated instances (also called multi-instance or single-tenant) design, each customer has dedicated resources. Workflows and data of different customers are physically isolated from each other. On the contrary, on a multitenant system, customer workflows and data are not physically separated, but reside on shared resources with only virtual segmentations among them. Multitenancy is a common characteristic of many cloud services. Cloud platforms such as Microsoft Azure are massive multitenant systems.

■ Cost advantages

One of the major benefits of multitenancy is reduced hosting cost. A multitenant system can dynamically allocate resources for different customers as their workloads change so that resource utilization can be optimized. In a traditional data center, a server utilization rate at 25% is already considered outstanding performance. With virtualization, server utilization rate may grow up to 50%–60%. On a cloud platform, a healthy multitenant system can reach over 70% of server utilization rate. With higher utilization rates, the same number of server resources can yield more outcomes, hence reducing the overall hosting cost.

In addition, modern data centers, such as the ones used by Microsoft Azure, often have better Power Usage Effectiveness (PUE) and lower carbon dioxide emissions. Such benchmarks and their measurements are out of the scope of this book (except for the following little note).

Note: About Power Usage Effectiveness (PUE)

PUE can be calculated using the following formula:

$$\text{PUE} = \frac{\text{Total facility power}}{\text{IT equipment power}}$$

Theoretically, the best PUE is 1, which means that all power used by a data center is consumed by the IT equipment. Obviously, it is impossible to reach this best value because a data center will always consume power for other purposes such as office lighting and elevator operations. Microsoft Azure data centers have the best PUEs in the industry, with some centers reaching PUEs as low as 1.15.

In terms of data storage, a multitenant system can use fewer resources to achieve higher service levels. For example, let us assume 100 customers have subscribed for 1G of storage space each. Instead of allocating 100G storage space, a multitenant system can allocate only 50G of space to satisfy all customers with an average utilization rate of 50%.

- Manageability advantages

Multitenant systems have unique manageability benefits as well. Because all customers share the same system, upgrading the system will upgrade all customers at the same time. This not only simplifies the deployment process, but also avoids the complexity of supporting multiple versions at the same time. Of course, the clients of a client/server still need to be individually updated.

- Opportunities for data mining and business intelligence

Centralized customer data provides tremendous opportunities for data mining. With consent from the customers, a service provider can aggregate and analyze data from multiple customers. For instance, a service provider analyzes service usage patterns across its customer base to determine areas to be improved. Getting consent from the customers is a prerequisite. In most cases, customers want their privacy protected, including how they are using the service. A service provider should fully honor user privacy and never make inappropriate use of customer data. A major obstacle of cloud adoption is lack of customer confidence in handing over critical data to a third party. Improper use of customer data without consent not only violates privacy, but also undermines customers' confidence in cloud services.

Multitenant systems bring significant benefits, but they come with obvious shortcomings as well. Complexity is one of the biggest disadvantages of multitenant systems. When designing a multitenant system, a service provider has to consider the following:

- Tenant isolation

Tenant isolation is not limited to isolation of customer data. It also means isolation of access rights and workflow states. For example, on an on-premise system, the widely used role-based security can satisfy most of the access control requirements. However, on a multitenant system, user roles are no longer sufficient—any roles from one tenant should not have any access to another tenant. At this point, there are not many good tenant-based access control mechanisms other than claim-based authentication, which we will discuss in Chapter 12.

- Problem containment and resource throttling

For single-tenant systems, a crashing server only affects one customer that it serves. However, on a multitenant system, a server-side error may impact the whole customer base. This is obviously a bigger risk that has to be properly managed so that it does not generate systemwide failures. In the following chapters, we will discuss several reliability concepts and techniques, such as redundancy, failover, and autorecovery. In addition, multitenant systems need to closely monitor resource usages to avoid a busy customer blocking the whole system. This raises additional requirements in system design, such as resource monitoring and request throttling. The increased system complexity in turn increases service design, development, and maintenance costs.

- Management disadvantages

The management benefits I just mentioned can turn into disadvantages in some circumstances. Not all customers are ready to accept the latest and greatest versions at all times. Some customers would rather stay with a stable version that has been proven to work and do not want to change their established workflows. Backward compatibility is usually an effective way to address such concerns.

In summary, multitenant systems bring many benefits, but they also involve great complexity. A successful multitenant system can be very efficient and cost-saving, creating vast competitive advantages. On the other hand, readers should fully realize the complexity of multitenant systems and plan accordingly.

8.1.4 Migrating Client/Server Systems to Cloud

The most straightforward way to migrate an existing client/server application to cloud is to use Microsoft Azure virtual machines to provision dedicated servers for customers. Such a migration strategy does not impact system architecture and hence has lower risks. However, such migration does not take full advantage of the high scalability and the high availability of cloud. Especially when the number of customers increases, because each customer has dedicated servers that need to be individually maintained and updated, the cost of operation increases rapidly. In order to take full advantage of cloud, the system architecture will have to be changed to include multitenant support, which requires significant investment. Of course, once multitenancy is achieved, getting a new customer onboard will become easier and faster, which helps in reducing sales cycles as well as operational costs. In the long run, a multitenant system will help the service provider to establish a competitive edge and to gain the agility for rapid market expansion.

A practical and effective approach to migrate to cloud is to divide and conquer. For example, before migrating the whole system to cloud, you can pick among the rich set of SaaS services provided by Microsoft Azure and start to leverage them to improve availability, reliability, and scalability of these components. Such incremental improvements avoid the risk of massive migration, and allow you to accumulate knowledge and experience for final migration as you enjoy the benefits of reducing operational costs throughout the process. As an additional bonus, this approach also helps you to modularize your system to improve flexibility and maintainability.

There are no fixed rules for multitenant system design. Although an increasing number of systems are adding multitenancy support, there are no general, leading solutions in the industry. Theoretically, multitenancy is not complex. However, as people say, “the devil is in the details.” Before embarking on a multitenant project, you should fully realize the potential difficulties to avoid overly optimistic project plans.

8.1.5 Client/Server Systems on Microsoft Azure

Regardless of the differences we just talked about, you can use your existing knowledge of client/server systems on Microsoft Azure. For example, your service can be one of the following:

- A WCF service running on a Web Role or a Worker Role
- REST API running on a Web Role or a Microsoft Azure website
- A legacy service running on a Worker Role
- A legacy service running on a Microsoft Azure virtual machine

On the client side, you can still use existing tools and languages, such as Windows Form Applications, XAML, mobile clients (iOS, Android, Windows Phone, Surface, etc.), and even Console Application.

As we previously discussed, because clients and servers communicate through open Internet, client/server systems have additional requirements in system design. Here are some of the important aspects to pay attention to:

- **Information security.** Developers have to ensure that communication channels between the servers and the clients are secured at transportation level, message level, or both. Common techniques include HTTPS, message signing, and encryption. In addition, the risk of being attacked over the Internet is greater than on local networks. Developers should perform attack modeling and guard against possible attacks such as playback, eavesdrop, man-in-the-middle, code injection, and DoS, just to name a few.
- **Client/server interactions.** On a local network, frequent exchange of large amounts of data is not always a problem, but could be a favorable design for faster performance and richer user experience. However, the communication pattern will not only cause performance problems, but also generate unnecessary costs for egress data. During system design, developers should pay attention to avoid chatty API design and reduce network traffic by techniques such as client-side caching.
- **User authentication.** Many programs running on local data centers can leverage Windows Active Directory for authentication and authorization. However, services hosted on cloud cannot directly access these directories. We will discuss user authentication and authorization on cloud in Chapter 12.

8.1.6 Mobile Clients

With the increasing popularity of smart phones and tablets, more client software has been migrated to various mobile devices. Mobile clients bring both new opportunities and new challenges to client/server system architects and developers. We will discuss integration between cloud and devices in Section III of this book.

8.2 Browser/Server

The complexity of managing a large number of clients makes people search for alternatives. The Internet provides the infrastructure, SaaS provides the contents, and JavaScript and HTML enhancements provide the technology needed for browser/server applications to take off. The architecture diagram of a browser/server application looks quite similar to that of a client/server application, as shown in Figure 8.2. During the early stages of browser/server development, the clients running in a browser had limited functionalities because of technology constraints. This is why

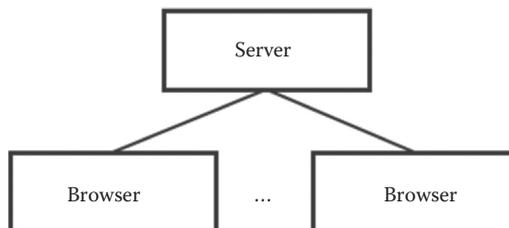


Figure 8.2 Browser/server architecture.

browser clients were commonly called “thin clients,” in contrast to the traditional “rich clients” or “fat clients” that can provide richer user experience. However, with the development of browser and scripting technologies, modern browsers are capable of supporting very rich feature sets nowadays. The differences between so-called “thin client” and “fat client” are not that obvious anymore.

8.2.1 Characteristics of Browser/Server Architecture

- Advantage: ease of management

One of the major benefits of browser/server architecture is ease of management. If you have ever participated in releasing client software, you may have come across situations where a client simply would not install, or would not function properly on some particular machines without obvious reasons. If you were lucky, you would spend considerable time and effort in creating, testing, and managing installable packages. Browser/server architecture eliminates the need to manage a large number of clients. In addition, because the architecture allows service users to always use the latest versions, it helps service providers to avoid the complexity of managing multiple versions of clients.

- Advantage: market penetration

Because a user can easily try out a service without installing any client software, it is easier for a service provider to attract potential users to adopt a new service by providing a friction-free experience. Of course, just as easily as a user would try out a service, it is easy for a user to forget about the service. So, the friction-free experience is a double-edged sword. It increases the opportunity for a service provider to present a new service to a wide audience, but it requires the service provider to be able to lead the user from casual browsing to a serious commitment during the first few tries. This requires the service provider to provide not only the required features, but also an intuitive and responsive experience.

- Advantage: cross platform

Most browsers support standard HTML, CSS, and JavaScript. The combination of the three essentially creates an abstraction layer that hides the details of underlying operation systems. A web page can run on this sandbox environment regardless of whether the hosting environment is Windows, iOS, or Linux. Many websites also take advantage of the HTML support on mobile devices to share a common client UI across different smartphones and tablets.

- Challenge: cross browser

A cross platform is nice, but a cross browser is, at this point of time, not that easy to achieve. Different browsers always behave differently, with some of the differences being intentionally introduced by browser vendors to gain advantage over competitors. Several techniques, such as Java Applet, Adobe Flash, and Microsoft Silverlight have been developed to create true cross-browser experience. But the additional installation often defeats the frictionless goal. Working with cross-browser HTML, CSS, and JavaScript is also a major headache. With the development of HTML5 standard and improvement in cross-browser tools such as jQuery and Visual Studio CSS editor, the situation has improved greatly over the past few years. However, a true cross-browser client is still hard to make.

- Challenge: other constraints

For security reasons, in-browser JavaScripts cannot access local resources and devices freely. For mobile devices, being able to interact with local sensors such as GPS, accelerometer, and gyroscope is often very important for building highly interactive, rich user experiences. In addition, generic offline support is just emerging with HTML5 standard. In many cases, you may have to fall back on native clients in order to build the exact applications you want.

8.2.2 Browser/Server Architecture on Cloud

Most cloud services with web front end can be considered browser/server systems. However, we can further categorize them into two broad types. Commonly, we refer to services built on top of PaaS as “native” cloud services, and services that are hosted on cloud using IaaS as cloud-hosted services.

In the previous section, we talked about multitenant design, which can be applied to both client/server and browser/server systems. It is not unusual for native cloud services to adopt a multitenant design in order to take full advantage of the reliability, scalability, and availability features provided by PaaS. In this section, we will focus on one aspect of a multitenant browser/server system—resource allocation among tenants. The key benefit of multitenancy is to allow a service provider to use a shared resource pool to serve many tenants. A service provider needs to ensure fair resource allocation among tenants so that a few busy tenants do not take over all the resources.

Two main mechanisms of managing resource allocation are quota and throttling. A quota system imposes upper limits on the amount of resources each tenant can utilize. Usually, a service provider sets up different subscription levels for customers to choose from, and each subscription level grants a certain amount of storage and compute resources that a customer can consume. Throttling, on the other hand, monitors resource usages and delays requests from customers who have exceeded certain limits. Note that when throttling happens, the requests from offending customers should not be rejected, but be queued up for later handling. This is because rejecting requests will affect system availability. Throttling is often harder to implement than a quota system, because throttling requires dynamic resource monitoring and allocation. Fortunately, Microsoft Azure cloud service supports multisite configurations. When combined with IIS throttling, the feature can help a cloud service provider to throttle user requests based on different subscription levels. Now let us see how it is done.

Example 8.1: Multisite Cloud Service and throttling by tenant

Difficulty: *****

Microsoft Azure Web Role supports multisite configurations, allowing a single Web Role to host multiple websites. In addition, IIS 8 supports throttling based on CPU usages. In this example, we combine these two features to achieve throttling by tenants. We create a new cloud service with a single Web Role, on which we define two websites: one for important customers and one for common customers. For important customers, we allow as much as 40% of server CPU usage, while for common users, we allow up to 30% of CPU usage.

Note: In this example, we use a local development machine. Running this example requires Windows 8 or Windows Server 2012.

1. First, we need to ensure that IIS 8 and ASP.NET 4.5 have been enabled. You can verify the setting via **Control Panel**→**Programs and Features**→**Turn Windows features on or off**, as shown in Figure 8.3.
2. For testing purpose, we will need to modify the *hosts* file of the Windows system. Launch **Notepad** as an administrator. Open the `c:\Windows\System32\Drivers\etc\hosts` file (assuming your Windows is installed in the default folder). Append these two lines to the file and save the file.

```
127.0.0.1    gold.haishi.com
127.0.0.1    silver.haishi.com
```

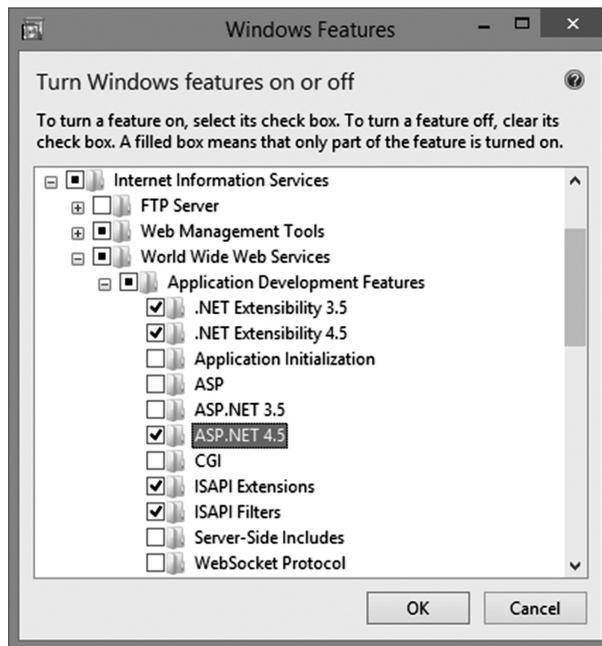


Figure 8.3 Verify if IIS features have been installed.

3. Launch Visual Studio as an administrator. Create a new cloud service with an ASP.NET MVC 4 Web Role (use an Internet Application template).
4. First, check **ServiceConfiguration.Cloud.cscfg** and **ServiceConfiguration.Local.cscfg** to ensure that the **osFamily** attribute on the root element is set to **3**, which means using Windows Server 2012 as the hosting operation system. On the virtual machines provided by Microsoft Azure, Windows Server 2012 servers use IIS 8, which is the version we need in this example.
5. Edit **ServiceDefinition.csdef** as shown in Code List 8.1. Lines 2–8 and lines 9–14 define two websites. Although they point to the same website code (`..\..\MvcWebRole1`), they have defined different host headers (*hostHeader*) to distinguish from each other.
6. Modify the **Index()** method of **Controller\HomeController.cs**. In Code List 8.2, line 3 puts the host header to *ViewBag* for display. Lines 4–7 set a different theme color based on the header text—we will make use of these values in the next step.
7. Modify **Views\Home\Index.cshtml** to add a **<div>** to display different background color based on *ViewBag.ThemeColor* we put in Code List 8.2. The modified code is as shown in Code List 8.3.
8. Press F5 to launch the application. You will get a 400 error, which is expected. Modify the address in the browser address bar to **http://silver.haishi.com:81** (your port might be different. When you modify the address, just change 127.0.0.1 to silver.haishi.com, keep the port).
9. Now you will see that the header information is displayed on a gray background, as shown in Figure 8.4.
10. You can also access the other site by the address **http://gold.haishi.com:81/**. Close the browser to terminate the application when you are done.
11. Under the **Controllers** folder, add a new empty MVC controller named **CPUThrottleController.cs**. Add a **HeavyLoad** method, which uses a tight loop to simulate a heavy load. Note that at line 11 in Code List 8.4 we specified the loop count to 4 to keep all CPU cores busy because we are using a 4-core system.

CODE LIST 8.1 DEFINE MULTIPLE SITES ON A WEB ROLE

```

1:<WebRole name="MvcWebRole1" vmSize="Small">
2:  <Sites>
3:    <Site name="gold" physicalDirectory="..\..\..\MvcWebRole1">
4:      <Bindings>
5:        <Binding name="Endpoint1" endpointName="Endpoint1"
6:          hostHeader="gold.haishi.com"/>
7:      </Bindings>
8:    </Site>
9:    <Site name="silver" physicalDirectory="..\..\..\MvcWebRole1">
10:     <Bindings>
11:       <Binding name="Endpoint1" endpointName="Endpoint1"
12:         hostHeader="silver.haishi.com"/>
13:     </Bindings>
14:   </Site>
15: </Sites>
16: <Endpoints>
17:   <InputEndpoint name="Endpoint1" protocol="http" port="80" />
18: </Endpoints>
19: <Imports>
20:   <Import moduleName="Diagnostics" />
21: </Imports>
22:</WebRole>

```

CODE LIST 8.2 USE REQUEST HEADER IN INDEX METHOD

```

1:public ActionResult Index()
2:{
3:    ViewBag.Message = Request.Headers["Host"];
4:    if (Request.Headers["Host"].Contains("gold"))
5:        ViewBag.ThemeColor = "gold";
6:    else
7:        ViewBag.ThemeColor = "silver";
8:    return View();
9:}

```

12. Create a **CPUThrottle** folder under the **Views** folder; then add a new **Index.html** beneath it:

```

@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<div class="killerback">
    <div class="killer">
        @Html.ActionLink("Kill CPU", "HeavyLoad")
    </div>
</div>

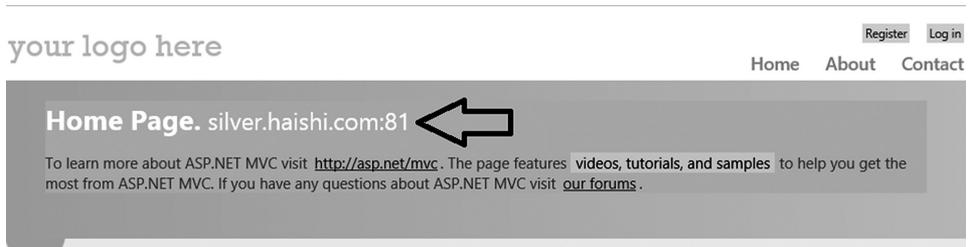
```

CODE LIST 8.3 ADDED DIV ON THE INDEX VIEW

```

@{
    ViewBag.Title = "Home Page";
}
@section featured {
    <section class="featured">
        <div class="content-wrapper">
            <div style="background-color:@ViewBag.
                ThemeColor">
                <hgroup class="title">
                    ...
                </div>
            ...
        </div>
    </section>
}

```

**Figure 8.4** Header information displayed on page.

13. Modify the **Site.css** file under the **Content** folder to add the following two styles:

```

.killer {
    background-color: #FFFF00;
    border: 20px dashed #000000;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 48px;
    font-weight: bold;
    padding: 10px;
    margin: 10px;
    text-align:center;
}
.killerback {
    background-color: #FFFF00;
}

```

CODE LIST 8.4 CPUTHROTTLECONTROLLER

```

1:using System.Threading;
2:...
3:public class CPUThrottleController : Controller
4:{
5:    public ActionResult Index()
6:    {
7:        return View();
8:    }
9:    public ActionResult HeavyLoad()
10:   {
11:       int count = 4;
12:       for (int i = 0; i < count; i++)
13:       {
14:           ThreadPool.QueueUserWorkItem((obj) =>
15:           {
16:               Random rand = new Random();
17:               while (true) { rand.Next(); }
18:           });
19:       }
20:       return new HttpStatusCodeResult(200);
21:   }
22:}

```

14. Finally, modify the `Shared_Layout.cshtml` file to change the Contact *ActionLink* to “CPU Pressure Test”:

```

<ul id="menu">
  <li>@Html.ActionLink("Home", "Index", "Home")</li>
  <li>@Html.ActionLink("About", "About", "Home")</li>
  <li>@Html.ActionLink("CPU Pressure Test", "Index",
    "CPUThrottle")</li>
</ul>

```

15. Run the application again. Click on the **CPU Pressure Test** link, and then click on the **Kill CPU** link, as shown in Figure 8.5.
16. On **Task Manager**, you can observe that the CPU is 100% busy, as shown in Figure 8.6.
17. If you open **Resource Monitor**, you will observe that the tight loop can cause a certain amount of damage. You can see that all 4 cores on the system are 100% occupied, as shown in Figure 8.7.
18. Stop the application. In the next step, we will use the IIS 8 throttling feature to ensure a heavy load from a single tenant does not bring down the whole system.

Note: IIS Features for Multitenancy

On IIS 8, every application pool runs in its own sandbox environment. These sandbox environments run under different user identifiers, which makes it possible to limit the maximum resource (such as CPU) the application pool can use. The design can be traced

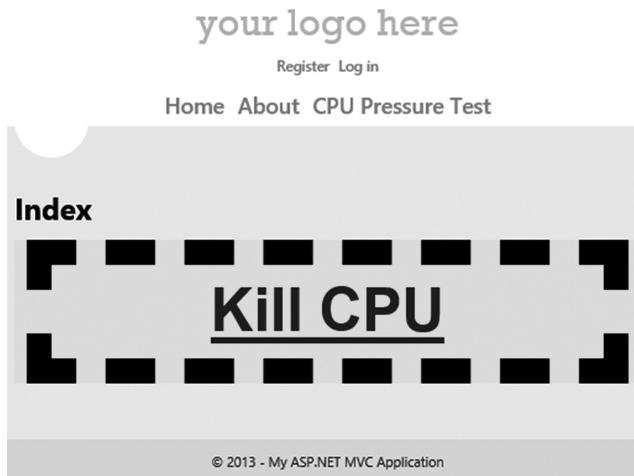


Figure 8.5 Kill CPU button.

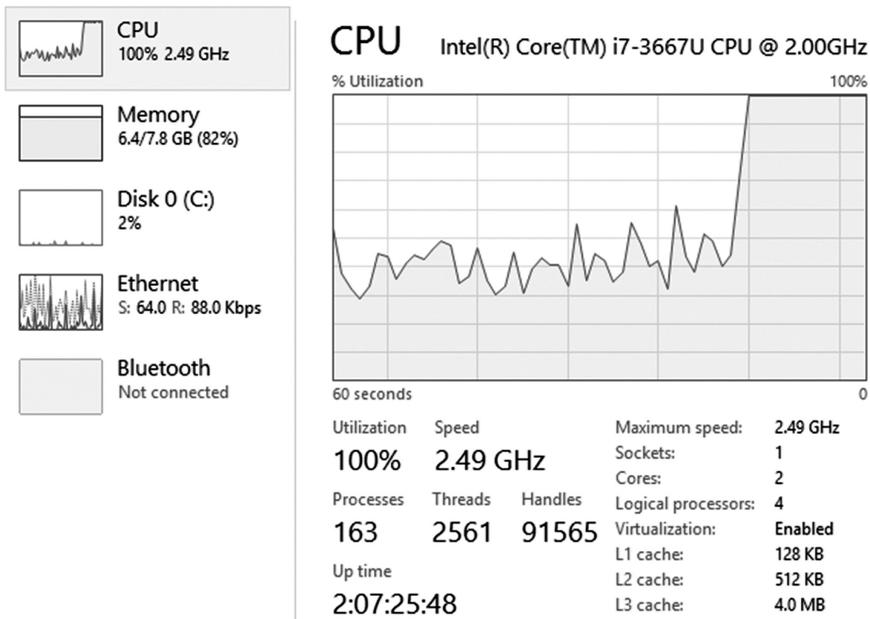


Figure 8.6 CPU being saturated.

back to IIS 7. However, on IIS 7, when a tenant exceeds its limit, the corresponding application will be recycled, causing service interruptions. On IIS 8, the requests from the offending tenant will be postponed, but the application will not be recycled. In addition, you can configure IIS 8 so that it allows an application to take more resources than allocated when the system is not busy, but only throttles resource usage when the system is under stress.

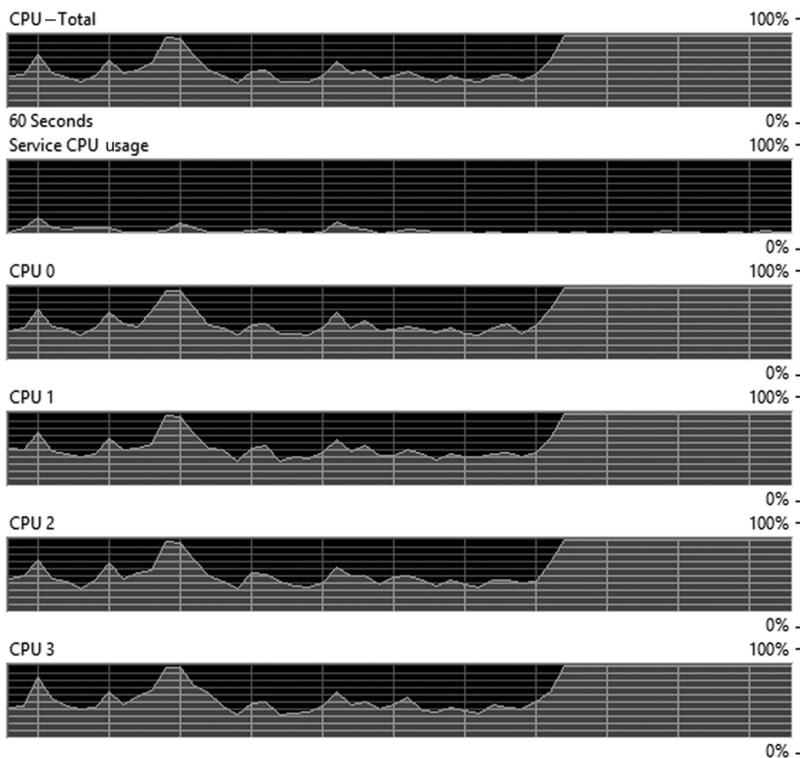


Figure 8.7 CPU usage charts in Resource Manager.

Those interested in finding out more on this topic may refer to

<http://www.iis.net/learn/get-started/whats-new-in-iis-8/iis-80-cpu-throttling-sandboxing-sites-and-applications>.

IIS 8 also provides other features to support multitenancy. For instance, IIS 8 supports high website density, allowing you to deploy thousands of websites on a server. It also supports centralized certificate management, which greatly simplifies certificate distribution and renewal. In addition, IIS 8 also supports dynamic throttling per IP address based on traffic generated from the IP.

19. On the Web Role project, add a reference to `assembly\system32\inetsrv\Microsoft.WebAdministration.dll`. Then, set its **Copy Local** property to `True`, as shown in Figure 8.8.

Note: Setting a reference assembly's `Copy Local` property to `True` ensures the assemblies will be deployed to Microsoft Azure when the cloud service is deployed. If your project uses assemblies only in your local GAC or other third-party assemblies without packaging them with your cloud service package, your cloud service will fail to start. Of course, the hosting virtual machines provided by Microsoft Azure already include .Net and Microsoft Azure runtime assemblies.

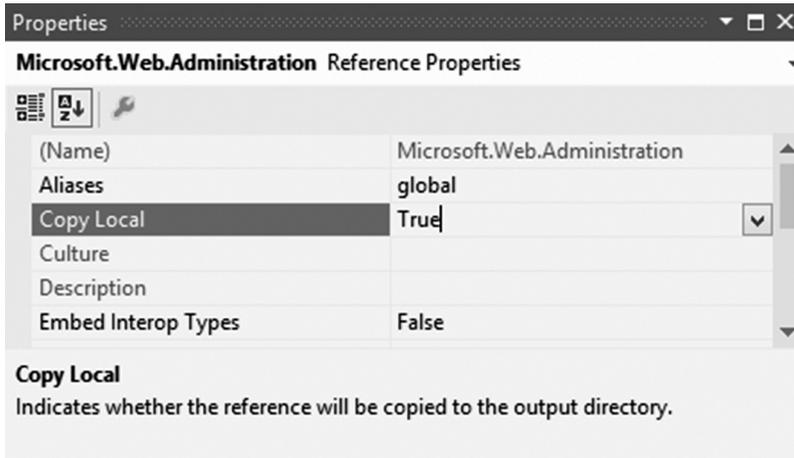


Figure 8.8 Set assembly Copy Local property to True.

20. Next, we will deploy our Web Role to IIS 8 instead of the default IIS Express. In **Solution Explorer**, right click the Web Role project and select the **Properties** menu to open its properties page. On the **Web** tab, uncheck **Use IIS Express**. Then, press Ctrl + S to save the file. If the system prompts to create a virtual directory, click the OK button to continue, as shown in Figure 8.9.
21. Replace the code for **WebRole** class in **WebRole.cs** with the code in Code List 8.5. This code defines a customer throttling policy (lines 27–31). Then, when the role is started, two instances of the policy are created. One is for “gold” customers, who are allowed to use as much as 40% of the CPU, and the other is for “silver” customers, who can use as much as 30% of CPU. Finally, lines 14–19 find the corresponding application pool and change the application pool settings.
22. Launch the application again. You can open both <http://gold.haishi.com:81/> and <http://silver.haishi.com:81/> and click on the Kill CPU buttons. Because of the CPU throttling protection, the CPU usage is kept at a safe level regardless of the eight tight loops we run, as shown in Figure 8.10.

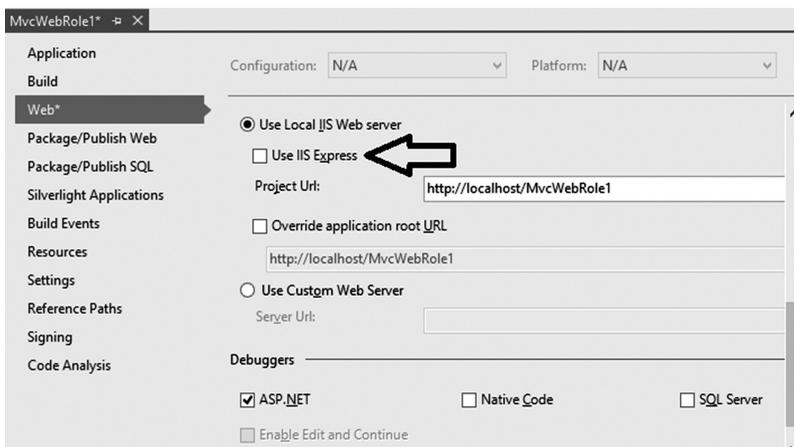


Figure 8.9 Configure the Web Role to use full IIS.

CODE LIST 8.5 WEB ROLE IMPLEMENTATION

```

1:public class WebRole : RoleEntryPoint
2:{
3:    public override bool OnStart()
4:    {
5:        CustomerProfile[] profiles =
6:            { new CustomerProfile{Name ="gold", CPUThrottle =
7:                40000},
8:              new CustomerProfile{Name ="silver", CPUThrottle =
9:                30000}};
10:        using (ServerManager serverManager = new ServerManager())
11:        {
12:            var applicationPools =
13:                serverManager.ApplicationPools;
14:            foreach (var profile in profiles)
15:            {
16:                var appPoolName =
17:                    serverManager.Sites
18:                    [RoleEnvironment.CurrentRoleInstance.Id
19:                    + "_" + profile.Name]
20:                    .Applications.First().ApplicationPoolName;
21:                var appPool = applicationPools[appPoolName];
22:                appPool.Cpu.Limit = profile.CPUThrottle;
23:                appPool.Cpu.Action = ProcessorAction.Throttle;
24:            }
25:            serverManager.CommitChanges();
26:        }
27:        return base.OnStart();
28:    }
29:    private struct CustomerProfile
30:    {
31:        public string Name;
32:        public int CPUThrottle;
33:    }
34:}

```

8.2.3 Difficulties of Adapting an Existing Single-Tenant Browser/Server Application for Multitenancy

Although browser/server architecture fits the cloud environment pretty well, migrating existing browser/server services to cloud might be problematic, especially if the browser/server service did not take network delays into consideration, relied on local services and devices, and was not designed for multitenancy. Obviously, you can start over by writing a new multitenant service. However, you would most likely like to upgrade an existing single-tenant service to a multitenant service. In this section, we will discuss some of the major difficulties of making such changes. In the next section, we will introduce a strategy that allows you to take a single-tenant service and host it on Azure as if it were a multitenant service.

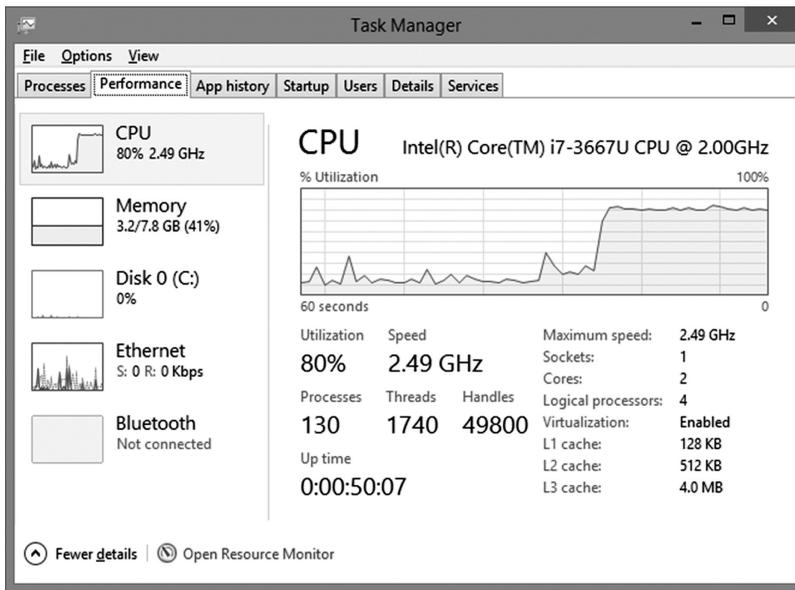


Figure 8.10 CPU usage is being throttled.

In the following paragraphs, we will discuss the main difficulties of achieving multitenancy at the presentation layer, business layer, and data layer.

■ Presentation layer

At the presentation layer, a big challenge is customization. Under a single-tenant architecture, customizations are often static and sometimes hardcoded. This is because each deployment of the system only serves for a single tenant. So the service does not need the complexity of defining and loading customization settings dynamically. On the contrary, a multitenant system needs to serve multiple tenants at the same time, so it has to allow dynamic customizations based on user contexts. Designing and implanting a flexible and yet maintainable dynamic customization system is not easy. To illustrate with a simple example, let us assume you want to create a simple Twitter reader site, which allows you to display latest tweets from multiple accounts. Each user should be able to customize the layout of the page, as well as to control if tweet pictures should be embedded. Figure 8.11 shows a mock-up of such a UI.

Now the UI does not look complex at all. What if you want the UI to be fully customizable? You will probably need artifacts like the following:

- Mechanism of storing customization settings
- Mechanism to apply these settings dynamically
- UI for users to change these settings

In addition, for better user experience, you may want to have a template gallery, drag-and-drop editing, live preview, and tenant-level templates. All these are just for the simplest case of customization. In some systems, different customers may want to have different data entry forms and different workflows, which are even more complicated.

Tweets from Friends



Figure 8.11 Fictional Twitter reader UI.

■ Business layer

At the business layer, the biggest challenge is state management, which presents two levels of problem.

First, a single-tenant system, especially a system that is not designed to be scaled out, often stores a large number of temporal states in its local storage or in memory. All user operations have to be carried out on a single server, because only the server holds all the necessary states. An obvious example is a shopping cart. If the state of a shopping cart is saved in memory, a user has to return to the saved server whenever he or she attempts to update the cart. Temporal states such as shopping carts need to be externalized to independent storages so the application servers can become stateless, which is essential for seamless scaling of application servers.

The second level of the problem is much harder to solve. This problem is caused by the central, or global, components in the system. A global component assumes it has global knowledge of the entire system, and it often assumes it has a singleton instance across the entire system. Such assumptions will cause many problems when the system is scaled out. We will present a case study of this problem in Chapter 9. As a matter of fact, because these components are single point of failures (SPoF) of the system, they should be eliminated regardless of whether the system is to be migrated to cloud.

■ Data layer

At the data layer, the main challenge is that the database is not designed for storing data from multiple tenants. On one hand, modifying the database schema to accommodate multiple tenants (such as by adding a tenant id column to all tables) is a tedious work and has significant impact on the entire system. On the other hand, if we keep a separate database for each tenant, we need to face the extra work of operating, maintaining, and updating multiple databases.

In many legacy systems, we often see the all-purpose database design, which uses a single database to hold all information that needs to be permanently or temporarily stored. All kinds of data, whether transactional data, log data, or messages, are stuffed into a unified database.

The design is sometimes justified in the name of simplicity and ease of management, but most of the time it is just a result of a habit. When we design databases, we should treat different types of data separately and choose the most appropriate techniques. For example, we may want to keep business data models in a relational database, but put log data into an NoSQL database because of the different write and access patterns as well as the different retention and backup requirements.

When designing a data layer, consider the following principles:

- Use smaller storages. Consider categorizing data into different types and using different storage techniques. For example, log data can be saved on NoSQL storages; temporal data can be saved in cache clusters; business data can be saved in relational databases; analytical data can be saved in data warehouses; and so forth. For different types of data, choose the most appropriate redundancy, backup, and archiving strategies.
- Periodically release unused storage spaces. Remember that cloud is cheap, but it is not free. Retaining unnecessary data means unnecessary costs.
- Avoid business logic in the data layer. Traditionally, many systems use database stored procedures to implement business logic. With the encouragements of leading database vendors, writing business logic directly on the database was once a very popular choice. Migrating such systems to a different database platform is extremely hard. In a changing world, having business logic bound to a particular database technique often turns out to be a bad bet.

8.2.4 Host Single-Tenant Systems on Microsoft Azure for Multiple Tenants

Converting a single-tenant system into a multitenant system is a huge task. Not all service developers have the will and the budget to take up such endeavors. Here we are going to present a strategy that uses IaaS, automation scripts, and supporting services to host single-tenant services as if they were multitenant services on Microsoft Azure. Note that at the time of writing this text, you will need to develop supporting services yourself if you take this approach, as there are no out-of-box services to support this strategy.

The strategy uses a four-step approach: node classification, virtualization, topology abstraction, and finally deployment.

■ Node classification

First, we need to classify service components (hereafter referred to as “nodes”) in a single-tenant system into three categories: stateless nodes, stateful nodes, and global nodes.

- Stateless nodes refer to the service components that do not maintain local states (though they can use externalized state stores). This kind of components can be scaled out directly without any changes. An important characteristic of a stateless node is that each request can be handled independently, and information carried by the request is sufficient for the node to handle the request correctly without relying on any contextual information.
- Stateful nodes are the services components that are aware of user sessions. For these nodes to function as expected, they need to use contextual information saved in these sessions. Most web technologies support externalizing session states to independent storages, hence making the nodes themselves stateless. If such externalization is impossible, we will need to utilize certain sticky session/server affinity techniques to ensure a user is routed to the same server throughout a session.

- Global nodes are the service components that assume to be singleton components that have complete knowledge of the entire system. These components cannot participate in horizontal scaling. Instead, we need to provide isolated environments for them so that they can still operate under single-tenant settings. Many single-tenant databases can be put into this category.

- Virtualization

Then, we need to create deployable packages for each of the identified nodes. These packages can be Microsoft Azure websites, Microsoft Azure cloud services, or virtual machine images. These packages will be used as standard templates for scaling out, so they should not contain any customizations or customer data.

- Topology abstraction

After classification, we can plan for server deployment. The key to this part is to use different routing mechanisms for different types:

- Request-level routing. This applies to stateless components, as well as local state components, when local states can be externalized to an external storage, such as a database or a cache cluster.
- Session-level routing. This applies to local state components that require sticky sessions or server affinity.
- Tenant-level routing. This applies to global state components as well as components that require tenant isolation.

Request-level routing is supported by Microsoft Azure out-of-box service. We simply create multiple instances of your services to be load-balanced. In addition, we can use different session state providers to externalize session states, such as the one that uses Microsoft Azure Caching. We can also use services such as Application Request Routing to achieve sticky sessions. The real challenge resides in tenant-level routing. There are no out-of-box services on Microsoft Azure yet, but we can easily list out two of the high-level requirements: the service shall add minimum overheads, and the redirection decision shall be based solely on untampered, authoritative information, such as a claim in the user's security context. As mentioned earlier, at this point, you need to author such services yourself. Possible directions include a lightweight cloud service, a URL rewrite module, or a DDNS service.

As for high availability, we can use multiple instances as backups of each other for the first two routing methods. For tenant-level routing, we will need to set up a master–slave deployment to provide high availability using failovers. Hence, our routing service shall support failover based on a priority list of server instances.

- Deployment

This topology is a logical definition, which will be instantiated during deployment. Automated system deployments and scaling, including deployments for newly provisioned tenants after initial deployment, can be performed by PowerShell scripts (see Chapter 11), configuration management systems such as Chef, Puppet and DSC. Microsoft Azure Caching service can be used for externalized state storage. Software routers as well as contextual services can be implemented as Microsoft Azure cloud services.

Note: Service Gateway

Service Gateway is an open source project (<http://sg.codeplex.com/>) that provides tenant-level routing, among other features.

8.3 n-Tiered Architecture

n-tiered architecture is not cloud-specific. It is a general architecture that separates the presentation layer, the business layer, and the data layer, as shown in Figure 8.12. This kind of separation of concerns increases the modularity of the system, hence greatly improving maintainability of the system. It is a popular choice not only for on-premise enterprise systems, but also for cloud-based systems.

8.3.1 Characteristics of n-Tiered Architecture

- **Advantage: separation of concerns**
Separation of concerns is a key principle of system design as well as an essential quality of good architecture. The principle requires a component to be responsible for no more than one type of work, and it requires a logical work unit not to be distributed across different components. If you found a component taking up more than one responsibility, or several components having to constantly work together to perform a task, you might spot problems in how component boundaries have been drawn. A system with clear separation of concerns is easier to maintain and upgrade. This is because it is easy to predicate and control the effects of changes when changes are made. On the contrary, a system without such qualities is less maintainable because any changes may cause unpredictable ripple effects that have broad impacts on the entire system.
- **Advantage: scaling independently**
Different layers of an n-tiered system can be scaled independently. For example, for a system that has complex business logic, multiple business layer instances can be used to share the workload, while a single presentation layer instance might suffice. On the other hand, for a system that has complex data operations, the data layer can be scaled out to increase system throughput. In addition, the independent scaling can also minimize operational costs without sacrificing system performance as you can fine-tune each layer separately.
- **Advantage: flexible coupling options**
Components in an n-tiered system can communicate with each other by different means. In addition to direct communication, they can also exchange data via a middle tier such as a service bus. The loose coupling provides buffers between components so that the workload from one layer can be gradually released to another layer to avoid overloading a slower layer. This is just one of the benefits of loose coupling. We will discuss loose coupling in more detail in Chapter 15.
- **Advantage: parallel development and deployment**
Layers of an n-tiered system can be developed using different languages, and can be deployed on different hosting environments. For example, the presentation layer can be authored as a Microsoft Azure cloud service, while the business layer can be a legacy service hosted on a virtual machine, and the data layer can be an SQL Server running on a local network.

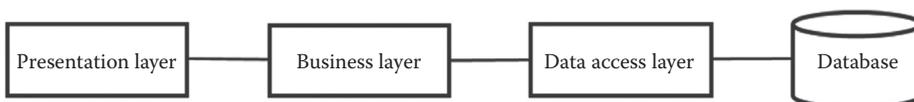


Figure 8.12 n-tiered architecture.

This is the so-called hybrid cloud deployment, where heterogeneous techniques and hosting environments are orchestrated to deliver a complete solution.

- “Disadvantage”: Complexity

One perceived disadvantage of n-tiered architecture is that it is complex. While it is true that an n-tiered system is comprised of multiple components, making it appear complex, we should remember that n-tiered architecture is meant for constructing a complex system. In other words, the complexity does not come from the architecture, but from the problem itself. As a matter of fact, because it allows us to divide and conquer a complex problem, n-tiered architecture helps us to simplify the problem so we can tackle a complex problem in a systematical way.

8.3.2 n-Tier, MVC, and MVVM

- MVC

MVC is a design pattern that appeared in the 1970s. A system following an MVC pattern is made up of Models, Views, and Controllers. There are different ways of understanding and depicting an MVC pattern. Figure 8.13 shows a controller-first variation of the pattern, which is how the architecture of ASP.NET MVC looks. There are several other patterns based on MVC that have been developed over time, including MVP and MVVM.

- MVVM

The core concept of MVVM is a View Model, which is an abstract view that can be bound to a view via data binding. View Models separate business logics and views so that when the UI changes, the business logic is not affected. Another benefit of MVVM is testability, which allows View Model testing to be automated by substituting UI with test scripts. People hold different opinions on the exact responsibility of a View Model. Some think Controllers should be reintroduced into the MVVM pattern, while others think the responsibility of a Controller should be merged into View Models. In addition, people have been debating as to what exactly a Model should do as well. In order for MVVM to guide the whole system design, however, it has to be supplemented with more granularities.

- n-tier, MVC, and MVVM

n-tier architecture and MVC are often used together in a system. n-tier is used to design the overall architecture, while MVC can be used to design the presentation layer. For example, you can create a cloud service with a web front end and a Worker Role back end, and the Web Role can use ASP.NET MVC for the presentation layer design.

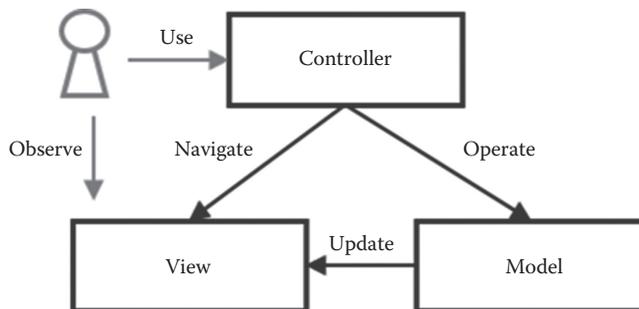


Figure 8.13 ASP.NET MVC architecture.

Because of the need to support rich user interactions, AJAX has become a mainstream technique for website development. In addition, multiclient support raises the requirement of designing client-agnostic APIs. We can combine n-tiered architecture, ASP.NET MVC Web API, and MVVM to construct clearly structured, maintainable rich user interfaces, which we will learn in the following example.

Example 8.2: ASP.NET Web API and MVVM

Difficulty: ***

In this example, we will create a simple website for customer management. Because the key of the example is to demonstrate how to combine ASP.NET Web API and MVVM to create an interactive web, we will not focus on the completeness of the site. Instead, we will just build enough to present the programming model. Here we are going to use knockout.js, which is an open-source library for implementing MVVM pattern with JavaScript.

1. Launch Visual Studio as an administrator. Create a new cloud service with an ASP.NET MVC 4 Web Role. When we create the Role, we will use the **Empty** template, instead of the *Internet Application* template we commonly use.
2. Add a **Customer** class under the **Models** folder of the Web Role. The class code is shown in Code List 8.6.
3. Add a new API-controlled named **CustomerController** (using the **API controller with empty read/write actions** template) to the **Controllers** folder. We can delete all but the **Get()** method, as shown in Code List 8.7.
4. Add a new **HomeController** (using **Empty MVC controller** template) to the **Controllers** folder:

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

5. Add a **Home** controller under the **Views** folder. Then, add a new **Index** view under the new folder.
6. Right-click the Web Role and select the **Manage NuGet Packages** menu.
7. On the **Manage NuGet Packages** dialog, search for “knockout.js.” Then, click the **Install** button to install knockout.js, as shown in Figure 8.14.
8. Similarly, add a reference to the **jQuery** NuGet package.
9. Replace the code in **Index.cshtml** with the code in Code List 8.8. Lines 1–7 of the code define the user interface (the View), which has a <div> tag bound to a Customers collection

CODE LIST 8.6 CUSTOMER MODEL

```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}
```

CODE LIST 8.7 CUSTOMERCONTROLLER

```

using MvcWebRole1.Models;
...
public class CustomerController : ApiController
{
    public IEnumerable<Customer> Get()
    {
        return new Customer[]
        {
            new Customer { Id = 1, Name= "Customer A", Age= 24},
            new Customer { Id = 2, Name= "Customer B", Age = 30}
        };
    }
}

```

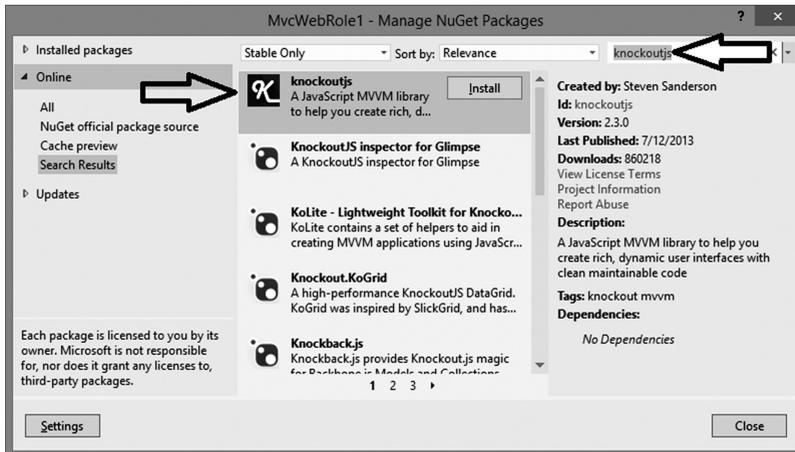


Figure 8.14 knockout.js NuGet package.

through the **data-bind** attribute. Lines 3–6 define the template for each customer display. Lines 11–22 define the View Model. As you can see, the View Model knows nothing about the View. The View is totally separated from the back-end calls (such as in lines 24–28). Furthermore, the JavaScript code and the HTML tags are not mixed together. Because each component has its distinct responsibility, the UI code is clean, easy to understand, and easy to maintain.

Note: Your jQuery version and knockout version are likely to be higher.

- Press **F5** to launch the application. Observe the customer list displayed on the screen. Modify the customer name in one of the customer name fields, and then click the **Update** button. Although we did not write any code to update the View Model with the new values on the View, because of the two-way data bind, the View Model has been automatically updated, as shown in Figure 8.15.

CODE LIST 8.8 CUSTOMER UI USING KNOCKOUT.JS

```

1:<h2>Customer Manager</h2>
2:<div data-bind="foreach: Customers">
3:   <div data-bind="text: Id"></div>
4:   <input data-bind="value: Name" />
5:   <input data-bind="value: Age" />
6:   <input type="button" data-bind="click: Update"
   value="Update" />
7:</div>
8:<script src="~/Scripts/jquery-2.0.3.js"></script>
9:<script src="~/Scripts/knockout-2.3.0.js"></script>
10:<script>
11:   var viewModel = new pageViewModel();
12:   function customerViewModel(id, name, age) {
13:     this.Id = id;
14:     this.Name = ko.observable(name);
15:     this.Age = ko.observable(age);
16:     this.Update = function () {
17:       alert(this.Name());
18:     };
19:   }
20:   function pageViewModel() {
21:     this.Customers = ko.observableArray([]);
22:   }
23:   $(function () {
24:     $.getJSON('/api/Customer', function (data) {
25:       $.each(data, function (key, val) {
26:         viewModel.Customers.push
27:           (new customerViewModel(val.Id, val.Name, val.
           Age));
28:       });
29:       ko.applyBindings(viewModel);
30:     });
31:   });
32:</script>

```

8.3.3 Microsoft Azure Service Bus Queue

Before we discuss developing n-tiered services on Microsoft Azure, we will introduce Microsoft Azure Service Bus Queue service because it is an important technique to achieve loose coupling between different layers.

Microsoft Azure Service Bus provides a series of integration services, including Queue, Topic/Subscription, Relay, and Notification Hub. To use these services, you first need to provision a namespace, in which you can subscribe to multiple Queue, Topic, Relay, and Notification Hub service entities.

The Microsoft Azure Service Bus Queue service and the Microsoft Azure Queue storage service are very similar to each other. They can both be used to transmit messages, and they are consumed in similar ways: Queue storage is accessed via a storage account, and Service Bus Queue is

Customer manager

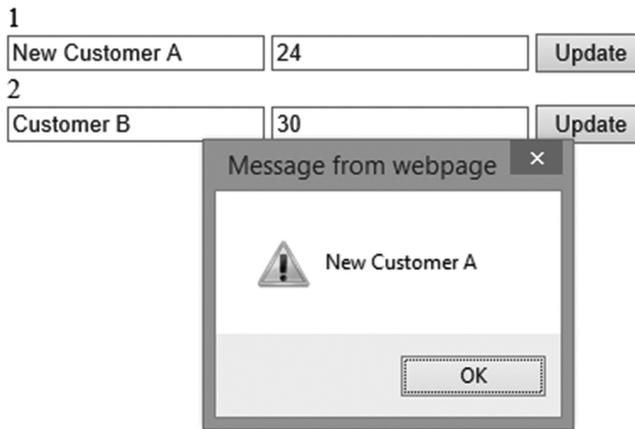


Figure 8.15 Customer manager.

accessed via a namespace; Queue storage is supported by the storage client library, and Service Bus Queue is supported by the service bus client library. Of course, they have significant differences as well. Service Bus Queue can handle larger message sizes, and it provides more functionalities compared to the Queue storage. We will learn more about these features in Chapter 11.

Now, let us learn how to manage a Service Bus namespace with a brief example.

Example 8.3: Managing Service Bus Namespaces and Queues

Difficulty: *

1. Log in to Microsoft Azure Management Portal.
2. Click on the **NEW** icon on the command bar, and then select **APP SERVICES**→**SERVICE BUS**→**QUEUE**→**CUSTOM CREATE**.
3. On the **CREATE A QUEUE** dialog, enter a name for the new queue and a name for the new namespace, and then click the next button to continue, as shown in Figure 8.16. The wizard combines the step of creating the namespace and the step of creating the queue into a single step. Logically, you would create a namespace first and then add service entities such as queues into it.
4. In the next step, accept all default settings and click on the check button to complete the operation. This will create a new Service Bus namespace with a new queue in it.
5. After the namespace has been created, you can use the **CONNECTION INFORMATION** icon to query how to connect to this namespace, as shown in Figure 8.17.
6. On the **Access connection information** dialog, you can look up connection strings you can use to connect to a Service Bus namespace. When you create a Service Bus client, you can use either the SAS connection string or the ACS connection string to connect to the namespace (Figure 8.18).
7. You can also manage your Service Bus namespaces and other entities (such as queues) in Visual Studio. Launch Visual Studio, and then open the **Server Explorer**.
8. Expand the **Microsoft Azure**→**Service Bus** node. Expand the namespace you have just created, and you will find the new queue under the Queues node, as shown in Figure 8.19.

Figure 8.16 Create a new namespace and a queue.

service bus

NAMESPACE NAME	STATUS	LOCATION	SUBSCRIPTION	CREATED DATE	
my-sb-ns →	✓ Active	West US	Azdem169H43081X	9/2/2013 7:46:50 PM	

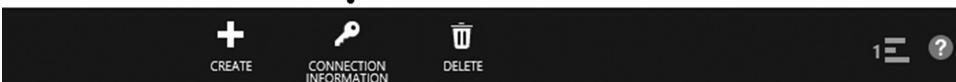
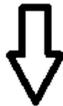


Figure 8.17 Access connection information of a namespace.

9. In addition to creating/configuring/updating queues, you can also send and receive test messages directly from Visual Studio. Right-click on the queue node, and select the **Send a test message** menu to enqueue a test message in the queue, as shown in Figure 8.20.
10. You see a confirmation dialog informing you that a string message has been added to the queue, as shown in Figure 8.21.
11. Similarly, you can right-click the queue again and select the Receive message menu to dequeue the message we just put in the queue, as shown in Figure 8.22.

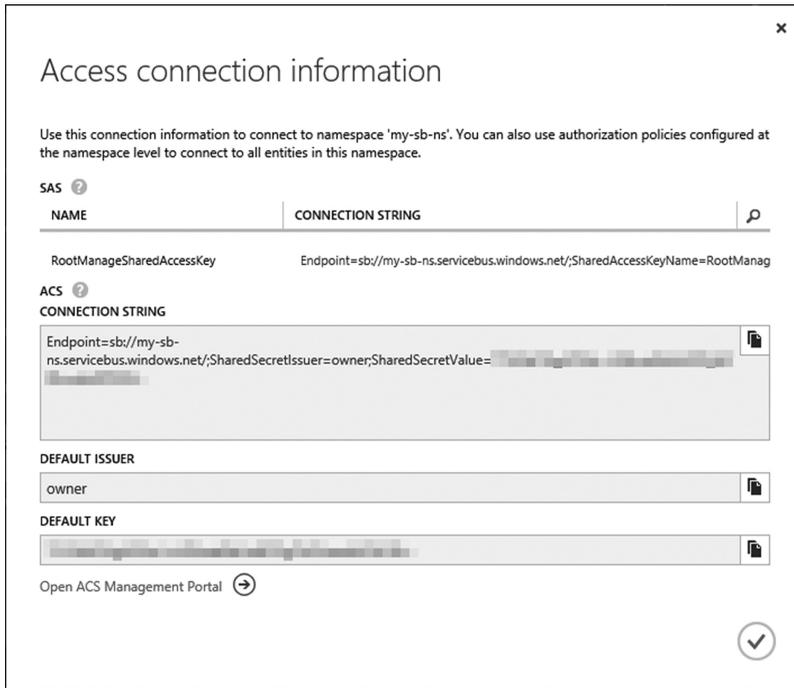


Figure 8.18 Service bus namespace access connection information.

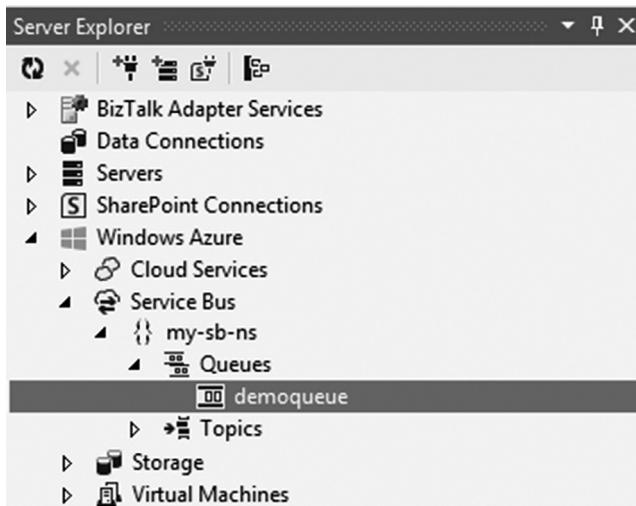


Figure 8.19 Managing queues in Visual Studio.

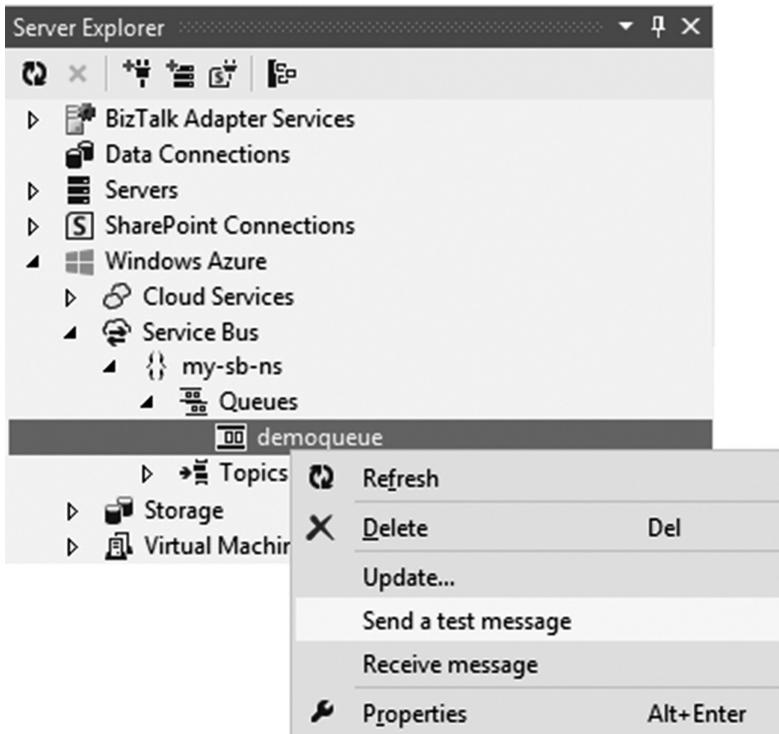


Figure 8.20 Sending a test message from Visual Studio.

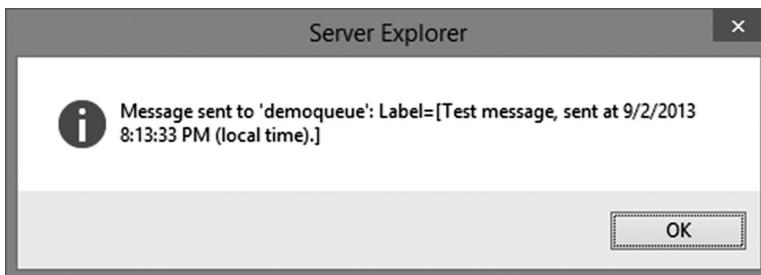


Figure 8.21 Test message has been added to the queue.

8.3.4 Implementing *n*-Tiered Services on Microsoft Azure

Microsoft Azure cloud services provides built-in support for *n*-tiered architecture. When you create a new cloud service project, you can directly add multiple roles into the project to construct a multitiered system. In addition, on Microsoft Azure Management Portal, you can scale different layers independently to optimize system throughput. Microsoft Azure also supports automated scripts for you to automate repetitive jobs. We introduce Microsoft Azure scripting in Chapter 17.

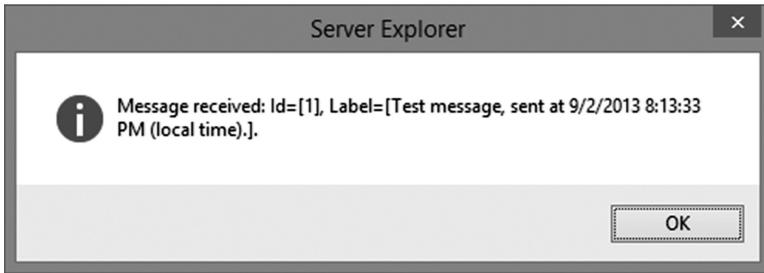


Figure 8.22 Received test message.

We have introduced different ways of cross-layer communications in Section 4.3 and have presented an example of direct communication in Chapter 4. Communicating through a message queue is also a commonly used technique. In the following example, we use Microsoft Azure Service Bus Queue service to achieve asynchronous communications between service layers.

Example 8.4: Service Bus Queue asynchronous communication: A translation service

Difficulty: ****

In this example, we create a Microsoft Azure cloud service that is made up by a Web Role and a Worker Role. The Web Role takes English texts from users and sends them to the Worker Role to translate them into Chinese. The Worker Role in turn uses Microsoft Translator service to perform the translation, and sends the results back to the Web Role via another queue.

Part of the source code in this example is taken from MSDN (<http://msdn.microsoft.com/en-us/library/hh454950.aspx>).

1. Launch Visual Studio as an administrator. Create a new cloud service.
2. Add an **ASP.NET MVC 4 Web Role** (using the Internet Application template) and a **Worker Role** (using Worker Role with Service Bus Queue template) to the service, as shown in Figure 8.23.

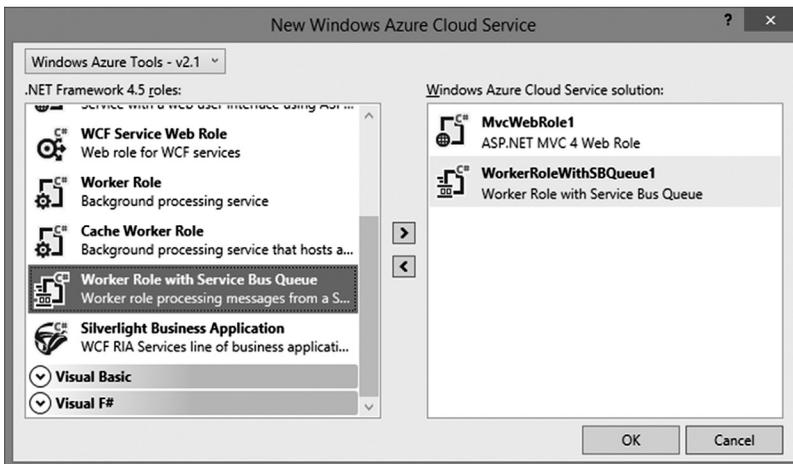


Figure 8.23 Creating an n-tiered cloud service.

3. In the cloud service project, double click the Worker Role to open its Properties page. Switch to the **Settings** tab. Then, replace the value of **Microsoft.ServiceBus.ConnectionString** with the connection string to your Service Bus namespace (see step 6 in Example 8.3). Save your changes.
4. Open the **WorkerRole.cs** file in the Worker Role project to examine its code.
5. Code List 8.9 shows the default `OnStart()` implementation. Lines 7–9 guide the connection string from the Role settings. Then, lines 10 and 11 create a new **NamespaceManager** instance using its static **CreateFromConnectionString** method. The **NamespaceManager** class provides various methods for managing Service Bus entities, such as checking if a queue exists (line 12) and creating a new queue (line 14). Finally, in order to interact with the queue, a **QueueClient** instance is created (lines 18 and 19).
6. Code List 8.10 shows the default implementation of the `Run()` method. This implementation uses an event-driven style. When a message is received and the **OnMessage** is triggered (line 7), the callback, which is an anonymous method in this case, is invoked (lines 7–19). Similar to Microsoft Azure Queue service, after you have successfully processed a message, you are supposed to mark the message as **Completed** within the default time window; otherwise, the message will reappear on the queue after the message lock expires. You do not see the message is marked in Code List 8.10 because by default the **OnMessage** method automatically marks a message as Completed once the callback returns. This behavior can be changed by passing an **OnMessageOptions** instance (with **AutoComplete** set to false) as the second parameter of the **OnMessage** method.
7. Press **F5** to launch the application. Use the **Server Explorer** to send a couple of test messages to the **ProcessingQueue** queue, which is automatically created when the Worker Role starts.

CODE LIST 8.9 ONSTART() METHOD OF THE WORKER ROLE

```

1:public override bool OnStart()
2:{
3:    // Set the maximum number of concurrent connections
4:    ServicePointManager.DefaultConnectionLimit = 12;
5:
6:    // Create the queue if it does not exist already
7:    string connectionString =
8:        CloudConfigurationManager.GetSetting
9:            ("Microsoft.ServiceBus.ConnectionString");
10:   var namespaceManager = NamespaceManager.
        CreateFromConnectionString
11:       (connectionString);
12:   if (!namespaceManager.QueueExists(QueueName))
13:   {
14:       namespaceManager.CreateQueue(QueueName);
15:   }
16:
17:   // Initialize the connection to Service Bus Queue
18:   Client = QueueClient.CreateFromConnectionString
19:       (connectionString, QueueName);
20:   return base.OnStart();
21:}

```

CODE LIST 8.10 RUN() METHOD OF THE WORKER ROLE

```

1:public override void Run()
2:{
3:    Trace.WriteLine("Starting processing of messages");
4:
5:    // Initiates the message pump and callback is invoked for
        // each message
6:    // that is received, calling close on the client will stop
        // the pump.
7:    Client.OnMessage((receivedMessage) =>
8:        {
9:            try
10:           {
11:               // Process the message
12:               Trace.WriteLine("Processing Service Bus message:
                    " +
13:                   receivedMessage.SequenceNumber.
                        ToString());
14:           }
15:           catch
16:           {
17:               // Handle any message processing specific
                    // exceptions here
18:           }
19:       });
20:
21:    CompletedEvent.WaitOne();
22:}

```



Figure 8.24 Log entries generated by the Worker Role while processing test messages.

8. On the Compute Emulator UI, observe log entries recorded by the Work Role while processing the messages, as shown in Figure 8.24.
9. Now let us implement the translation functionality. First, add a Class Library project to the solution. Then, add a **SampleJob** class to the library:

```
public class SampleJob
{
    public int Id { get; set; }
    public string OriginalText { get; set; }
    public string TranslatedText { get; set; }
}
```

10. In the Worker Role project, add a reference to the previous Class Library as well as a reference to **System.Web**.
11. To use Microsoft Translation Service, you first need to subscribe to it. Then, you need to authenticate using OAuth when invoking the API. In the **WorkerRole.cs** file, add an **AdmAccessToken** class:

```
using System.Runtime.Serialization;
...
[DataContract]
public class AdmAccessToken
{
    [DataMember]
    public string access_token { get; set; }
    [DataMember]
    public string token_type { get; set; }
    [DataMember]
    public string expires_in { get; set; }
    [DataMember]
    public string scope { get; set; }
}
```

12. We also need to create a second queue for returning the translation results. Similar to the **ProcessingQueue**, we create a **ReturnQueue** in this case:

```
public class WorkerRole : RoleEntryPoint
{
    ...
    const string ReturnQueueName = "ReturnQueue";
    QueueClient ReturnClient;
    ...
    public override bool OnStart()
    {
        ...
        if (!namespaceManager.QueueExists(ReturnQueueName))
        {
            namespaceManager.CreateQueue(ReturnQueueName);
        }
        ...
        ReturnClient = QueueClient.CreateFromConnectionString
            (connectionString, ReturnQueueName);
        ...
    }
}
```

CODE LIST 8.11 USE OF THE RETURNING QUEUE

```

1:public override void Run()
2:{
3:    ...
4:    Client.OnMessage((receivedMessage) =>
5:        {
6:            try
7:            {
8:                if (receivedMessage != null)
9:                {
10:                   SampleJob job = receivedMessage.
                       GetBody<SampleJob>();
11:                   job.TranslatedText = translateText(job.
                       OriginalText);
12:                   ReturnClient.Send(new BrokeredMessage(job));
13:                }
14:            }
15:            catch
16:            {
17:                // Handle any message processing specific
                       exceptions here
18:            }
19:        });
20:
21:    CompletedEvent.WaitOne();
22:}

```

13. In the Worker Role `Run()` method, invoke a `translateText()` method, which we defines in a moment, and then enqueue the result to the **ReturnQueue**. Code List 8.11 shows how this is done. First, line 10 retrieves the **SampleJob** instance embedded in the body of the received **BrokeredMessage** instance. Then, line 11 invokes the translation method. Finally, the updated **SampleJob** instance is packaged into another **BrokeredMessage** instance and sent to the returning queue (line 12).
14. Now let us look at the `translateText()` method. The complete code of the method is shown in Code List 8.12. Lines 5–24 are to get the security token, and lines 26–41 are to invoke the translation API, attaching the security token as a bearer token (line 32).
15. In the Web Role project, add a reference to the Class Library containing the **SampleJob** class.
16. In the Web Role project, add a reference to the **WindowsAzure.ServiceBus** NuGet package (see Figure 8.25). This is a typical way of consuming various Microsoft Azure services—to add references to corresponding NuGet packages in your project and code away.
17. Replicate the **Microsoft.ServiceBus.Connection** setting from the Worker Role to the Web Role.
18. Add a new API Controller named **JobController** to the Web Role. The code of creating and operating the queues is very similar to that in the Worker Role. Among the methods, `SendJob()` is used to send a translation job, and `GetCompletedJob()` is used to query completed jobs. Note that the `GetCompletedJob()` method uses periodical polling instead of the event-driven mode. The complete source code is listed in Code List 8.13.

CODE LIST 8.12 TRANSLATETEXT METHOD

```

1:private string tranlateText(string text)
2:{
3:    string clientID = "[Client ID]";
4:    string clientSecret = "[Client Secret]";
5:    string strTranslatorAccessURI =
6:        "https://datamarket.accesscontrol.windows.net/v2/
        OAuth2-13";
7:    string strRequestDetails =
8:        string.Format("grant_type=client_credentials&client_
        id={0}
9:            &client_secret={1}&scope=http://api.
        microsofttranslator.com",
10:            HttpUtility.UrlEncode(clientID),
11:            HttpUtility.UrlEncode(clientSecret));
12:    WebRequest webRequest = WebRequest.
        Create(strTranslatorAccessURI);
13:    webRequest.ContentType = "application/x-www-form-urlencoded";
14:    webRequest.Method = "POST";
15:    byte[] bytes = System.Text.Encoding.ASCII.
        GetBytes(strRequestDetails);
16:    webRequest.ContentLength = bytes.Length;
17:    using (System.IO.Stream outputStream = webRequest.
        GetRequestStream())
18:    {
19:        outputStream.Write(bytes, 0, bytes.Length);
20:    }
21:    WebResponse webResponse = webRequest.GetResponse();
22:    DataContractJsonSerializer serializer =
23:    new DataContractJsonSerializer(typeof(AdmAccessToken));
24:    AdmAccessToken token = (AdmAccessToken)serializer.ReadObject
25:
26:        (webResponse.GetResponseStream());
27:    string headerValue = "Bearer " + token.access_token;
28:    string uri =
29:        "http://api.microsofttranslator.com/v2/Http.svc/
        Translate?text="
30:        + HttpUtility.UrlEncode(text) + "&from=en&to=zh-CHS";
31:    HttpWebRequest request = (HttpWebRequest)WebRequest.
        Create(uri);
32:    request.Headers.Add("Authorization", headerValue);
33:    try
34:    {
35:        WebResponse response = request.GetResponse();
36:        using (Stream stream = response.GetResponseStream())
37:        {
38:            XElement elm = XElement.Load(stream);
39:            var ret = elm.FirstNode.ToString();
40:            return ret;

```

```

41:     }
42: }
43: catch (WebException)
44: {
45:     return text;
46: }
47: }

```

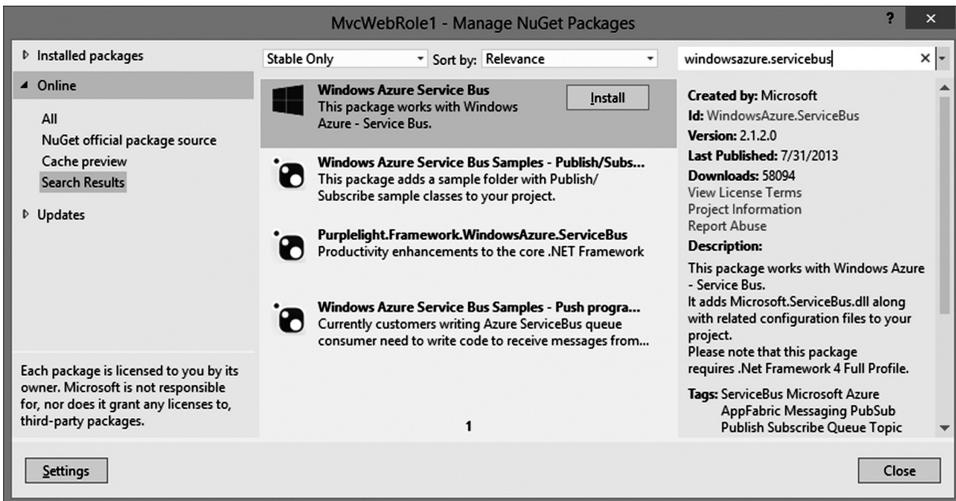


Figure 8.25 Adding reference to WindowsAzure.ServiceBus NuGet package.

19. Replace the entire code in **View\Home\Index.cshtml** with the code in Code List 8.14. The page refreshes itself every 2 s to display newly found translation results from the returning queue (lines 11–22). When a user clicks on the **Submit** button, the translation task is added to the sending queue (lines 6–10).
20. Press **F5** to launch the application. Enter the English texts to be translated, and click the **Submit** button. Observe the translation results showing up on the page, as shown in Figure 8.26.

8.4 Distributed System

In a distributed system, multiple distributed computers coordinate their work over a network to jointly complete a complex task. Two key aspects of distributed systems are parallel computing and internode communications. Microsoft Azure provides two different communication mechanisms: message-based integrations (more on this in Chapter 15), and Microsoft Azure Service Bus Relay service, as shown in Figure 8.27. Relayed connections

CODE LIST 8.13 JOBCONTROLLER

```

using Microsoft.ServiceBus.Messaging;
using Microsoft.WindowsAzure;
using Microsoft.ServiceBus;
...
public class JobController : ApiController
{
    const string QueueName = "ProcessingQueue";
    const string ReturnQueue = "ReturnQueue";
    QueueClient Client;
    QueueClient ReturnClient;
    Random rand = new Random();

    public JobController()
    {
        string connectionString = CloudConfigurationManager
            .GetSetting("Microsoft.ServiceBus.ConnectionString");
        var namespaceManager = NamespaceManager
            .CreateFromConnectionString(connectionString);
        if (!namespaceManager.QueueExists(QueueName))
            namespaceManager.CreateQueue(QueueName);
        if (!namespaceManager.QueueExists(ReturnQueue))
            namespaceManager.CreateQueue(ReturnQueue);
        Client = QueueClient.CreateFromConnectionString
            (connectionString, QueueName);
        ReturnClient = QueueClient.CreateFromConnectionString
            (connectionString, ReturnQueue);
    }
    [HttpGet]
    public void SendJob(string payload)
    {
        Client.Send(new BrokeredMessage(new SampleJob
            { Id = rand.Next(1, 100), OriginalText = payload }));
    }
    [HttpGet]
    public SampleJob GetCompletedJob()
    {
        var message = ReturnClient.Receive(TimeSpan.FromSeconds(3));
        if (message != null)
        {
            var ret = message.GetBody<SampleJob>();
            message.Complete();
            return ret;
        }
        else
            return null;
    }
}
...

```

CODE LIST 8.14 USER INTERFACE

```

1:Enter the text to be translated: <input id="jobText" type="text"
  /><br />
2:<input id="submit" type="button" value="Submit" />
3:<div id="jobs"></div>
4:@section Scripts{
5:  <script>
6:    $('#submit').click(function () {
7:      $.getJSON('/api/Job/SendJob?'
8:        + 'payload='
9:        + encodeURIComponent(jobText.value), null);
10:    });
11:    $(function () {
12:      setInterval(refreshJobs, 2000);
13:    });
14:    function refreshJobs() {
15:      $.getJSON('/api/Job/GetCompletedJob', function (json) {
16:        if (json != null)
17:          $('#jobs').append('<b>Task ' + json.Id
18:            + ' completed. </b><i>Original Text</i>:'
19:            + json.OriginalText
20:            + ', <i>Translated Text</i>:' + json.
21:              TranslatedText
22:            + '<br/>');
23:        });
24:      </script>
25:}

```

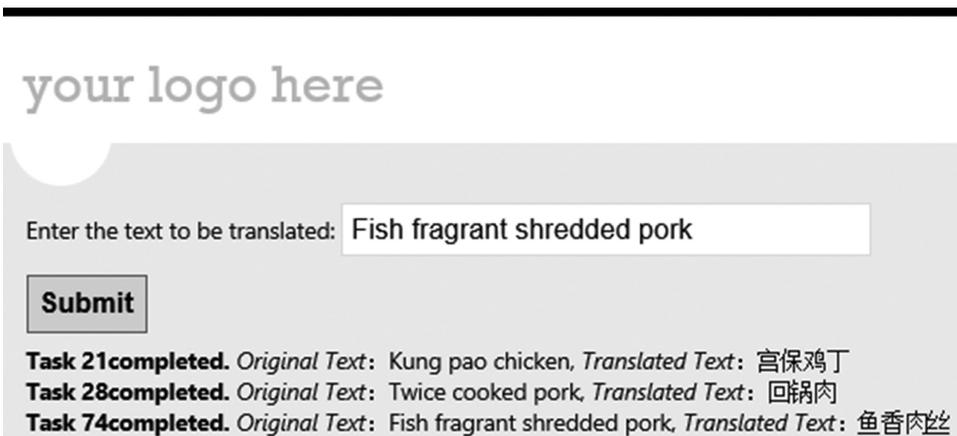


Figure 8.26 Translation service running results.

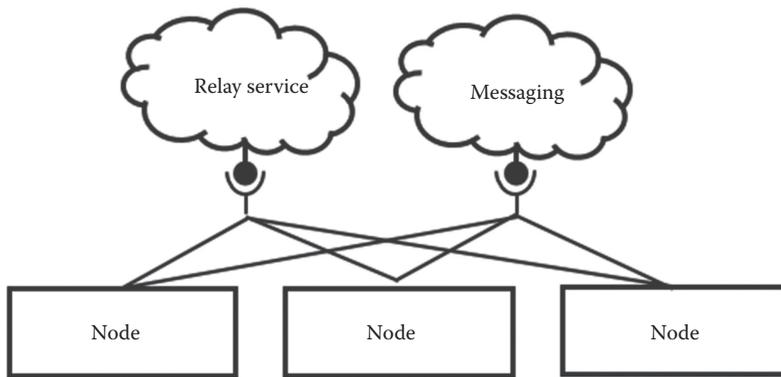


Figure 8.27 Distributed systems using Microsoft Azure.

provide highly available connectivity so that any two computers connected to the Internet can communicate with each other.

8.4.1 Message-Based Connections

In the previous section, we implemented an n-tiered system in which roles communicate with each other via Service Bus Queues. Because Microsoft Azure Service Bus is an SaaS that is accessible over the Internet, it can be used as a medium to connect any two computers on the Internet. In addition to the Queue service, Service Bus provides a Topic/Subscription service, where a publisher can send a message to a topic, which forwards the message to all attached subscriptions. So, the service is ideal for broadcasting messages to a group of subscribers. In addition, each subscription can apply different filters to receive messages that satisfy certain criteria.

Example 8.5: Service Bus Topic/Subscription: A simple chat system

Difficulty: **

In this example, we create a Windows Console chat program. When the program starts, it prompts the user to enter a self-picked user name, and then to create or join a chat room. Once in a chat room, the user can exchange messages with all users within the same room. Each chat room is a Service Bus Topic, which is subscribed by all the users in the same room. Messages sent to the Topic are broadcasted to all Subscriptions.

This example also shows that you do not have to access Microsoft Azure SaaS services in a cloud service project. You can leverage these services from any type of applications you are building. Although in this case we are building Windows Console applications, we can still use Microsoft Azure services to achieve our goals.

1. Create a new Windows Console application.
2. Add a reference to the WindowsAzure.ServiceBus NuGet package (see step 16 in Example 8.4).
3. First, we define a **ChatText** class, which represents a chat message (see Code List 8.15). Because the messages sent to Service Bus need to be serializable, we decorate the class and its members with the **DataContract** and the **DataMember** attributes.

CODE LIST 8.15 CHATTEXT CLASS

```
[DataContract]
class ChatText
{
    [DataMember]
    public string User { get; set; }
    [DataMember]
    public string Text { get; set; }
    [DataMember]
    public int Color { get; set; }
    public ChatText(string user, int color, string text)
    {
        User = user;
        Text = text;
        Color = color;
    }
}
```

- Next, we define a **ChatRoom** class to encapsulate the behaviors and properties of a chat room. The **ChatRoom** class supports an event-driven programming mode. When a new message is received, it raises a **TextReceived** event. The event parameter is defined as follows:

```
class TextEventArgs : EventArgs
{
    public ChatText ChatText { get; private set; }
    public TextEventArgs(ChatText chatText)
    {
        ChatText = chatText;
    }
}
```

- The **ChatRoom** class is the core of the program. Lines 13–16 in Code List 8.16 create a Service Bus Topic for the chat room. Note that line 14 specifies that the Topic should be automatically removed when there are no activities for a period of time (Service Bus Queue supports a similar feature as well). Lines 17 and 18 create a new Subscription to the Topic for the current user. Lines 24–29 are the same event-driven programming mode that we have seen in the previous example. This event-driven programming mode is very suitable for writing client applications.
- The main program is relatively easy. You may consult the comments in Code List 8.17.
- Press **Ctrl + F5** to launch the program. When the program starts, it asks for a user name and a chat room name (see Figure 8.28). Note that in this version because we are directly using user inputs as the Topic name and the Subscription name, the names can only contain English letters, numbers, and dashes.
- You can press **Ctrl + F5** to launch multiple instances of the program, or distribute the executable to your friends (anywhere in the world) to try out multiuser chatting, as shown in Figure 8.29.

CODE LIST 8.16 CHATROOM CLASS

```

1:class ChatRoom
2:{
3:    public event EventHandler<EventArgs> TextReceived;
4:    public string Name { get; private set; }
5:    TopicClient mTopicClient;
6:    SubscriptionClient mSubscriptionClient;
7:    const string mConString = "[Service Bus Connection String]";
8:    public ChatRoom(string chatRoom, string userName)
9:    {
10:        Name = chatRoom;
11:        NamespaceManager nm =
12:            NamespaceManager.CreateFromConnectionString
13:                (mConString);
14:        TopicDescription desc = new TopicDescription(chatRoom);
15:        desc.AutoDeleteOnIdle = TimeSpan.FromMinutes(10);
16:        if (!nm.TopicExists(chatRoom))
17:            nm.CreateTopic(chatRoom);
18:        if (!nm.SubscriptionExists(chatRoom, userName))
19:            nm.CreateSubscription(chatRoom, userName);
20:        mTopicClient = TopicClient
21:            .CreateFromConnectionString(mConString, chatRoom);
22:        mSubscriptionClient = SubscriptionClient
23:            .CreateFromConnectionString
24:                (mConString, chatRoom, userName);
25:        mSubscriptionClient.OnMessage(m) =>
26:        {
27:            var text = m.GetBody<ChatText>();
28:            if (TextReceived != null)
29:                TextReceived(this, new EventArgs(text));
30:        }
31:    public void Send(string user, int color, string text)
32:    {
33:        mTopicClient.Send(
34:            new BrokeredMessage(new ChatText(user, color,
35:                text)));
36:    }
37:    public void Close()
38:    {
39:        mSubscriptionClient.Close();
40:        mTopicClient.Close();
41:    }

```

CODE LIST 8.17 THE MAIN PROGRAM OF THE CHAT PROGRAM

```

using Microsoft.ServiceBus;
using Microsoft.ServiceBus.Messaging;
using System.Collections.Generic;
using System.Runtime.Serialization;
...
class Program
{
    static int linePos = 2;//display column of current row
    static int myColor = 1;//color index of current user
    //This variable is for conversion from int to ConsoleColor
    static Array mColors = Enum.GetValues(typeof(ConsoleColor));
    static void Main(string[] args)
    {
        Random rand = new Random();
        myColor = rand.Next(0, mColors.Length);//Randomly assign a
            display color
        Console.Clear();
        Console.Write("Please Enter a User Name: ");
        string userName = Console.ReadLine();
        //Enter a new chat room name, or enter an existing name to
        //join a previously created room
        Console.Write("Please Enter a Chat Room Name: ");
        string chatRoom = Console.ReadLine();
        //Chat room UI
        Console.Clear();
        Console.Write(userName);
        Console.SetCursorPosition
            (Console.WindowWidth - chatRoom.Length, 0);
        Console.Write(chatRoom);
        Console.Write(new String('*', Console.WindowWidth));
        Console.SetCursorPosition(0, Console.WindowHeight - 4);
        Console.Write(new String('*', Console.WindowWidth));
        ChatRoom room = new ChatRoom(chatRoom, userName);
        room.TextReceived += room_TextReceived;//handle new message
            event
        while (true)
        {
            setCursorAtBottom();
            var input = Console.ReadLine();
            if (string.IsNullOrEmpty(input))//enter empty string to
                exit
                break;
            room.Send(userName, myColor, input);//send message
        }
        room.Close();//disconnect
    }
}

```

```

static void setCursorAtBottom()
{
    //put the cursor at the bottom of the screen for user inputs
    Console.SetCursorPosition(0, Console.WindowHeight - 3);
    Console.Write(new String(' ', Console.WindowWidth));
    Console.SetCursorPosition(0, Console.WindowHeight - 3);
    Console.Write("] ");
}
static void room_TextReceived(object sender, TextEventArgs e)
{
    if (linePos >= Console.WindowHeight - 4)
    {
        //to avoid scrolling, clear chat screen and reset the
        cursor
        //at the top of the screen
        for (int i = 2; i < Console.WindowHeight - 4; i++)
        {
            Console.SetCursorPosition(0, i);
            Console.Write(new string(' ', Console.WindowWidth));
        }
        linePos = 2;
    }
    //Display newly received message
    Console.SetCursorPosition(0, linePos);
    Console.ForegroundColor =
        (ConsoleColor)mColors.GetValue(e.ChatText.Color);
    Console.Write("[ " + e.ChatText.User + "]:");
    Console.ForegroundColor = ConsoleColor.White;
    Console.Write(e.ChatText.Text);
    linePos++;
    setCursorAtBottom();
}
}

```



The screenshot shows a black console window with white text. The first line reads "Please Enter a User Name: tom" and the second line reads "Please Enter a Chat Room Name: room3_". The cursor is positioned at the end of the second line.

Figure 8.28 Launching the chat program.

8.4.2 Relayed Connections

Microsoft Azure Service Bus Relay service provides a highly available service to connect any two computers over the Internet. Relayed connections can be used to construct distributed systems as well as to provide local-to-cloud connectivity in hybrid cloud solutions. For example, you can expose a local WCF service to the public through a Service Bus Relay endpoint, even if your WCF service is running on a machine without public IP address, behind layers of

The image shows two overlapping Windows command prompt windows. The top window, titled 'C:\WINDOWS\system32\cmd.exe', shows a chat session where Jack asks Tom what's happening, Tom replies, Jack says the TPS report is still on the old cover sheet, and Tom apologizes and says he'll fix it. The bottom window, also titled 'C:\WINDOWS\system32\cmd.exe', shows the same chat session from Tom's perspective, with Jack's messages appearing as '[jack]:Hi Tom. What's happenin?'. Both windows have a 'room3' tab in the top right corner.

Figure 8.29 Multiuser chat.

firewalls, or even without a static IP address. In the following example, we build a distributed game using Service Bus Relay service.

Example 8.6: Service Bus Relay: A WCF mastermind game

Difficulty: ***

Mastermind is one of my favorite board games when I was in elementary school. The game is a code-breaking game played by two players, one as the code maker and the other as the code breaker. The code maker randomly picks 4 pegs of 6 possible colors to make a code, and the code breaker tries to crack the code within 12 attempts. Each time after the code breaker makes a guess, the code maker provides some clues by showing key pegs based on how well the breaker's guessed code matches with the actual code:

- If the breaker guessed correctly in both color and position of a code peg, he gets a black key peg.
- If the breaker guessed the color correctly but not the position of a code peg, he gets a white key peg.

For example, if the code is “red-red-blue-yellow,” and the breaker’s guess is “red-green-blue-red,” then the breaker gets two black pegs (one for the first red peg and one for the blue peg) and a white peg (for the last red, which is correct in color but wrong in position).

It has been proved that the code can be decrypted within six guesses. We leave the algorithm details for interested readers to explore by themselves. In this example, we create a WCF service that acts as the code maker. Although the WCF service will be hosted on a local computer, any client can access this service over the Internet, provided they can pass authentication.

1. Create a new Windows Console application (MastermindWCF.Server).
2. Add a reference to the **WindowsAzure.ServiceBus** NuGet package (see step 16 in Example 8.4).
3. Add a C# Class Library (MastermindWCF.Lib) project to the solution.
4. In the Class Library project, add a reference to the **System.ServiceModel** assembly.

- In the Class Library project, define an **IMastermindEncoder** interface, which defines the WCF service contract. The contract contains two methods—a **StartGame()** method to start a new game, and a **Guess()** method for the player to submit a guess and to get a response from the server. When calling the **Guess()** method, the client has to provide the same identifier it gets when it calls the **StartGame()** method. This design allows the server to serve for multiple game sessions at the same time. The **Guess()** method returns a string with letters “W(hite)” and “B(lack)”, representing the key pegs. The string “BBBB” indicates that the code has been decrypted.

```
using System.ServiceModel;
...
namespace MastermindWCF.Lib
{
    [ServiceContract(Namespace = "urn:mm")]
    public interface IMastermindEncoder
    {
        [OperationContract]
        string StartGame();
        [OperationContract]
        string Guess(string gameId, string pattern);
    }
}
```

- Define an **IMastermindEncoderChannel** interface, which will facilitate channel lifecycle management.

```
namespace MastermindWCF.Lib
{
    ...
    public interface IMastermindEncoderChannel : IMastermindEncoder,
        IClientChannel { }
}
```

- In the **MastermindWCF.Server** project, add a reference to **MastermindWCF.Lib**, a reference to **System.ServiceModel**, and a reference to the **WindowsAzure.ServiceBus** NuGet package.
- Implement the **IMastermindEncoder** interface in the **MastermindWCF.Server** project, as show in Code List 8.18.
- Modify the **App.config** file in the **MastermindWCF.Server** project. Although you can configure a WCF service in code, we recommend you use the configuration file as much as possible so the service can be reconfigured if necessary at runtime without changing the code. Code List 8.19 shows the configuration file, which defines two endpoints for the service: one is a local address using the **netTcpBinding** binding, and the other is a Relay address using the **netTcpRelayBinding** binding. Observe that the format of the second address is **sb://[namespace].servicebus.windows.net/mastermind**. This endpoint also has an associated behavior (*sbTokenProvider*), which specifies that the security mode is a shared secret (the combination of the Service Bus account and its secret key).

CODE LIST 8.18 SERVER IMPLEMENTATION

```

using MastermindWCF.Lib;
using System;
using System.Collections.Generic;

namespace MastermindWCF.Server
{
    public class MastermindEncoder : IMastermindEncoder
    {
        private Dictionary<string, string> mGame =
            new Dictionary<string, string>(); //save states for
            multiple games
        private string mColors = "RGBYPO";
        private Random mRand = new Random();

        public string StartGame()
        {
            Guid id = Guid.NewGuid();
            mGame.Add(id.ToString("N"), makeAPattern());
            return id.ToString("N");
        }

        public string Guess(string gameId, string pattern)
        {
            return makeFeedback(mGame[gameId], pattern);
        }

        private string makeAPattern() //generate a random code
        {
            string ret = "";
            for (int i = 0; i < 4; i++)
                ret += mColors[mRand.Next(0, mColors.Length)];
            return ret;
        }

        private string makeFeedback(string code, string pattern)
        {
            string ret = "";
            var theCode = code.ToCharArray();
            var thePattern = pattern.ToCharArray();
            for (int i = 0; i < theCode.Length; i++)
            {
                if (theCode[i] == thePattern[i])

```


CODE LIST 8.19 WCF SERVICE CONFIGURATION

```

<system.serviceModel>
  <services>
    <service name="MastermindWCF.Server.MastermindEncoder">
      <endpoint contract="MastermindWCF.Lib.IMastermindEncoder"
        binding="netTcpBinding"
        address="net.tcp://localhost:3456/mastermind" />
      <endpoint contract="MastermindWCF.Lib.IMastermindEncoder"
        binding="netTcpRelayBinding"
        address="sb://[namespace].servicebus.windows.net/mastermind"
        behaviorConfiguration="sbTokenProvider" />
    </service>
  </services>
  <behaviors>
    <endpointBehaviors>
      <behavior name="sbTokenProvider">
        <transportClientEndpointBehavior>
          <tokenProvider>
            <sharedSecret issuerName="[issuer, such as owner]"
              issuerSecret="[access key]" />
          </tokenProvider>
        </transportClientEndpointBehavior>
      </behavior>
    </endpointBehaviors>
  </behaviors>
  <extensions>
    ...
  </extensions>
</system.serviceModel>

```

11. Add a new Microsoft Azure Console application named MastermindWCF.Client to the solution. Add a reference to **MastermindWCF.Lib**.
12. Add a reference to the **WindowsAzure.ServiceBus** NuGet package.
13. Enter WCF client configuration in the App.config file of the client program:

```

<system.serviceModel>
  <client>
    <endpoint name="Mastermind" contract="MastermindWCF.Lib.
      IMastermindEncoder"
      binding="netTcpRelayBinding"
      address="sb://[namespace].servicebus.windows.net/mastermind"
      behaviorConfiguration="sbTokenProvider" />
  </client>
  <behaviors>
    <endpointBehaviors>

```

```

    <behavior name="sbTokenProvider">
      <transportClientEndpointBehavior>
        <tokenProvider>
          <sharedSecret issuerName="[issuer, such as owner]"
            issuerSecret="[access key]" />
        </tokenProvider>
      </transportClientEndpointBehavior>
    </behavior>
  </endpointBehaviors>
</behaviors>
<extensions>
  ...
</extensions>
</system.serviceModel>

```

14. Implement the client. The code is very straightforward, as you can see in Code List 8.20.
15. Set both the server application and the client application as startup projects.
16. Press **F5** to run the applications.
17. When the server displays *Press [Enter] to close the server*, indicating that it is ready, press the **Enter** key on the client application to start a new game, as shown in Figure 8.30.
18. Enter your guesses, and refine your inputs based on the server's feedbacks. Try to guess the code within 12 attempts, as shown in Figure 8.31.

CODE LIST 8.20 MASTERMIND CLIENT

```

using MastermindWCF.Lib;
using System.ServiceModel;
...
namespace MastermindWCF.Client
{
    class Program
    {
        static void Main(string[] args)
        {
            var factory = new
                ChannelFactory<IMastermindEncoderChannel>
                    ("Mastermind");
            Console.WriteLine("Press [Enter] to start game.");
            Console.ReadLine();
            using (var channel = factory.CreateChannel())
            {
                string game = channel.StartGame();
                int count = 1;
                bool won = false;
                while (count < 12)

```

```
        {
            Console.WriteLine(string.Format("Guess #{0}:",
                count));
            var pattern = Console.ReadLine();
            if (string.IsNullOrEmpty(pattern))
                break;
            string feedback = channel.Guess(game, pattern);
            Console.WriteLine("Server returned:" +
                feedback);
            if (feedback == "BBBB")
            {
                won = true;
                break;
            }
            count++;
        }
        Console.WriteLine(won ? "You won!" : "You lost!");
    }
    Console.WriteLine("Press [Enter] to exit game.");
    Console.ReadLine();
}
}
```

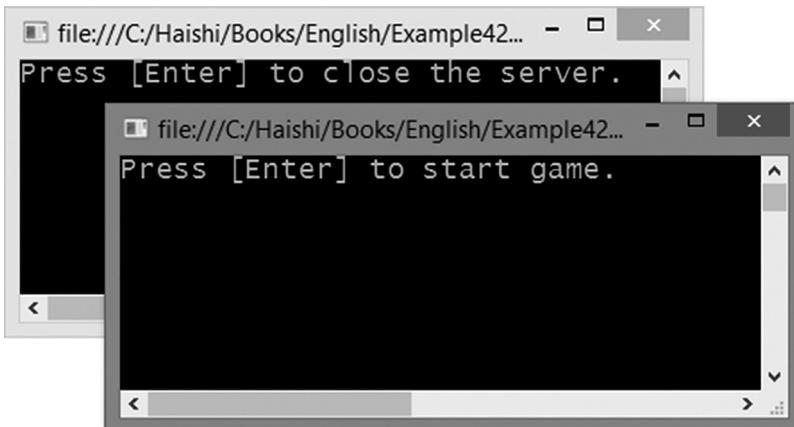
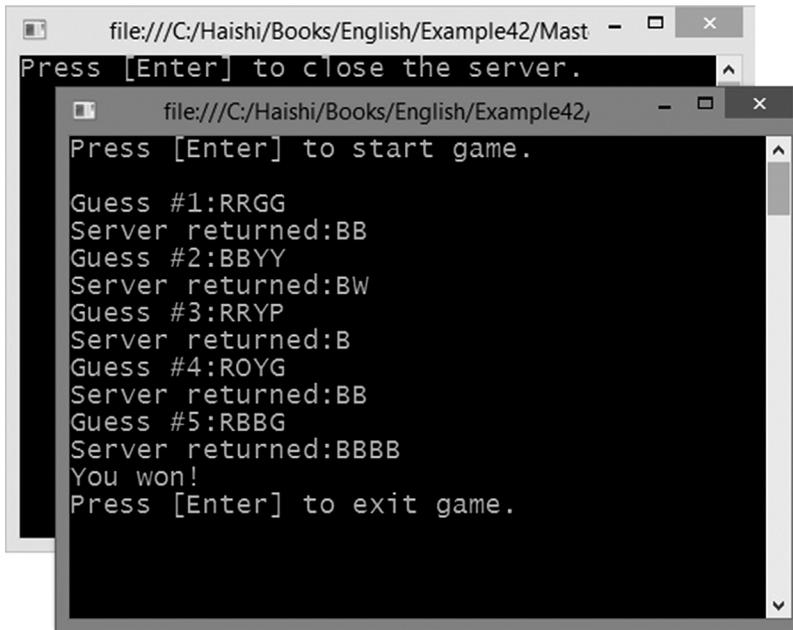


Figure 8.30 Start a new game.



```
file:///C:/Haishi/Books/English/Example42/Mast - □ ×
Press [Enter] to close the server.
file:///C:/Haishi/Books/English/Example42, - □ ×
Press [Enter] to start game.
Guess #1:RRGG
Server returned:BB
Guess #2:BBYY
Server returned:BW
Guess #3:RRYP
Server returned:B
Guess #4:ROYG
Server returned:BB
Guess #5:RBBG
Server returned:BBBB
You won!
Press [Enter] to exit game.
```

Figure 8.31 Sample game session.

8.5 Summary

In this chapter, we discussed different architecture choices for designing cloud-based solutions, including client/server, browser/server, n-tier, and distributed systems. For each of the architectures, we summarized its characteristics, compared it with corresponding on-premise architectures, presented implementation samples, and provided some guidance for migrating on-premise systems to cloud. Along the way, we introduced Microsoft Azure Service Bus Queue, Topic and Subscription, and Relay services. Microsoft Azure Service Bus is a very useful service for implementing many integration patterns, which we will cover in more detail in Chapter 15.

Chapter 9

High-Availability Design

In the first section of this book, I introduced some general concepts of high availability as well as what Microsoft Azure provides to achieve high availability, such as Update Domain, Fault Domain, VIP Swap, and redundancy in database services and storage services. Although Microsoft Azure provides every possible support for high availability, service developers still need to ensure that their services are designed for high availability. In this chapter, we will go through some of the practical strategies, techniques, and patterns for designing highly available services.

9.1 Availability

Availability can be viewed as the probability that a system is in a running state during a given period of time. A more strict definition refines the time period to a point where instant availability is measured. An easier way to understand the concept is via the following formula:

$$\text{Availability} = \frac{\text{Up time}}{\text{Up time} + \text{Down time}}$$

As we learned in the last chapter, redundancy improves system availability. For example, when we put two service instances with 90% of availability behind a load balancer, we can create a system with 99% of availability, as shown in Figure 9.1.

On the other hand, when we compose the components in a layered fashion, as in most systems, the overall system availability is lower than the least available components. This is very important to realize, because you can only achieve desired availability when you ramp up availabilities of ALL layers in the system. Otherwise, the layer that you miss will drag down the system availability, negating the efforts you have put in other layers. Figure 9.2 shows that when three components with 99% availability are composed together in layers, they can only provide 97% of availability.

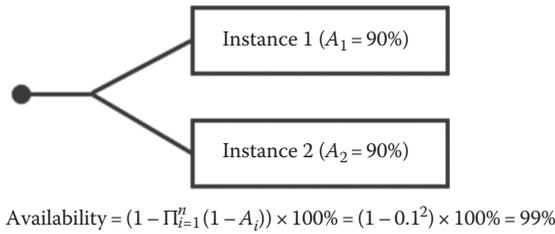


Figure 9.1 Redundancy improves availability.

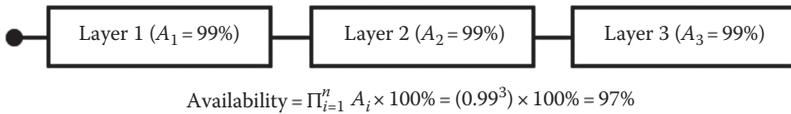


Figure 9.2 Availability is decided by lowest available component.

Table 9.1 System Availabilities

<i>Availability (%)</i>	<i>Manage Level</i>	<i>Yearly Downtime</i>	<i>FAA Ranking</i>
90	Unmanaged	5 weeks	
99	Managed	4 days	ROUTINE
99.9	Well-managed	9 h	ESSENTIAL
99.99	Fault-tolerant	1 h	
99.999	Highly available	5 min	CRITICAL
99.9999	Very highly available	30 s	
99.99999	Ultra available	3 s	SAFETY CRITICAL

Because availability is achieved by redundancy, it does not come for free. To get higher availability, you need to invest more, such as by deploying more instances. When you decide your availability goals, one question you should ask yourself is how much downtime your customers can tolerate without losing faith in your service. Table 9.1 shows how different levels of availability translate into downtime per year, with FAA ranking of system availability for your reference. If your customer would be fine with 4 days of downtime per year, it is not worth pursuing higher availability because that would just result in extra cost without added returns.

9.2 High-Availability Techniques

High availability is usually achieved by using redundancies. In addition, load-balancing and failover strategies have significant impacts on system availability.

9.2.1 Redundancy

The so-called redundancy is to create multiple copies of services and data in a system so that as long as one of the copies is working, the system remains available. Microsoft Azure provides redundancies at many different layers:

- When you create a SQL Database server, Microsoft Azure automatically maintains two hot backups. When the main database fails, Microsoft Azure automatically fails over to the backups to deliver continuous service.
- When you use storage services, your data are replicated twice using a locally redundant storage (LRS) mechanism. When some of the system components (such as data disks, servers, or server rack) fail, Microsoft Azure automatically recovers the data from backup copies. In addition, you can opt to use geo redundant storage (GRS), which replicates your data to a secondary data center that is hundreds of miles away from your main storage location. In this case, even if a disaster wipes out the entire data center, you can still recover your data from the backup location. When GRS is turned on, your data are actually saved six times: three local copies times two storage locations.
- When you use Microsoft Azure Caching service (see Chapter 11), you can turn on the high-availability option to create duplicated copies of your data in the cache cluster.
- Microsoft Azure Service Bus provides high availability by using redundancy in internal storages, utilizing temporary entities to buffer data when main entities are offline, and providing paired namespaces for cross data center failovers.

9.2.2 Load Balancing

Load balancing uses multiple service instances to share the workload of the entire system. Because the participating instances can take over each other's work, they can be considered live backups for each other. So, load balancing is a mechanism used not only for scaling but also for high availability. This is also the reason why many of Microsoft Azure's SLAs require multi-instance deployments. When some of the instances fail, the overall throughput of the system will suffer, but as long as one of the instances is still working, the system remains available. In previous chapters, we have experienced load balancing on multiple occasions:

- Multi-instance Microsoft Azure Websites (see Section 2.4).
- Multi-instance Microsoft Azure Cloud Services (see Section 3.5.2).
- Load-Balanced Microsoft Azure Virtual Machines (see Example 7.7).

9.2.3 Failover

On a system with redundancy, failover allows transactions on failing instances to be seamlessly transferred to other healthy instances. The failover process ideally is hidden from the service consumers. For these service consumers, the service remains available and functions correctly all the time.

Unfortunately, the seamless transfer is not always possible. In many cases, the client will notice that something unusual is happening, such as failed transactions, timeouts, or even duplicated transactions, and it is up to the service developer to handle these situations. Regardless, because the automated failover process restores the system to an available state much quicker than manual interventions, it is still a very effective way to improve system availability.

As an example of failover, we will present a sample scenario that uses another Microsoft Azure service—Traffic Manager. Traffic Manager allows service developers to route customer traffics to different data centers based on certain policies such as round-robin, performance-based, and failover.

Example 9.1: Traffic Manager: Cross-region failover

Difficulty: **

In this example, we will learn how to use Traffic Manager to route user traffic. For example, to better serve customers from different regions, you may want to deploy your cloud services to different geographic regions and dynamically route users to the servers that can provide them with the best performance. In this example, we will set up two deployments of a cloud service and exercise Traffic Manager policies. Readers should note that because Traffic Manager uses dynamic DNS resolution to redirect users, actual user requests do not go through Traffic Manager but directly go to the destination service, so the overhead is minimum.

When defining a Traffic Manager policy, you can define a custom probe address, which by default is the root folder of your service. Traffic Manager tries to access this probe address every 30 s. And if the probe returns an HTTP status code 200 within 10 s, Traffic Manager considers the service to be healthy. Otherwise, if Traffic Manager cannot get a positive response after four attempts, it will mark the service as offline. Because of DNS caches, a client may still be routed to an offline service even if the service has been marked offline. The client is redirected to a healthy instance only when DNS caches expire.

In addition to a round-robin policy and a performance-based policy, you can also design a policy for cross-region failover. In such a policy, you specify a priority list of service instances. Traffic Manager follows the order of the list to find the first available service instance when the service with higher priority fails. In this example, we will create a cloud service with a web role, and we will define a custom probe that can simulate a broken service instance. Then, we will observe how Traffic Manager reroutes traffic when we deliberately take one instance offline.

1. Launch Visual Studio as an administrator. Create a new cloud service with a single ASP.NET MVC 4 Web Role (using the Internet Application template).
2. Create a new empty API controller named **HealthController** under the **Controllers** folder. We will define a **Probe()** method on the controller that can configure it as our custom probe in Traffic Manager. The **Probe()** method returns either status code 200 or status code 500 based on the value of a static variable **isHealthy**. We also define a **SetHealthState** for us to flip this flag at will. The **isHealthy** flag resets itself after 5 min. The complete source code of the controller is shown in Code List 9.1.

Note: This controller is stateful, and it saves its state (**isHealthy**) in memory. Such a design is not scalable. We chose this implementation for simplicity.

3. Replace the code in **Views\Home\Index.cshtml** with the following code:

```
<h1>Health State:</h1>
<h2>
  <label id="state">Healthy</label></h2>
<input type="button" id="healthy" value="Set to Healthy" />
<input type="button" id="unhealthy" value="Set to Unhealthy" />
@section Scripts{
  <script>
```

```

    $('#healthy').click(function () {
        $.getJSON('/api/Health/SetHealthState?health=true',
            null);
        $('#state').text('Healthy');
    });
    $('#unhealthy').click(function () {
        $.getJSON('/api/Health/SetHealthState?health=false',
            null);
        $('#state').text('Unhealthy');
    });
</script>
}

```

4. Deploy this service to two different regions. In our test, we deployed the service to **example43west.cloudapp.net** (US west region) and **example43east.cloudapp.net** (US east region).

CODE LIST 9.1 HEALTH PROBE

```

public class HealthController : ApiController
{
    private static bool isHealthy = true;
    private static DateTime lastUnhealthTimestamp;
    [HttpGet]
    public HttpResponseMessage Probe()
    {
        if (isHealthy)
            return new HttpResponseMessage(HttpStatusCode.OK);
        else
        {
            if (DateTime.UtcNow - lastUnhealthTimestamp
                <= TimeSpan.FromMinutes(5))
                return new HttpResponseMessage
                    (HttpStatusCode.ServiceUnavailable);
            else
            {
                isHealthy = true;
                return new HttpResponseMessage(HttpStatusCode.OK);
            }
        }
    }
    [HttpGet]
    public void SetHealthState(bool health)
    {
        isHealthy = health;
        if (!health)
            lastUnhealthTimestamp = DateTime.UtcNow;
    }
}

```

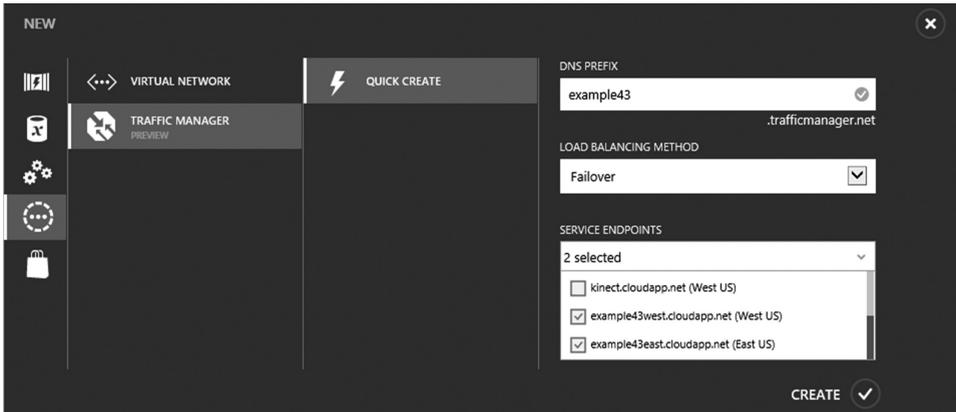


Figure 9.3 Create a new Traffic Manager policy.

5. Log in to Microsoft Azure Management Portal.
6. On the command bar, click on the **NEW** icon. Then select the **NETWORK SERVICES**→**TRAFFIC MANAGER**→**QUICK CREATE** menu.
7. Enter a **DNS PREFIX**. This will be the address you provide to your customer to access your services. Select **Failover** in the **LOAD-BALANCING METHOD** field. Then, in the **SERVICE ENDPOINTS** field, select the two service instances you have deployed in step 4. Finally, click on the **CREATE** link to create the policy (Figure 9.3).

Note: The three load-balancing methods are as follows:

- Performance. Traffic Manager periodically collects Microsoft Azure data center performance metrics around the world. When Traffic Manager receives a request from a client, it will route the traffic to the data center with best performance based on the historical performance data it collects.
- Failover. Traffic Manager periodically probes service instance health and routes traffic to the first healthy instance.
- Round-robin. Traffic Manager evenly distributes user requests to healthy service instances.

8. Switch to the **CONFIGURE** view of the Traffic Manager registration. Then, change the **RELATIVE PATH AND FILE NAME** field to **/api/Health/Probe**, which is the address of our custom probe. Also, change **DNS TIME TO LIVE (TTL)** to **30 s** (see Figure 9.4). We use a shorter TTL here so we can more easily observe how Traffic Manager behaves.
9. Click the **SAVE** icon on the command bar to save configuration changes.
10. Once the policy is created and activated, you can observe service statuses on the **ENDPOINTS** page. As shown in Figure 9.5, both service instances are online.
11. Open a browser and navigate to **http://[DNS prefix].trafficmanager.net** (in our case, the address is **http://example43.trafficmanager.net**). Now, let us simulate the service instance going down by clicking on the **Set to Unhealthy** button.
12. After a couple of minutes, refresh the **ENDPOINTS** page, and you will observe the service instance is offline, as shown Figure 9.6.
13. After a couple of minutes, you will observe the instance coming back online, as shown in Figure 9.7.

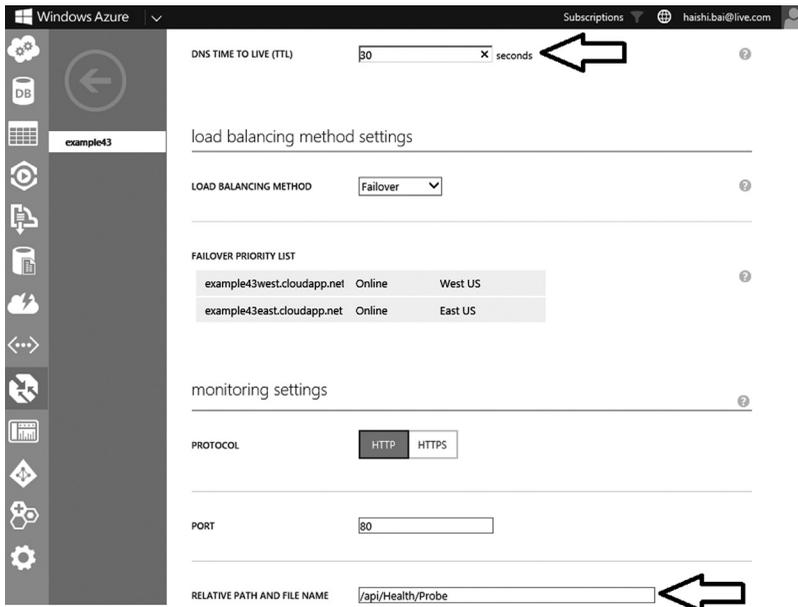


Figure 9.4 Traffic Manager Configuration.

example43 PREVIEW

ENDPOINTS CONFIGURE

NAME	STATUS	DNS NAME	LOCATION
example43west	✓ Online	example43west.cloudapp.net	West US
example43east	✓ Online	example43east.cloudapp.net	East US

Figure 9.5 Service instance statuses.

example43 PREVIEW

ENDPOINTS CONFIGURE

NAME	STATUS	DNS NAME	LOCATION
example43west	⚠ Offline	example43west.cloudapp.net	West US
example43east	✓ Online	example43east.cloudapp.net	East US

Figure 9.6 Service instance is offline.

example43 PREVIEW

 ENDPOINTS CONFIGURE

NAME	STATUS	DNS NAME	LOCATION	
example43west	✓ Online	example43west.cloudapp.net	West US	
example43east	✓ Online	example43east.cloudapp.net	East US	

Figure 9.7 Service instances are back online.

In this section, we discussed different high-availability mechanisms that Microsoft Azure provides. In the next two sections, we will refocus on cloud service and introduce techniques to improve cloud service availability. Of course, we cannot enumerate all possible techniques and patterns here. Instead, we will pick only two techniques that capture some of the essential concepts and philosophy in high-availability service design.

9.3 Load Balancing and Health Probe

As mentioned earlier, you can add multiple role instances to a load balancer to share system workload. The work distribution algorithm is often very simple, which is the round-robin method that we have mentioned multiple times in this book. However, the round-robin method has two requirements on the role instances:

- **Homogeneous**
Load balancer assumes that all instances have identical processing power and even workloads. So when distributing work, the load balancer does not need to consider the differences among the instances and simply assigns work to different instances in turn. When you deploy a Microsoft Azure Cloud Service, all instances have the identical virtual machine configuration, and run the same code; hence, they are homogeneous.
- **Autonomous**
Autonomous means the role instances do not have dependencies on each other. In other words, adding a new instance or removing an existing instance does not impact any other instances. This characteristic is very important for dynamic scaling and failure recovery. Only when the instances are autonomous, Microsoft Azure can horizontally scale instances at any time as needed.

Although the Microsoft Azure Cloud Service design provides homogeneous and autonomous role instances, it cannot stop you from writing codes that lead to interdependencies among role instances. To ensure your role can be scaled correctly, you should avoid such codes.

Another aspect of load balancing is to monitor instance health. The Microsoft Azure load balancer monitor instances status by three different mechanisms:

- The host agent running on the virtual machine host periodically sends heartbeat signals to the *WaAppAgent* process (see Section 4.4.1) running on the virtual machine. If the

WaAppAgent fails to respond to heartbeat signals within 10 min, the host agent will restart the virtual machine.

- If the role instance process has thrown an exception or has exited, the role instance is recycled.
- If the role defines a custom health probe, the probe needs to return a “ready” state when invoked. Under the TCP, the probe needs to return ACK. Under the HTTP, the probe needs to return status code 200. Otherwise, the instance is considered busy, and the load balancer will not dispatch more jobs to it.

You can define custom probes on web roles, worker roles, and virtual machines. Custom probes on web roles and worker roles are defined in the service definition (.csdef) files. Code List 9.2 shows an HTTP-based custom probe. The relative path to the probe is *Probe.aspx* in this case. The load balancer checks this path every 5 s (defined by the *intervalInSeconds* attribute), and the probe has 100 s (defined by the *timeoutInSeconds* attribute) to provide a response. In addition, the endpoint definition specifies that a custom probe (defined by the *loadBalancerProbe* attribute) should be used.

Code List 9.3 is a simple probe implementation. The sample shows that to return a “ready” state, the method simply exits (which results in an HTTP 200 response); otherwise, it throws an exception (which results in an HTTP 500 response) to indicate it is busy. Of course, this probe is not particularly useful as it randomly fails.

CODE LIST 9.2 CUSTOM PROBE

```
<LoadBalancerProbes>
  <LoadBalancerProbe name="MyProbe" protocol="http" port="80"
    path="Probe.aspx" intervalInSeconds="5"
    timeoutInSeconds="100"/>
</LoadBalancerProbes>
...
<Endpoints>
  <InputEndpoint name="Endpoint1" protocol="http" port="80"
    loadBalancerProbe="MyProbe"/>
</Endpoints>
```

CODE LIST 9.3 CUSTOM PROBE IMPLEMENTATION (PROBE.ASPX)

```
protected void Page_Load(object sender, EventArgs e)
{
    Random rand = new Random();
    if (rand.Next(0,2) == 0)
        throw new ApplicationException("Bad instance");
}
```

9.4 Competing Consumers

Now, let us switch gear to discuss a particular pattern that can help improve system availability—the Competing Consumers pattern.

The Competing Consumers pattern is a very useful design pattern. It provides many desirable attributes such as loose coupling, dynamic load balancing, dynamic scaling, and failover. In addition, it is an effective way to eliminate centralized components (see Section 8.2.3). In the next chapter, you will see how your system’s availability is decided by the lowest available component. Eliminating centralized components, which lead to poor availability, is a key step to achieving high overall system availability.

The Competing Consumers pattern is depicted in the diagram in Figure 9.8. With this pattern, the job generator and the job processors do not directly communicate with each other. Instead, the job generator adds jobs to a job queue, and multiple job processors compete for available jobs.

9.4.1 Loose Coupling

The job queue creates a loose coupling between job generators and job processors. Because there are no direct dependencies among the components, the job creators and the job processors do not need to be online at the same time, and they do not need to wait for each other. In addition, as long as the message format remains stable, a component can be replaced by other implementations without affecting other components. You can use loose coupling to construct 1-to-1, 1-to-many, many-to-1, and many-to-many topologies among the components. Typical scenarios of loose coupling include the following:

- Load balancing. You can use multiple job processors to share heavy workloads.
- Load leveling. If there are sudden spikes in the number of jobs, the job queue can serve as a buffer to release the jobs gradually to avoid overloading the processors.
- Batch processing. Because job generation and job processing can happen independently, job generators can add jobs to the job queue when job processors are offline. The job processors can come online later to process all queued jobs in batches.
- System integration. When integrating components are built by different parties, a job queue can serve as an intermediate medium that bridges different technologies and platforms.

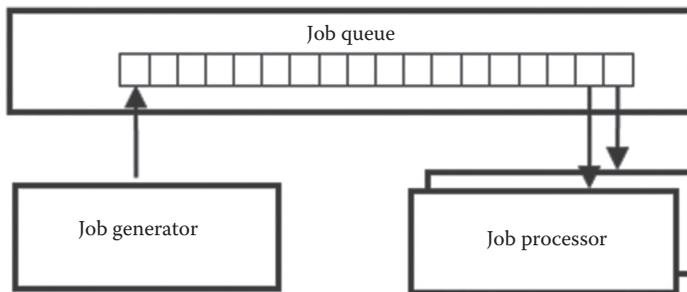


Figure 9.8 Competing Consumers pattern.

9.4.2 Dynamic Load Balancing

In Section 9.2, we introduced how the Microsoft Azure load-balancing mechanism uses round-robin to evenly distribute jobs across role instances. However, if the complexity of jobs varies greatly, it is possible that some instances get assigned with a number of complex jobs while other instances remain idle. On the contrary, with Competing Consumers pattern, a job processor gets a new job only when it is finished with its current job. On the one hand, a job processor will not get overloaded as it does not get assigned more than what it can take. On the other hand, while a job processor is busy with a complex task, other processors can go forward with other simpler tasks so that the overall workloads, instead of number of jobs, are balanced across the instances.

9.4.3 Dynamic Scaling

Because you can add or remove job processors at any time, you can dynamically adjust system throughput as needed. For instance, where there is a sudden spike in system load, you can deploy a second set of role instances to drain the job queue faster. You can even have on-premise instances that you can spin up to take some pressure off the cloud instances. Moreover, all such throughput adjustments can be done without impacting existing instances.

9.4.4 Failover

Failover is achieved by a temporary lock mechanism. When a job processor gets a job from the job queue, instead of removing the job from the queue directly, it places a temporary lock so the job becomes invisible to other job processors. The job processor is given a time window to complete the job and to mark the job as completed, which formally removes the job from the queue. If the job processor fails to complete the task within the time window, abandons the job, or crashes without marking the job as completed, the job reappears on the queue once the lock expires so that the job can be picked up by other job processors.

Such a failover mechanism ensures that a job is processed at least once, but it does not guarantee if a job is processed multiple times. For example, a job processor may have completed processing a job but crashed right before it could mark the job as completed. Then, when the lock expires, the job reappears on the queue and will be processed again.

There are different ways to deal with this situation, among which is the idempotent operation. Idempotent operations are operations that do not have accumulative effects on system states when invoked multiple times. For example, with an idempotent operation design, an online booking system can avoid duplicated bookings when a user clicks on the booking button multiple times by accident.

Note: Idempotent operations can be expressed by the formula $f(f(x)) = f(x)$.

The mechanism does not ensure strict ordering either. However, that is a general problem with parallel job processors.

Example 9.2: Compete Consumers

Difficulty: ***

In this example, we will create a Windows Console application to demonstrate dynamic load balancing with the Competing Consumers pattern. The example generates 1000 jobs with different complexities. Each of the jobs takes 100 ms to 5 s to be processed. These jobs are sent to a Service

Bus queue by a job generator. Four job processors listen to the queue and compete for new jobs. Finally, the program compares the result with round-robin load balancing.

1. Create a new Windows Console application.
2. Add a reference to **System.Configuration**. We will use the *ConfigurationManager* class to read the Service Bus connection string from the application's configuration file.
3. Add a reference to **WindowsAzure.ServiceBus** NuGet package.
4. Define a **Job** class, which will be the job message we send to job processors. The **Complexity** attribute corresponds to how much time (in milliseconds) it needs to be processed.

```
[DataContract]
class Job
{
    [DataMember]
    public int Id {get; set;}
    [DataMember]
    public int Complexity {get; set;}
}
```

5. Then, we define another class to encapsulate the state of a job processor, including the number of jobs it has handled (**JobCount**) and the accumulative processing time it has spent (**JobTime**):

```
public class WorkerInfo
{
    public int JobCount {get; set;}
    public int JobTime {get; set;}
}
```

6. Code List 9.4 shows the main method. It is a bit long but should be fairly easy to understand. You may consult the comments in the code.
7. Modify the **Microsoft.ServiceBus.ConnectionString** setting in your **App.config** file to enter the connection string of your Service Bus namespace.
8. Run the application. Figure 9.9 shows the result of one round of tests. In this test, four job processors jointly handled 1000 jobs with a total workload of 2670 s. Through parallelization, all the jobs were processed within 671 s. In the case of Competing Consumers pattern, although the processors handled different numbers of jobs, the workloads on them were roughly the same, ranging from 664 to 671 s. On the other hand, with round-robin, the processors get assigned to the exact same number of jobs, but there are greater variations in the actual workloads they handle, from 607 to 718 s. This test proves that the Competing Consumers pattern distributes the workloads more evenly among the processors.

9.5 Case Study: High-Availability Service Bus Entities

In this section, we will present a case study to show how Microsoft Azure Service Bus improves its availability by introducing redundancies and failovers at all layers of the system.

CODE LIST 9.4 THE MAIN PROGRAM

```

class Program
{
    static void Main(string[] args)
    {
        string queueName = "DemoQueue";
        //recreate DemoQueue to ensure we have a fresh start
        NamespaceManager manager = NamespaceManager
            .CreateFromConnectionString(Configuration
                Manager
                    .AppSettings["Microsoft.ServiceBus.
                        ConnectionString"]);
        if (manager.QueueExists(queueName))
            manager.DeleteQueue(queueName);
        if (!manager.QueueExists(queueName))
            manager.CreateQueue(queueName);
        QueueClient client = QueueClient
            .CreateFromConnectionString(Configuration
                Manager
                    .AppSettings["Microsoft.ServiceBus.
                        ConnectionString"],
                    queueName);
        Random rand = new Random();//to generate random processing
            time
        //initilaize test jobs
        int jobCount = 1000;//number of jobs
        int jobVariation = 5000;//processing time varies from 100ms
            to 5000ms
        long jobTotalTime = 0;//accumulated processing time
        var jobs = new Job[jobCount];
        for (int i = 0; i < jobs.Length; i++)
        {
            jobs[i] = new Job
            {
                Id = i,
                Complexity = 100 + rand.Next(0, jobVariation)
            };
            jobTotalTime += jobs[i].Complexity;
            client.Send(new BrokeredMessage(jobs[i])); //send the
                job
        }

        int workerCount = 4; //number of job processors

        int completedCount = 0;//accumulated number of completed jobs
        WorkerInfo[] workers = new WorkerInfo[workerCount];
        for (int i = 0; i < workers.Length; i++)
            workers[i] = new WorkerInfo();//initialize processor info
    }
}

```

```

Console.WriteLine(string.Format("Processing {0} jobs",
    jobCount));
Console.WriteLine(string.Format("Total workload: {0}ms",
    jobTotalTime));
for (int i = 0; i < workerCount; i++) //start process
    threads
    {
        ThreadPool.QueueUserWorkItem(new WaitCallback((obj) =>
        {
            QueueClient workerClient = QueueClient
                .CreateFromConnectionString(Configuration
                    Manager
                    .AppSettings["Microsoft.ServiceBus.
                        ConnectionString"],
                    queueName);

            while (true)
            {
                var message = workerClient.Receive();
                if (message != null)
                {
                    var job = message.GetBody<Job>();
                    workers[(int)obj].JobCount++;
                    workers[(int)obj].JobTime += job.Complexity;
                    Thread.Sleep(job.Complexity); //simulate
                        processing time
                    Interlocked.Increment(ref completedCount);
                    message.Complete(); //mark the job as completed
                }
            }
        })), i);
    }
while (completedCount < jobCount)
{
    Thread.Sleep(100);
}
Console.WriteLine();
Console.WriteLine("Competing Consumer Processing Time:"
    + workers.Max(worker => worker.JobTime));
Console.
    WriteLine("=====");
for (int i = 0; i < workers.Length; i++)
    Console.WriteLine(string.Format("Processor {0} processed
        {1}
            jobs in {2}ms",
            i, workers[i].JobCount, workers[i].JobTime));
int[] simulatedTime = new int[workerCount];
for (int i = 0; i < jobs.Length; i++)
{
    simulatedTime[i % workerCount] += jobs[i].Complexity;
}

```

```

    }
    Console.WriteLine("=====");
    Console.WriteLine();
    Console.WriteLine("Round Robin Processing Time:" +
        simulatedTime.Max());
    Console.WriteLine("=====");
    for (int i = 0; i < simulatedTime.Length; i++)
        Console.WriteLine(string.Format("Processor {0} processed
            {1}
            jobs in {2}ms",
                i, jobs.Length / workerCount, simulatedTime[i]));
    Console.WriteLine("=====");
    Console.WriteLine("Done! ");
}
}

```

```

Processing 1000 jobs
Total workload:2670331ms

Competing Consumer Processing Time:671274
=====
Processor 0 processed 257 jobs in 664018ms
Processor 1 processed 242 jobs in 671274ms
Processor 2 processed 246 jobs in 666427ms
Processor 3 processed 255 jobs in 668612ms
=====

Round Robin Processing Time:717447
=====
Processor 0 processed 250 jobs in 717447ms
Processor 1 processed 250 jobs in 685381ms
Processor 2 processed 250 jobs in 606994ms
Processor 3 processed 250 jobs in 660509ms
=====
Done!
Press any key to continue . . .

```

Figure 9.9 Result of one test run.

9.5.1 Background

Systems do not live in vacuum. Many systems need to work with other systems to accomplish complex, distributed operations. When we integrate different systems together, the most obvious way is to allow these systems to directly communicate with each other. However, as the number of systems increases, we will face two problems: first, there will be many connections to be maintained. Second, these systems are tightly coupled. For the communications to happen, they

have to be online at the same time, and their interfaces must match up. One system changing its interface requires all related systems to make corresponding changes as well, otherwise the communication will fail.

A common approach to fix these problems is to use a broker. Instead of communicating with many systems, a system only needs to talk to the broker and the broker will provide the necessary features such as reliable messaging, message transforms, dynamic routing, broadcasting, and other integration concerns.

This intermediary not only decouples the systems, but also separates integration concerns from these systems. However, because many systems rely on this intermediary, it has to be highly available so that it does not become a single point of failure of the integrated systems. Microsoft Azure Service Bus is such a broker system. In this case study, we will summarize existing and new Service Bus features that enable high-availability messaging among systems.

Service Bus is built on a Microsoft Azure platform. When users subscribe to Service Bus, they get endpoints representing the subscribed service entities, and they can use these endpoints to communicate with each other. Behind each endpoint is a Service Bus entity such as a queue or a topic, and each entity in turn uses a message broker to handle the messages, and a message store to preserve messages. So, Microsoft Azure Service Bus relies on Microsoft Azure data storage services to preserve data; it relies on Microsoft Azure computing power to host messaging logics; and it relies on Microsoft Azure data center to provide continuous service. Any of these components may fail, rendering the Service Bus unavailable. Now let us see how to address these failures at different levels.

9.5.2 Segmented Message Pipelines

Let us see a typical scenario, where a sender creates a Service Bus queue and starts to send messages to it. The queue is one of the many queues Service Bus manages at the same time. Internally, each queue is assigned to a message broker, who does the actual message handling. Up till today, a queue is assigned to a single broker, which means when a broker fails, all queues assigned to the same broker will break. With Microsoft Azure SDK 2.1, the way to work around this is to use multiple queues, so that when one queue fails, you can still use other queues to send and receive messages.

Starting with Microsoft Azure SDK 2.2, when you create a new queue, you can specify if you want to segment messages to multiple brokers. In this case, even if one or several assigned brokers fail, as long as there is at least one working broker, your queue remains available.

Figure 9.10 illustrates the difference. To the left, because a queue is assigned to a single broker, a failing broker will render all dependent queues unavailable. To the right, because the queue is supported by multiple brokers, a failing broker does not affect the availability of the queue.

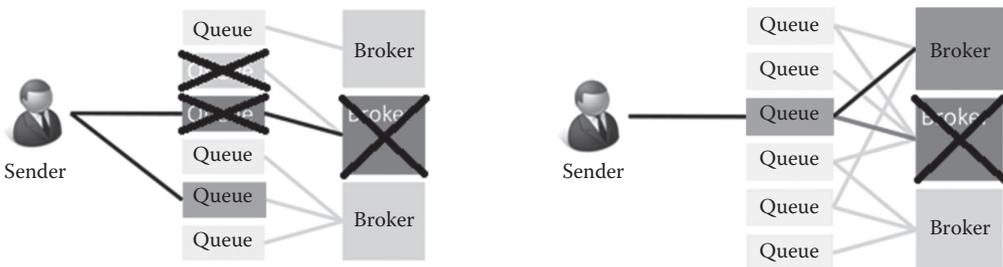


Figure 9.10 Segmented message pipelines.

9.5.3 Paired Namespaces

What if the Service Bus entity itself fails? Or even worse, what happens when the whole data center fails? This rarely happens, but Server Bus is prepared for that. To deal with such failures, you can enable Paired Namespaces, which has been included in SDK 2.1. When using Paired Namespaces, you set up a primary namespace, a secondary namespace, as well as a failover interval. You send and receive messages through the primary entity in the main namespace. But, when the primary entity fails, your messages will be forwarded to the backlog entities in the paired namespace. And later on, when your primary entity comes back online, the messages are forwarded back to the primary entity to continue flow through the pipeline.

And this is how the mechanism works:

1. Under normal circumstance, you send messages to the primary entity.
2. Service Bus runtime periodically pings your primary entity. If it cannot ping the entity, it disables the primary entity.
3. A random backlog entity in the secondary namespace is picked and messages are forwarded to these backlog entities.
4. At the same time, the sender starts to ping the primary entity to check its health.
5. One of the senders or receivers should be running a siphon, which receives messages from the backlog entities and sends them back to the primary entity when the primary entity is back online.

9.5.4 Conclusion

Service Bus provides high-availability messaging by building redundancies into the system to provide failover at different levels.

9.6 Summary

In this chapter, we focused on high-availability design. We went through some basic concepts and techniques for high availability. We learned how Microsoft Azure provides high-availability supports at different levels. We also learned how to use Traffic Manager to achieve cross-site failovers. Then, we looked deeper at the load-balancing mechanism by examining custom health probe, and two different load-balancing algorithms: round-robin and Competing Consumers. Finally, we concluded the chapter with a case study of how Microsoft Azure Service Bus improves its own availability.

Chapter 10

High-Reliability Design

The basic requirement of a software or a service is that it works as advertised, which means the software or the service delivers the required functionalities, runs without unexpected interruptions, does not destroy or lose user data, and is able to recover from failures. In this chapter, we will first review some of the basic concepts of software availability and reliability, and then discuss high-reliability design on Microsoft Azure.

10.1 Reliability, Availability, and Maintainability

Reliability, availability, maintainability, and security (RAMS) are key metrics in evaluating a system, used by many software and service customers. The reliability and the maintainability directly impact the system availability. Reliability engineering is a topic that cannot be fully covered in this book. Instead, we will try to explain the three concepts in simple language before we move on to some practical techniques.

10.1.1 Reliability

Reliability is the probability that a system functions correctly during any given period of time. Generally, we can use the following simple formula:

$$\text{Reliability} = \left(1 - \frac{\text{Failed Requests}}{\text{Total Requests}} \right) \times 100\%$$

Many matrices such as POFOD, ROCOF, MTBF, and MTTF can be used to reflect system reliability. However, we will focus on mean time to restore (MTTR) and mean time to failure (MTTF) in this book. The relationships between MTTF, MTTR, and mean time between failures (MTBF) can be depicted in Figure 10.1. In a traditional data center, MTTR is often very long, because both hardware and software recovery can take considerable amount of time.

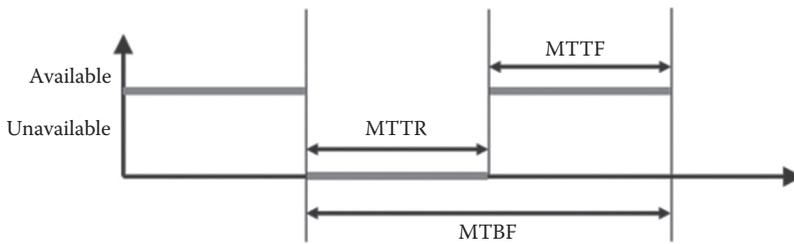


Figure 10.1 Relationships between MTTR, MTTF, and MTBF.

The long MTTR requires a long MTTF, because the running servers have to remain healthy long enough to give the system administrators enough time to fix the broken ones. The requirement of long MTTF increases the complexity of hardware design and cost. On cloud platforms, to reduce the cost, the commodity hardware servers, which do not guarantee extremely high MTTFs, are used to host cloud services. To ensure system availability, cloud platforms use a very different strategy—instead of trying to increase MTTF, it tries to reduce MTTR. It only takes a minute for Microsoft Azure to reset a virtual machine, and when a virtual machine or physical server fails completely, Microsoft Azure can simply allocate a new virtual machine or physical server from its humongous resource pool to replace the failing ones. By doing this, the MTTR is reduced to a matter of minutes, so that even with shorter MTTF, the system availability can be maintained at a high level.

10.1.2 Maintainability

Simply put, maintainability refers to the degree of difficulty to restore a system to its running state. It is rather hard to quantitatively measure maintainability as the “degree of difficulty” includes both subjective and objective aspects. The simplest way to evaluate maintainability, however, is to measure the time needed to restore a system. As a system is unavailable to provide service when it is being repaired, the longer the time for restoration, the greater the negative impacts it has on system availability.

10.1.3 Relationships between Availability, Reliability, and Maintainability

System availability is largely decided by both reliability and maintainability. However, with careful system design, the system can maintain a high availability level even when reliability and maintainability degrade. For example, a bus route can use multiple buses to provide the transportation service. When one of the buses fails, a backup bus can be dispatched to keep the route running. Although it usually takes a long time to fix a bus, the built-in redundancy keeps the bus service available. By the way, transferring passengers from the broken bus to the backup bus is a case of failover.

In Section 1.2.2 and the last chapter of this book, we discuss several design patterns and techniques to achieve high availability without improving system component reliability. In the following sections of this chapter, we focus on how to reduce the probability of system failures. System reliability is affected by many aspects across the whole service lifecycle, including requirements, architecture, coding practice, and testing. Here we will cover only a small set of selected topics.

10.2 Embracing Failures

On cloud, failures are not exceptions but the norm. In a sizable cloud-based solution, all kinds of failures may occur at any time. If there is only one sentence to be remembered out of this chapter, it would be “At any time, anything may fail; and eventually, everything will fail.” Conducting Failure Model Effects Analysis (FMEA) of all failure scenarios is one of the best practices in reliable engineering. Instead of trying to prevent errors from occurring, a more effective way is to “design for failures,” which means to prepare for various failures and build mechanisms into the system to recover from these failures. In this section, we will first categorize failures into different types, and then discuss some of the ways to deal with different types of failures.

10.2.1 Failures in Operation

When a system is in operation, typical error types include the following:

- **Transient errors**
Transient errors are caused by some temporal conditions. One key characteristic of this type of error is that it is rather unpredictable. They happen occasionally, and they may disappear even if you retried the operation milliseconds later. Any calls to Microsoft Azure services can potentially fail because of network connectivity, service throttling, and other transient states of the platform.
- **Infrastructural errors**
Cloud services run on the software and hardware infrastructure provided by Microsoft Azure. Any of these components, such as servers, racks, power supplies, and software agents, can fail. Microsoft Azure adopts Recover-Oriented Computing (ROC) principles and is able to automatically recover from most of the failures. It also provides redundancy in all layers so that services relying on these components will not be affected if some of these components fail.
- **Service errors**
These errors are caused by software bugs. These kinds of errors can be resolved only by fixing the bugs. The ease of fixing bugs and deploying a fix has significant impacts on system maintainability, and hence availability. Modularized design, testable components, effective tracing, and other engineering practices are proven techniques to raise the software quality; however, elaborate discussions on these topics are beyond the scope of this book.
- **Human errors**
Many disastrous accidents are actually caused by humans. Compared to computers, humans are very unreliable, yet humans often have control of the whole system. The most effective way to eliminate human factors is to use automation scripts. With these scripts, best practices can be captured and reliably applied by computers, which are really good at repeating predefined jobs tirelessly and precisely.

10.2.2 Failures in State Management

From the perspective of state management, failures can be put into two broad categories:

- **Data loss failures**
This type of failures destroys system states, such as losing data and breaking data integrities. Such failures often have fatal effects on a system.

- Nondata loss failures

Such failures do not cause permanent data loss, but may cause inconsistencies across the system. In a distributed system, temporary inconsistency is sometimes a design choice instead of an error (see the CAP theorem). However, there is a risk for such inconsistency to convert to data loss failures if states are reconciled incorrectly. There are many proposed solutions to maintain data consistency in a distributed system, such as distributed transactions, the Paxo algorithm, compensating transactions, etc.

10.2.3 Failures in System Design and Implementation

From the perspective of system design and implementation quality, errors appear in two different forms:

- Stably recreatable failures

This type of failures has stable recreation steps. They only occur under a specific condition, such as specific operation steps, or under certain runtime environments.

- Not stably recreatable failures

This type of failures is difficult to recreate. They are often related to threading problems, complex interdependencies, accumulative errors (such as memory leak), and scaling problems.

10.3 Transient Errors

As mentioned earlier, transient errors may happen at any time. Any service calls may fail, no matter how simple the call appears to be. For example, Code List 10.1 shows a typical Service Bus code that we have used several times. This code is actually subject to the effects of transient errors. Not only complex actions such as **CreateQueue** may fail because of service throttling or temporary network conditions, but even a simple check such as **QueueExists** may be affected by transient errors. When performing fault modeling, you need to ensure that transient errors are covered.

The appropriate way to handle transient errors is to retry the operation. Because the transient errors are temporary, retrying a couple of times is very likely to solve the problem. However, adding retrying logics around every single service call is obviously a tedious job, and the extra retry logics will lower the readability of the source code, especially when there are branches and loops. Readers can try to come up with a common library to handle transient errors, or to use the Transient Fault Handling Application Block from the Microsoft Patterns and Practices team.

Many Azure client libraries, such as Service Bus client library, have built-in retry logics. Please consult SDK documentations for methods that provide built-in transient error handling.

CODE LIST 10.1 THE CODE TO CREATE A SERVICE BUS QUEUE

```
if (!namespaceManager.QueueExists("TestQueue"))
{
    namespaceManager.CreateQueue("TestQueue");
}
```

Note: The Transient Fault Handling Application Block is an open-source library that is free for download. See the library's document on MSDN: [http://msdn.microsoft.com/en-us/library/hh680934\(v=pandp.50\).aspx](http://msdn.microsoft.com/en-us/library/hh680934(v=pandp.50).aspx) for more details.

10.3.1 Transient Fault Handling Application Block

Microsoft's Transient Fault Handling Application Block encapsulates common transient error-handling tasks. Using the library, you can easily define and apply error detection and handling strategies to improve code reliability without sacrificing readability. The application block provides built-in support for most Microsoft Azure services such as SQL Database, Service Bus, Storage services, and Cache service.

To use the application block, you need to do two things:

- Define how to detect errors. Obviously, you should not treat all exceptions as transient errors. By defining a detection policy, you can specify which exception types can be treated as transient errors. The Transient Fault Handling Application Block provides a set of pre-defined policies that include common transient error exceptions thrown by different services.
- Define how to handle errors. The application block provides several built-in retry policies, such as retry at fixed time intervals, incremental intervals, or exponential back-offs.

Now, let us learn how to use the application block with an example.

Example 10.1: Use Transient Fault Handling Application Block

Difficulty: ***

In this application, we will create a simple WPF application that sends and receives several messages using the Microsoft Azure Service Bus queue. Then, we will use the Transient Fault Handling Application Block to handle transient errors. The exact functionality of the program is not important. The focus is to demonstrate how to use the application block. Here we use a WPF application to increase sample variety, and to prepare the reader for the second section of the book, where we will look at more client-side developments:

1. Create a new WPF application.
2. Add a reference to **WindowsAzure.ServiceBus** NuGet package.
3. Modify the **MainWindow.xaml** file to create a simple UI:

```
<Window x:Class="Example10.1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
        presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Service Bus Test" Height="350" Width="525"
        FontSize="18">
  <StackPanel>
    <DockPanel>
      <Button x:Name="sendMessage" Click="sendMessage_Click_1"
              DockPanel.Dock="Right" Content="Send"/>
```

```

        <TextBox x:Name="messageText"
            HorizontalAlignment="Stretch"
                Width="Auto" />
    </DockPanel>
    <ListBox x:Name="messageList"/>
</StackPanel>
</Window>

```

4. Modify **MainWindow.xaml.cs** to add logics to send and receive messages, as shown in Code List 10.2.
5. Press **F5** to launch the application. Send several messages to ensure the application is working, as shown in Figure 10.2.

CODE LIST 10.2 LOGICS TO SEND AND RECEIVE MESSAGES

```

using Microsoft.ServiceBus.Messaging;
using System.Windows;
namespace Example10.1
{
    public partial class MainWindow : Window
    {
        const string conString = "[Service Bus Connection String]";
        const string queueName = "[Queue Name]";
        QueueClient mSender, mReceiver;

        public MainWindow()
        {
            InitializeComponent();
            mSender = QueueClient.CreateFromConnectionString
                (conString, queueName);
            mReceiver = QueueClient.CreateFromConnectionString
                (conString, queueName, ReceiveMode.
                    ReceiveAndDelete);
            mReceiver.OnMessage((m) =>
            {
                messageList.Dispatcher.Invoke(() =>
                {
                    messageList.Items.Add(m.GetBody<string>());
                });
            }, new OnMessageOptions { MaxConcurrentCalls = 1 });
        }

        private void sendMessage_Click_1(object sender,
            RoutedEventArgs e)
        {
            mSender.Send(new BrokeredMessage(messageText.Text));
            messageText.Text = "";
        }
    }
}

```

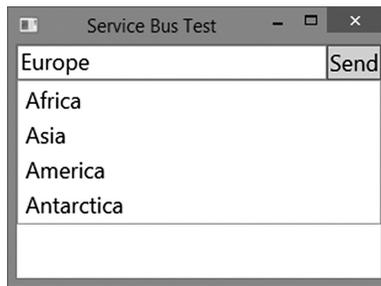


Figure 10.2 Sample test result.

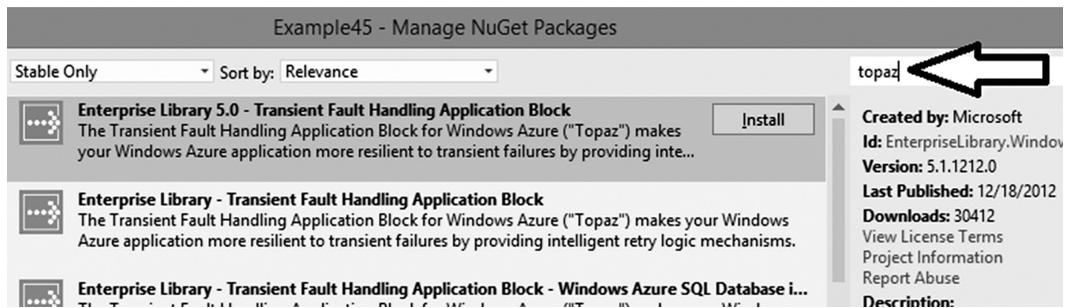


Figure 10.3 Add a reference to the Transient Fault Handling Application Block.

6. Now let us add a reference to the Transient Fault Handling Application Block. On **Manage NuGet Packages** dialog, search for **topaz**, then click the **Install** button besides **Enterprise Library 5.0—Transient Fault Handling Application Block** to install the package, as shown in Figure 10.3.
7. Back in the **MainWindow.xaml.cs** file, add several namespace imports:

```
using Microsoft.Practices.TransientFaultHandling;
using Microsoft.Practices.EnterpriseLibrary.WindowsAzure.
TransientFaultHandling;
using Microsoft.Practices.EnterpriseLibrary.WindowsAzure
    .TransientFaultHandling.ServiceBus;
```

8. Lines 3 and 4 in the Code List 10.3 define a retry policy with incremental time intervals. The policy specifies that the function call should be tried three times, and the first retry interval is 1 s. Then the retry interval is increased by 2 s before each retry. Lines 5 and 6 use the default error detection policy for Service Bus: **ServiceBusTransientErrorDetectionStrategy** to detect common Service Bus transient errors, such as **ServerBusyException**, **MessagingCommunicationException**, **TimeoutException**, etc. Lines 9–12 invoke the method we want to call (**Send()** in this case) using the retry policy. As you can see, although we used a complete retry policy, the code remains clear. Finally, we wrap everything in a

CODE LIST 10.3 MESSAGING LOGIC WITH RETRIES

```

1:private void sendMessage_Click_1(object sender, RoutedEventArgs e)
2:{
3:    var retryStrategy = new Incremental
4:        (3, TimeSpan.FromSeconds(1), TimeSpan.FromSeconds(2));
5:    var retryPolicy = new RetryPolicy
6:        <ServiceBusTransientErrorDetectionStrategy
7:            >(retryStrategy);
8:
9:    try
10:    {
11:        retryPolicy.ExecuteAction(() =>
12:            {
13:                mSender.Send(new BrokeredMessage(messageText.Text));
14:            });
15:    }
16:    catch (Exception)
17:    {
18:        MessageBox.Show("Failed to send message!");
19:    }
20:    messageText.Text = "";
21:}

```

try-catch block (lines 7–17) because all retries could fail, in which case we will inform the user that the send operation has failed.

This example shows the basic usage of the application block. The application block also has other features such as supporting asynchronous operations, and configure-driven policies, etc. You may consult MSDN documents if you are interested to learn more.

10.4 Design for Reliability

Reliability of a system is affected by many aspects. Different systems have different requirements for reliability. There are vast differences in terms of reliability in different projects. For example, the fly-by-wire (FBW) controlling system of a fighter jet usually requires triple or quadruple redundancies, with each system controlling all the mechanical surfaces independently. The controlling system of a fighter jet has to be extremely reliable because any failures may lead to severe consequences. On the other hand, the steering system of a bicycle is a single handle bar. If the handle bar is broken, the bicycle cannot be used normally any more (“normally” because many of us have seen that some skilled people can actually operate a bicycle just fine without the handle bar!). Even within the same system, reliability requirements of each component differ as well. To continue with the bicycle example, the front brake and the back brake comprise a redundant design in the braking system—not being able to start a bicycle is bad, but not being able to stop one is much worse.

Reliability engineering is an important discipline that deserves its place in books dedicated to the subject, and there have been many studies and techniques developed to improve system

reliability. For instance, reliability modeling alone has hundreds of models. In this section, we pick several topics that could be the most important and most relevant to developers.

10.4.1 *Single Point of Failure*

A Single Point of Failure (SPoF) is a component that brings down the whole system when it fails. SPoFs are not only dangerous, but also tend to become performance bottlenecks. In some cases, SPoFs can be resolved by segmenting the jobs. However, if a central component assumes to be the only instance in the system, resolving it is rather difficult. For example, a control tower of an airport assumes it has total control over all the flights and runways. It is a SPoF of the airport, but we cannot simply build more towers to resolve it, because the potential conflicts and confusions could be disastrous. Instead, we will have to build failover mechanisms so there is always one and only one tower running. However, in many systems, the seemingly necessary singleton components can actually be eliminated by some creative thinking. The following is a case study of eliminating a SPoF in a fictional scenario that is based on a real-life case.

Case Study: Eliminating a SPoF

A company provides roadside assistance to its customers. The company provides a vehicle system that automatically notifies the dispatch center of the company to provide necessary help, such as sending a tow truck or repair crew, calling the police, etc. The vehicle system also provides a user interface for other requests, such as asking for directions, looking for a restaurant or a gas station, reporting other accidents, etc. In order to provide fast responses, the company establishes a dispatch center for each of the cities it covers.

Because it was a central dispatching system, naturally the architect of the system designed a central dispatching service for the system. The central service receives requests from users, classifies them into different priorities, and then distributes these requests to a group of dispatchers. The service monitors the job queues of the dispatchers and tries to dispatch jobs to the least busy dispatcher each time to keep a fair distribution of workload. For severe incidents, a dispatcher is given 15 s to make the initial response; otherwise, the job is reassigned to another dispatcher to ensure fast responses. If no dispatchers respond to the event within 1 min, the event bubbles up to dispatch managers.

There seemed to be nothing wrong with the design. Actually, the system worked well on a local network. As the business grew, the company tried to consolidate its resources by creating super dispatcher centers that will cover many cities. This allowed them to use fewer dispatchers to serve for more customers, and to provide continuous service when customers were traveling from city to city. The company migrated the dispatching service to a cloud data center, and changed the original dispatcher clients to browser-based clients. In a small-scale trail run, everything worked fine and the company was ready to announce itself as an SaaS provider.

However, severe problems started to surface when the system was rolled out in scale. Soon they found out that the single service instance was not able to handle the workloads from multiple cities. The API was too chatty and the network was clogged. They tried to scale out the service layer. But because each of the service instances assumed they had full control of

the whole system, and a browser client was only connected to one of the instances at a given time, there were many cases when a service instance failed to find an idle dispatcher. In addition, although the clients constantly called back the service to report their connection statuses, there were still cases where an incident was assigned to an offline client. This made customers in critical incidents very angry when their requests were not handled in time. The company spent quite some time improving the service by allowing instances to synchronize states and creating externalized client state repositories. But because the fixes were put into place under great time constraint, the service became complex and buggy. Soon enough, they found that they had to make storage highly available as the storage became the new SPoF. After several serious outages, they had to revert to the smaller dispatch center configuration to keep their business. Plenty of money and business opportunities were wasted.

Switching back to the smaller dispatch center configuration gave the team some time to bring in a cloud consultant to help them to redesign the dispatching service. To their surprise, the consultant quickly proved that they did not actually need a centralized dispatching service. By using the Competing Consumer pattern, instead of a centralized service for dispatching jobs, each client queried for new jobs from a highly available job queue when it became idle. It was indeed a simple and elegant solution. First, the clients remained fully occupied without being overloaded, as they only requested for new jobs when they were able to take on more. Second, the clients did not need to send busy signals to the server anymore; they simply requested for new jobs when they were ready. Third, adding or removing a dispatch client had no effect on existing clients, so the system throughput could be adjusted at any time. Finally, by creating additional monitoring topics, the managers could monitor the whole system in real time, further ensuring the quality of the service. Yet the code was greatly simplified. Actually, the original service eventually evolved into an analytical service that provided additional business intelligence and valuable insights to further improve the service. The company eventually became a truly SaaS company selling both services and data to consumers and third parties.

10.4.2 *Writing Reliable Code*

Writing reliable code means writing a precise, clean, testable, and readable code. In this section, we discuss some of the most important practices. Obviously, this is nowhere near a complete list, and it is just a personal top-3 list. Writing a reliable code is a practice applicable not only to cloud services, but also to other application types. However, the importance of following the practice is amplified on cloud because a bug in a multitenant system impacts all customers instead of just one. Although Microsoft Azure's autorecovery mechanism can restore a failed service, frequent interruptions reduce system availability as well as customer satisfaction.

■ Clarity

Clarity does not only mean writing a precise and readable code. Clarity also means that the code has clearly defined behavior and explicit working conditions. Each method in the code defines and implements a contract. The method should work correctly when contract conditions are met and should not work at all otherwise. By “not work at all” is meant that the method should not perform any operations other than throwing an exception notifying the consumer that it has violated the contract. Consider the following method:

```
int Add(int a, int b)
{
    return a + b;
}
```

The contract it defines is simple: it takes two integers and returns the sum. However, what happens if you invoke the method with *int.MaxValue* and **1** as parameters? As an experienced C# developer, you may expect the method to simply overflow. However, as a consumer is calling a custom *Add* method instead of using the default operator, the consumer could as well be expecting the method to throw an exception when overflow happens. So, this contract, as simple as it is, has a hidden clause in it. As we write this method down, we are introducing uncertainty into the system. Now, consider the following implementation:

```
int Add(int a, int b)
{
    return checked(a + b);
}
```

This implementation turns on overflow check and throws an exception when overflow happens. This is a more explicit contract as it does not allow the consumer to assume the method to simply overflow. Now let us see a third way of implementing the method:

```
int Add(int a, int b)
{
    try
    {
        return checked(a + b);
    }
    catch
    {
        return 0;
    }
}
```

On the surface, this implementation is very friendly to its consumers. It does not overflow, it does not throw an exception. It is all quiet and peaceful. However, it is indeed a very bad implementation. It has odd behavior that few will successfully predict; it swallows exception, hiding the real cause of the potential problems; and it mixes normal conditions with exceptional conditions.

An important technique to ensure clarity in method behavior is to use defensive programming, which means a method checks for predefined conditions and only proceeds when all conditions are met. Let us examine another simple example shown in Code List 10.4.

The method first checks if the prerequisite is met—the file has to exist. Then, it moves on when the condition is met. It does not catch any exceptions because it does not handle any exceptions. The last sentence may sound awkward but it reflects a golden rule in exception handling—do not catch an exception unless you know exactly how to handle it.

CODE LIST 10.4 A METHOD TO PRINT A TEXT FILE

```

void PrintAllLinesInFile(string path)
{
    if (!File.Exists(path))
        throw new FileNotFoundException("File not found!", path);
    using (StreamReader reader = new StreamReader(File.Open(path,
        FileMode.Open)))
    {
        string line = null;
        while (!string.IsNullOrEmpty(reader.ReadLine()))
            Console.WriteLine(line);
    }
}

```

Important: Do not catch an exception unless you know exactly how to handle it. If you catch an exception only to log the error, you should rethrow the exception after the error is logged.

Appropriate language is very important for drafting a clear contract. Some junior developers do not understand the importance of good naming conventions. In Code List 10.4, it is easy to anticipate that a method with the name *PrintAllLinesInFile(string filePath)* will print all lines in a file, which is given by a file path. On the contrary, if the method is named *DoAFile(string a)*, it gives little clue for readers to understand what the method does. When we say the best documentation is the code itself, we do not just mean to add meaningful comments, but also to keep the code itself precise and readable.

- Testability

Most people will agree on the importance of software testing. However, in real-life projects, we often encounter components that are hard to test. One obvious characteristic of these components is that they rely on a complex context to function correctly. To test these components, we have to establish the context with sufficient fidelity for conducting meaningful tests. Although this is achievable by mock objects and dependency injection, such components often reflect problems in how responsibilities are assigned to different components. In a well-designed system, the dependencies among components are simple and clear. If you find two components that always need to be used as a pair, then the two components should probably be combined. Some architects prefer big components with many capabilities. Others prefer to use components with the finest granularities. Both extremes have their shortcomings. Bigger components have complex internal structures and often take on mixed responsibilities, while smaller components have convoluted dependencies among them. Striking a balance between granularity and clarity is what separates a great architect from ordinary ones.

- Maintainability

Code clarity is only one of the aspects that affect maintainability. What is even more important is that the system architecture has to be designed to embrace changes. A great architect should have deep insight into the product roadmap, and be prepared for future changes.

What we often observe in many projects is the phenomenon of architecture decay: In the early phases of a project, the architecture is solid and clear, but as the project progresses the architecture becomes unstable and may eventually collapse. There are several reasons that may cause architecture decay. For instance, the architecture may not be designed for changes. Such systems are often made of large components with rigid interdependencies among them. On the other hand, the development team may not be well trained to follow the design. However, the most important reason is that the reasoning behind the architecture is not captured and explained fully to the team. An architecture is a collection of design decisions. Capturing the reasoning behind these decisions is far more important than saving the system diagrams. Once the reasoning is understood and accepted by the team, the team will naturally keep the coherence of the architecture instead of trying to work around it.

At the same time, developers should keep an open mind to refactoring the code and avoid being overprotective toward existing work. Refactoring is not to disapprove previous work, but to adjust the software to improve clarity and performance. In many cases, refactoring does not mean the previous implementation is wrong. It simply means that a newer implementation is surfacing as requirements change or as the project progresses.

10.5 Summary

System reliability is a complex topic. Different systems have different requirements for reliability. In this chapter, we discussed a number of selected topics. First, we summarized common terms such as availability, reliability, and maintainability. Then, we analyzed different error types and discussed how to deal with them. Next, we discussed how to design for reliability by presenting some general discussions as well as a case study. Finally, we demonstrated how to write a reliable code.

Chapter 11

High-Performance Design

Performance is very important for a successful cloud service. Nowadays, users are very impatient. They expect a service to be stable and responsive whenever they use it. As the bar of entering the SaaS business is dramatically lowered by cloud platforms such as Microsoft Azure, there will be fierce competition no matter what kind of service you are providing. For each potential customer, you basically have one and only one chance to prove that your service is better than that of others—the service needs to work flawlessly at the first try. So, as a service developer, you need to make your service perform the first time, and every time. In addition, because your resource utilization is metered and has direct link to your operational cost, the ability to serve for more customers using fewer resources means higher margin and more competitive edges, which makes you more successful than your competitors.

For a software system, two key techniques to improve performance are asynchronous operations and reduction of I/O operations. This chapter will focus mostly on Microsoft Azure services, which can help you reduce I/O. General discussions of asynchronous operations and parallelization are left out as there are many books and resources on these topics.

Microsoft Azure provides two different options to create and host distributed in-memory cache clusters for your cloud services. First, you can use Microsoft Azure Cache Service, which is a Microsoft Azure hosted service providing dedicated cache clusters that you can subscribe and use. Second, you can use In-Role Cache, which is to host an in-memory cache cluster using the existing memory of the virtual machines hosting your Role instances.

Note: At the time of writing this book, there are actually three options: the Cache Service, In-Role Cache, and Shared Cache. However, Shared Cache is retiring.

11.1 Microsoft Azure In-Role Cache

In any software systems, I/O operations are much slower than CPU operations. Any opportunity to reduce I/O operations, or to make I/O operations asynchronous, can greatly improve

system performance. Among the I/O operations, external service calls and disk operations are often the slowest. Microsoft Azure Cache service allows you to keep frequently used data in memory so that you can use low-latency memory I/O to replace expensive disk I/O and external service calls.

11.1.1 Overview

Microsoft Azure In-Role Cache allows you to use the existing memory on your virtual machines to create and host distributed, high-performance, in-memory cache clusters. The In-Role Cache has the following features:

- **No throttling**
Because the cache cluster runs on the same virtual machines that host your Role instances, it is dedicated to your Role instances. Your service calls are not subject to throttling, which is common when calling SaaS services. There is also no maximum size of the cache. You can create a cache cluster of any size as long as your virtual machines have enough memory to support it.
- **Security**
Because your Role instances have exclusive access to the cache cluster, the cached data are visible only within the scope of your cloud service.
- **Low cost**
Because the cache cluster is not an external service, you are not charged any transaction fee. In addition, because the cache cluster utilizes the memory you have already paid for, there is no extra cost of hosting and storing cached data.
- **Scalability**
You can adjust your cache settings at any time to fine-tune the performance of the cache cluster. When you use dedicated caching roles (see next section), you can infinitely scale out the cluster.
- **Integration with Visual Studio**
It is very easy to enable and configure caching using Visual Studio. The Microsoft Azure Compute Emulator also provides a high-fidelity simulation of the service, making development and diagnostics much easier.
- **Support Memcache Protocol**
Microsoft Azure Cache supports both Memcache text protocol and Memcache binary protocol.
- **Rich feature set**
Microsoft Azure Cache service supports a rich set of features, which we will introduce in Section 11.1.3. In addition, Microsoft Azure Cache service API is very similar to Microsoft AppFabric 1.1 API on Windows Servers, simplifying application migrations.
- **High performance**
Because Microsoft Azure keeps Role instances close to each other in a data center, the network latency among the instances is minimum. A memory cluster hosted on the same Role instance provides even better performance.

11.1.2 Deployment Options

You have two options to deploy an In-Role cache cluster.

■ In-Role

This option allocates a certain amount of memory on an existing virtual machine to be used as the cache cluster. When you define a Web Role or a Worker Role, you can specify it to allocate a certain amount (such as 30%) of the virtual machine memory to be used by the cache cluster. When your service is deployed, Microsoft Azure will organize the allocated memory into a distributed cluster that can be accessed by all Role instances within the same cloud service. For example, let us say there is a cloud service with two Role instances running on small virtual machines, which have 1.75G memory each. The developer has specified to use 30% of memory as cache. Then the two instances give you a 1.05G cache cluster ($1.75\text{G} \times 30\% \times 2$).

■ Dedicated Role

You can also add a dedicated caching Role into your cloud service, in which case the corresponding virtual machines will be dedicated to host and run the cache cluster. This option allows you to create bigger cache clusters and achieve more stable performance as the cache cluster is not competing for resources with other Role logics you have defined on your Worker or Web Roles.

Microsoft Azure Cache service uses a simple key-value dictionary as data structure, so the cache operation is very simple. Now let us learn how to create and use In-Role Cache with an example.

Example 11.1: Use Microsoft Azure In-Role Cache

Difficulty: *

In this example, we create a simple cloud service with a single Web Role. We also reserve a certain amount of memory to be used as a cache cluster.

1. Launch Visual Studio as an administrator. Create a new cloud service with a single ASP.NET MVC 4 Web Role (using the Empty template).
2. Add a new empty MVC Controller named **HomeController** under the **Controllers** folder.
3. Create a new **Home** folder under the **Views** folder. Then add a new **Index.cshtml** View under the **Home** folder.
4. In the cloud service project, double click on the Web Role to open its Properties page. Switch to the **Cache** tab. Check the **Enable Caching** checkbox. Then, under the **Cache Cluster Settings** section, specify **Cache Size** to be 30% (of the memory on the hosting virtual machine), as shown in Figure 11.1.
5. Save the configuration. Now, when your service is deployed, you will have a distributed cluster at your disposal. That was easy, wasn't it?
6. To access the cache cluster, you will need to get a client library, which is delivered as a **Microsoft.WindowsAzure.Caching** NuGet package (there are several packages with similar names, so ensure you pick the right one).
7. The NuGet package modifies the **Web.config** file automatically. However, we need to edit the file to point the client to the cache cluster, which is named after the Role it belongs to. For example, if the name of our Web Role is **MvcWebRole1** (which is default), then the In-Role cache cluster is identified by the same name. Open the **Web.config** file, and modify the **identifier** attribute of the **dataCacheClients\dataCacheClient\autoDiscover** element to the name of your Web Role:

```
...
<dataCacheClients>
  <dataCacheClient name="default">
```

```
<autoDiscover isEnabled="true" identifier="MvcWebRole1" />
...
</dataCacheClient>
</dataCacheClients>
```

8. Save the **Web.Config** file.
9. Modify the **Index** method of **HomeController** as shown in Code List 11.1. To access the cache cluster, we first need to create a new **DataCache** instance (line 9). Then, we can use its **Get** method to check if the cache contains an item named "LaunchTime" (line 10). If we cannot find the item, we will record current UTC time into the cache by calling the **Add** method (lines 11–15; you can also use **Put** method to update/insert an item). Finally, we save the same information to **ViewBag** to be displayed on UI (line 16).
10. Modify **index.cshtml**. We only need one line of code:

```
<h2>The site was launched at: @ViewBag.LaunchTime</h2>
```

11. Press **F5** to launch the application. You can see the timestamp when the first instance of the Web Role is started, as shown in Figure 11.2.
12. If you refresh the browser page, the timestamp will not change, because after the initial writing, we are reading it back from the cache. However, if you keep the program running for more than 10 min and refresh the page again, you will see that the timestamp actually changes! This is because by default the items you put in the cache only live for 10 min. After 10 min, the item is removed and the controller records a new time. In the next section, we will describe how to fix this by using **Expiration Type** and **Time to Live** settings.

In the previous example, we used the In-Role deployment option. If you want to use a dedicated role, you can add a **Cache Worker Role** to your cloud service, as shown in Figure 11.3. The way to configure and use cache clusters on dedicated roles is the same as for In-Role deployment.

Configuration Service Configuration: All Configurations

Settings

Endpoints Enables an in-memory distributed cache cluster that is hosted across instances of your role. This cache can be programmatically a application for fast storage and retrieval of data.

Local Storage Enable Caching

Certificates Cache Cluster Settings

Caching Specify how the role hosts the cache to set an appropriate cache size. The cache size sets the maximum amount of memory tha cluster on instances of the virtual machine.

Co-located Role

Cache Size (%): [Learn about setting the cache size](#)

Dedjicated Role

Specify the storage account credentials to use for maintaining the cache cluster's runtime state.

Named Cache Settings

The cache cluster can support multiple logical named caches with individual policies. Add caches and manage their settings be

[Add Named Cache](#) Remove Named Cache

Name	High Availability	Notifications	Eviction Policy	Expiration Type	Time To Live (min)
default	<input type="checkbox"/>	<input type="checkbox"/>	LRU	Absolute	10

Figure 11.1 Enabling In-Role Cache.

CODE LIST 11.1 MODIFIED HOMECONTROLLER

```

1: using Microsoft.ApplicationServer.Caching;
2: ...
3: namespace MvcWebRole1.Controllers
4: {
5:     public class HomeController : Controller
6:     {
7:         public ActionResult Index()
8:         {
9:             DataCache cache = new DataCache();
10:            var timestamp = cache.Get("LaunchTime");
11:            if (timestamp == null)
12:            {
13:                timestamp = DateTime.UtcNow;
14:                cache.Add("LaunchTime", timestamp);
15:            }
16:            ViewBag.LaunchTime = timestamp;
17:            return View();
18:        }
19:    }
20: }

```



Figure 11.2 Last launch time.

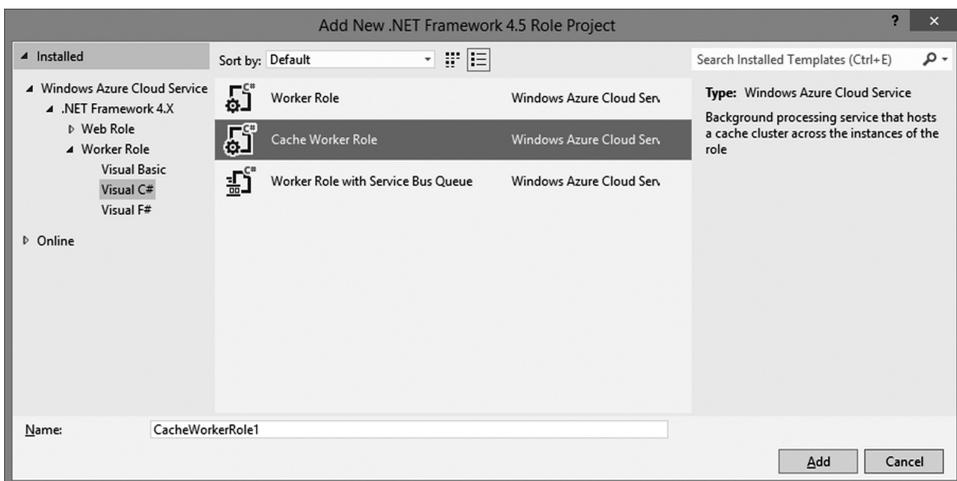


Figure 11.3 Add a Cache Worker Role.

Named Cache Settings

The cache cluster can support multiple logical named caches with individual policies. Add caches and manage their settings below.

[+ Add Named Cache](#) [✕ Remove Named Cache](#)

Name	High Availability	Notifications	Eviction Policy	Expiration Type	Time To Live (min)
default	<input type="checkbox"/>	<input type="checkbox"/>	LRU	Absolute	10

Figure 11.4 Named Cache Settings.

11.1.3 Cache Features

Some readers may have noticed that on the caching tab in Figure 11.1, there is a Named Cache Settings section, which is zoomed in and shown in Figure 11.4.

Each of the settings is explained as follows:

- **Named caches**

You can create multiple named caches within a cache cluster. You can manage and configure named caches separately for different purposes. For instance, you can create a named cache for each of your customers to manage their cached data separately. By default, when you create a new cache cluster, a named cache with the name “default” is created for you. You can create additional named caches by clicking on the **Add Named Cache** button. To access the named cache other than the default cache, you need to use an overloaded constructor of **DataCache** to provide the name of the cache, for instance:

```
DataCache cache = new DataCache("NamedCache1");
```

- **High availability**

Your cached items are distributed across the instances that run the cache cluster. When an instance is recycled, the cached items on that instance will be lost. To improve availability of cached data, you can turn on the **High Availability** option to ensure each item is replicated on two different instances. Then, when one instance crashes, you can still access the cached data from the other instance. Of course, because data are written twice in this case, each item takes twice the storage space to be stored.

- **Notifications**

You can respond to cache cluster events, such as items being added or removed, by registering callbacks. For example, the following code shows how to respond to the event when a new item is added:

```
cache.AddCacheLevelCallback(DataCacheOperations.AddItem,
    (cacheName, region, key, version, operation, descriptor) =>
    {
        //access event parameters such as cache name and
        key here.
    });
```

By default, the client library polls for notifications every 5 min. To make this polling interval smaller, you need to use a **DataCacheFactory** with custom **DataCacheFactoryConfiguration** to create the **DataCache** instance. The following code changes the notification polling interval to 5 s so that you can observe that the callback is called sooner (you can also do this via configuration—see Code List 11.4).

```
DataCacheFactoryConfiguration configuration = new DataCache
    FactoryConfiguration();
configuration.NotificationProperties =
    new DataCacheNotificationProperties(1000, new TimeSpan
        (0, 0, 5));
DataCacheFactory factory = new DataCacheFactory(configuration);
DataCache cache = factory.GetDefaultCache();
```

■ Eviction policy

To keep the cache cluster running smoothly, you can instruct the caching service to remove the least recently used (LRU) items when the available memory space reaches a low level. You can change the Eviction policy to **None** to disable eviction. When eviction is disabled, the cache may be filled up, and you may receive an exception when trying to add new items.

■ Expiration types and time to live

When you use the **Add** method or the **Put** method to add an item to the cache cluster, you can specify a time-to-live (TTL) value that controls when the item expires. Expired items will be automatically removed from the cache. If you do not specify a TTL, the lifetime of the item is decided by the **Expiration Type** setting and the **Time to Live** setting on the named cache. Expiration types include the following:

- **None**: The item does not expire. It resides in the cache until it is evicted.
- **Absolute**: The item expires after the TTL since it is added.
- **Sliding window**: The item expires after the TTL since it is last accessed. This option allows frequently accessed data to live longer in the cache.

■ Regions and tags

Within a named cache, you can create multiple regions to segment cached data. You can also group cached items by attaching tags onto them. Then, you will be able to use methods such as **GetObjectsByTag** and **GetObjectsInRegion** to retrieve these items. When using regions, you should pick smaller regions for more even distribution of items because all cached items within the same region are saved on the same virtual machine (if High Availability is enabled, the whole region is replicated to a second machine).

Code List 11.2 shows an example of using regions and tags. Line 2 of the code creates a new **ImportantItems** region. Then, lines 3–6 create a tag list. Line 7 adds a new item to the new region, with the tag list attached to the item.

11.1.4 Concurrency Modes

Microsoft Azure Cache service supports both optimistic concurrency and pessimistic concurrency.

■ Optimistic concurrency

Optimistic concurrency detects conflicts by checking item versions. For example, Figure 11.5 shows how an item is updated by multiple clients. The initial version of the

CODE LIST 11.2 USE REGIONS AND TAGS

```

1: DataCache cache = new DataCache();
2: cache.CreateRegion("ImportantItems");
3: List<DataCacheTag> tags = new List<DataCacheTag>()
4: {
5:     new DataCacheTag("aTag")
6: };
7: cache.Put("key1", "value1", tags, "ImportantItems");

```

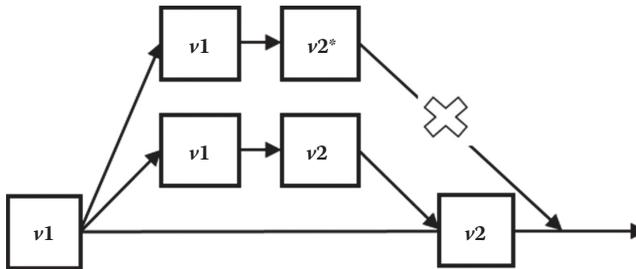


Figure 11.5 Optimistic concurrency.

item is $v1$. Then, two clients read and update the time at the same time. One of the clients updates the item from $v1$ to $v2$, and successfully writes $v2$ back to the cache. At the same time, the other client updates the item from $v1$ to $v2^*$. Then, when it tries to write $v2^*$ back to the cache, the caching service detects that the latest version of the item has been updated to $v2$, which differs from the version the client originally read ($v1$). Hence, the update from the second client is rejected.

■ Pessimistic concurrency

Pessimistic concurrency avoids conflicts by placing exclusive locks on items. A client places a lock on the item it needs to update, and it unlocks the item when the update is done. A locked item is inaccessible by other clients, and it will not expire either (but it could expire right after it has been unlocked). Code List 11.3 shows an example of pessimistic concurrency. Line 4 places a 20 s lock on the item to be updated, and line 6 updates and unlocks the item.

CODE LIST 11.3 PESSIMISTIC CONCURRENCY

```

1: DataCache cache = new DataCache();
2: ...
3: DataCacheLockHandle lockHandle;
4: var item = cache.GetAndLock("akey", TimeSpan.FromSeconds(20), out
    lockHandle);
5: //update item
6: cache.PutAndUnlock("akey", item, lockHandle);

```

11.1.5 Local Cache

When a client requests a cached item, the caching service needs to first determine the virtual machine on which the item is located. Then, it reads the item from the corresponding machine and sends the item back to the requesting client. This is a multistep operation with potentially multiple network transportations involved. To further improve the performance, the client can enable local cache, which uses the local memory used by the client process. Obviously, you can get the best performance with local cache (submillisecond access time).

When local cache is used, the data in the local cache need to be synced with the data in the cache cluster. You can specify a fixed interval to refresh your local cache. You can also refresh your local cache when certain events are triggered. When you use the second method, you also need to specify how frequently the client should poll for events (like what we have done in the Notifications example). In addition, you can specify item TTL in the local cache. When the item expires in the local cache, it needs to be retrieved again from the cache cluster when it is accessed again. Finally, when the number of items in the local cache reaches a certain threshold, the local cache will be automatically refreshed.

Code List 11.4 shows an example of how the local cache is configured. This configuration specifies that the local cache is refreshed based on cache cluster events (**NotificationBased**). The maximum item count is 100,000 (**objectCount**). The item TTL is 300 s (**ttlValue**). The client should poll for events every 60 s (**pollInterval**).

11.1.6 Session State

Stateless service design is an important way to design scalable services. If a service is statefull, all requests from the same user session have to be handled by the same service instance. Here is a simple example: An online shopping service runs on two servers. When a user accesses the service, he or she is randomly directed to one of the servers—let us say **server 1** in this case (step 1 in Figure 11.6). Then, the user requests to add an item, **item 1**, into the shopping cart. **Server 1** detects that it does not have a shopping cart for the user yet, so it creates one and puts the **item 1** into the cart (step 2 in Figure 11.6). The user continues to shop, and tries to add another item, **item 2**, to the cart. This request happens to be routed to **server 2** in this case (step 3 in Figure 11.6). Similarly, **server 2** finds out that it does not have a cart for the user yet, so it creates a new cart and puts **item 2** into the cart (step 4 in Figure 11.6). Now, the user has two instances of shopping carts with different contents on the two servers. When the user tries

CODE LIST 11.4 CONFIGURING LOCAL CACHE

```
<dataCacheClients>
  <dataCacheClient name="default">
    <autoDiscover isEnabled="true" identifier="MvcWebRole1" />
    <localCache isEnabled="true" sync="NotificationBased"
      objectCount="100000" ttlValue="300" />
    <clientNotification pollInterval="60"/>
  </dataCacheClient>
</dataCacheClients>
```

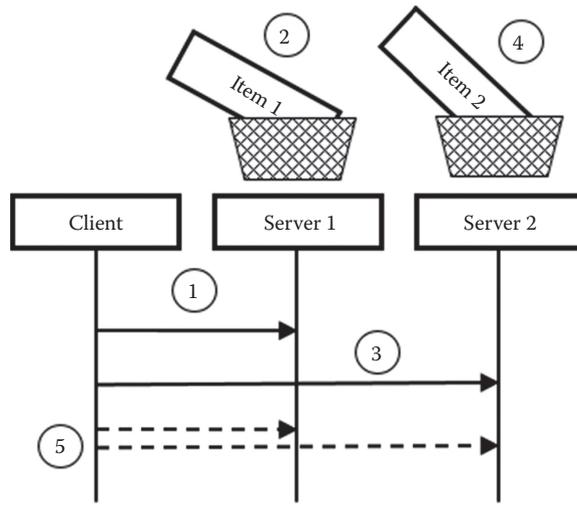


Figure 11.6 State management and load balancing.

to access the shopping cart again (step 5 in Figure 11.6), he or she might encounter either of the two states, and neither is correct.

There are two basic ways to solve this problem. The first is to use sticky sessions, which means that all requests from the same user session are handled by the same server instance. Although you can achieve sticky sessions using Application Request Routing (ARR) on Microsoft Azure, a better way is to externalize session states, as shown in Figure 11.7.

Microsoft Azure Cache service provides a built-in Session State Provider, which can be used by your ASP.NET applications to save session states to a cache cluster. Now, let us learn how to use this state provider with an example.

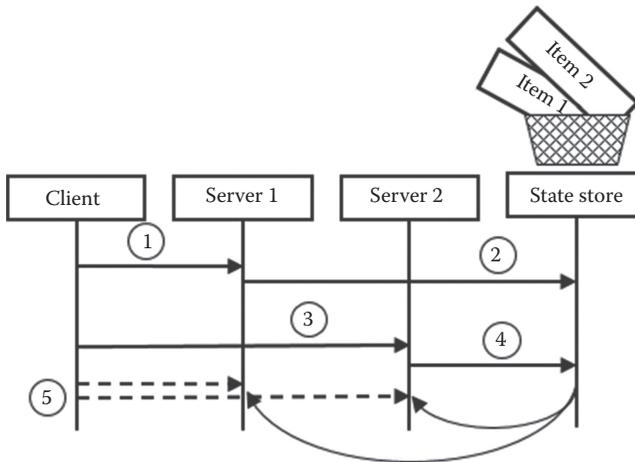


Figure 11.7 Externalized state.

Example 11.2: Use external Session State Provider

Difficulty: **

In this example, we create a simple online shopping application. First, we use a local state to demonstrate the problem shown in Figure 11.6, and then, we use a cache-based Session State Provider to solve the problem.

1. Launch Visual Studio as an administrator. Create a new cloud service with a single ASP.NET MVC 4 Web Role (using the Internet Application template).
2. Modify the **HomeController** to add simple online shopping logics. Line 3 in Code List 11.5 turns off ASP.NET caching for the convenience of testing. Lines 6–11 indicate the modified Index method. If the passed-in *item* parameter is not null, the method appends the item to a string list and then saves the string list to a session variable named “cart.” Finally, the method returns existing items as a string list to UI (line 10).
3. Modify **View\Home\Index.cshtml** as shown in Code List 11.6. Line 1 declares that the business model to be bound to UI is a list of strings. Lines 2–8 comprise the list of available

CODE LIST 11.5 HOMECONTROLLER

```

1: namespace MvcWebRole1.Controllers
2: {
3:     [OutputCache(NoStore = true, Duration = 0)]
4:     public class HomeController : Controller
5:     {
6:         public ActionResult Index(string item)
7:         {
8:             if (!string.IsNullOrEmpty(item))
9:                 addItem(item);
10:            return View(getItems());
11:        }
12:        private void addItem(string item)
13:        {
14:            if (Session["cart"] != null)
15:            {
16:                List<string> items = getItems();
17:                items.Add(item);
18:                Session["cart"] = items;
19:            }
20:            else
21:                Session["cart"] = new List<string> { item };
22:        }
23:        private List<string> getItems()
24:        {
25:            if (Session["cart"] != null)
26:            {
27:                return (List<string>)Session["cart"];
28:            }
29:            else
30:                return new List<string>();
31:        }
32:    }
33: }

```

CODE LIST 11.6 ONLINE SHOPPING UI

```

1: @model List<string>
2: <h2>Please choose the merchandise you want to purchase:</h2>
3: <select id="items">
4:   <option value="Xbox 360">Xbox 360</option>
5:   <option value="Xbox One">Xbox One</option>
6:   <option value="PlayStation">PlayStation</option>
7:   <option value="Wii">Wii</option>
8: </select>
9: <input type="button" value="Purchase" id="purchase" /><br />
10: <br />
11: <h2>Your shopping cart:</h2>
12: <div id="shoppingcart">
13:   @foreach (var item in Model)
14:   {
15:     <span style="font-size: 18px; margin: 5px;">[@item]</
      span>
16:   }
17: </div>
18:
19: @section Scripts{
20:   <script>
21:     $('#purchase').click(function () {
22:       $.ajax(
23:         {
24:           url: '?item=' + encodeURIComponent(items.
             value),
25:           type: 'GET',
26:           success: function (result) {
27:             window.location = "/";
28:           }
29:         }
30:       );
31:     });
32:   </script>
33: }

```

merchandise. Line 9 renders the purchase button, which triggers the JavaScript function in lines 21–30 to add the selected item to the shopping cart. The shopping cart is displayed in lines 11–17.

4. Modify the Web Role property to increase the instance count to 2.
5. Press **F5** to launch the application. Randomly “purchase” some merchandise, as shown in Figure 11.8.
6. As you purchase more items, the contents in your shopping cart may suddenly change. This is because requests are sent to different service instances, which hold different copies of the shopping cart in their memory.
7. Stop the application.
8. Modify the Web Role property to enable caching and allocate 30% of the virtual machine memory as a cache cluster.
9. Save the settings.

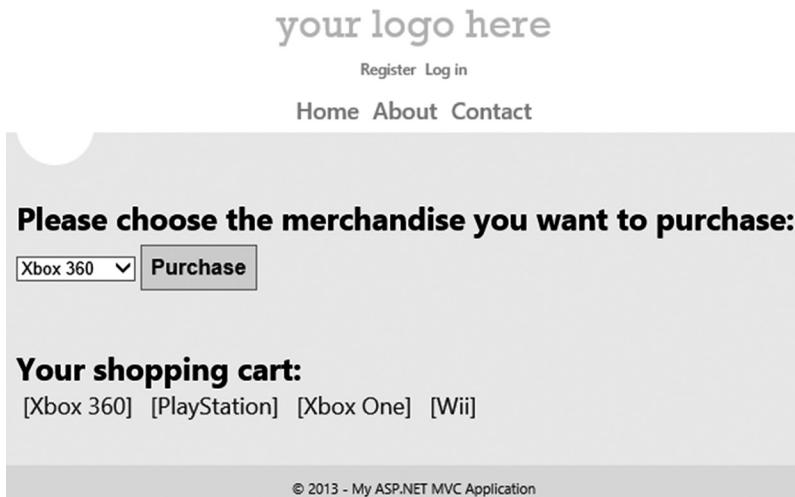


Figure 11.8 Online shopping program.

10. In the Web Role project, add a reference to the **Microsoft.WindowsAzure.Caching** NuGet package.
11. Modify **web.config** to enter the Web Role name as the cache cluster identifier (see Example 11.1).
12. The NuGet package has already added a cache-based Session State Provider to the **web.config** file as comments. Uncomment the **<sessionState>** element and then save the **web.config** file.

```

<!-- Microsoft Azure Caching session state provider -->
<sessionState mode="Custom" customProvider="AFCacheSessionState
  Provider">
  <providers>
    <add name="AFCacheSessionStateProvider"
      type="Microsoft.Web.DistributedCache
        .DistributedCacheSessionStateStoreProvider,
        Microsoft.Web.DistributedCache"
      cacheName="default "
      dataCacheClientName="default "
      applicationName="AFCacheSessionState"/>
  </providers>
</sessionState>

```

13. Run the application again. This time, the shopping cart state is stable.
14. *[Optional]* To ensure high availability of session data, turn on the High Availability option of the cache.

11.2 Microsoft Azure Cache Service

Microsoft Azure Cache service is an SaaS provided by Microsoft Azure. You can create and manage high-throughput, low-latency, dedicated cache clusters to make your services more responsive.

11.2.1 Overview

Microsoft Azure Cache Service allows you to create high-performance cache clusters for your Microsoft Azure Websites, Web Roles, Worker Roles, and Virtual Machines. At the time of writing this book, the service is in preview and provides the following three different cluster options:

- Basic: Shared cache clusters in sizes from 128 MB to 1 GB
- Standard: Dedicated cache clusters in sizes from 1 GB to 10 GB
- Premium: Dedicated cache clusters in sizes from 5 GB to 150 GB

11.2.2 Cache Service versus In-Role Cache

Microsoft Azure Cache Service and In-Role Cache provide similar functionalities and follow the same programming model. Because there are no quotas or throttling in either option, you do not need to consider different usage patterns when choosing between the two options. However, they are different in several aspects:

- Hosted versus self-managed
Microsoft Azure Cache clusters are hosted by Microsoft Azure, while In-Role clusters are hosted by your service Roles. Microsoft Azure Cache clusters can be deployed and scaled independently from your service instances, while In-Role clusters scale together with your Roles. Microsoft Azure Cache Service ensures high availability of cache clusters it provides, while you have to manage the availability of In-Role clusters yourself, such as by using multiple instances and enabling high-availability options.
- Public versus private
Microsoft Azure Cache clusters are publicly accessible by authenticated users, while In-Role clusters are only accessible within the hosting cloud service. Multiple cloud services or other applications can share the same Microsoft Azure Cache cluster, enabling more collaboration scenarios.

11.2.3 Managing Cache Clusters on Microsoft Azure Management Portal

You can manage your cache clusters easily on Microsoft Azure Management Portal. To create a new cluster, simply click on the **NEW** icon on the command bar, and then select **DATA SERVICES**→**CACHE**→**QUICK CREATE** to create a new cluster. Pick endpoint name, region, cache offering, as well as how much memory you actually plan to use, and then click the check icon to create the cluster, as shown in Figure 11.9.

Once the cluster is created, you can configure it and monitor its status on the portal. For instance, you can get the cluster's access key by clicking on the **MANAGE KEYS** icon on the command bar. Then, you should enter the key to your configuration file, as shown in the following example. Because the cache cluster is publicly accessible, you have to enable the **securityProperties** element and provide your authentication key. In addition, note that the identifier of the cluster is the full endpoint URL instead of a Role name.

```
<dataCacheClients>
  <dataCacheClient name="default">
```

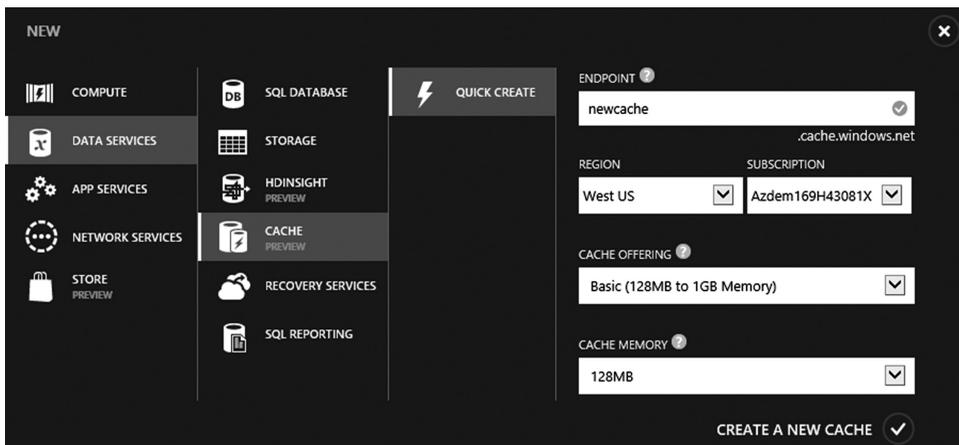


Figure 11.9 Create a new cache cluster.

```
<autoDiscover isEnabled="true" identifier="mycache.cache.windows.net" />
<securityProperties mode="Message" sslEnabled="false">
  <messageSecurity authorizationInfo="[Authentication Key]" />
</securityProperties-->
</dataCacheClient>
</dataCacheClients>
```

11.2.4 Memcache Support

Both Microsoft Azure Cache Service and In-Role cache support Memcache protocol. You can enable Memcache support in two different ways: by using a server gateway or a client-side shim. Both of them translate between the Memcache protocol and the proprietary protocol used by Microsoft Azure caching services. At the time of writing this book, the server gateway has not yet been enabled. You will need to install the client shim for your Memcache clients. The shim is delivered as a NuGet package (Microsoft Azure Caching Memcache Shim). Once installed, it creates a local service on your hosting virtual machine that exposes a Memcache-compatible endpoint (with default address of 127.0.0.1:11211) that your Memcache client can connect to.

Note: The author has created a complete walkthrough of how to use Microsoft Azure Cache to save Java Servlet session states running on a Tomcat application server. You can find this two-part walkthrough on the following blog:

Part 1: <http://haishibai.blogspot.com/2013/09/use-tomcat-with-windows-azure-caching.html>.

Part 2: http://haishibai.blogspot.com/2013/09/use-tomcat-with-windows-azure-caching_3.html.

11.2.5 Future of Azure Cache

At the time when this book is being written, Azure Cache Service is getting ready for the next iteration. A new caching service based on Redis will be provided. Microsoft Azure automates cache hosting for you, and you access the cache cluster just as accessing any other Redis cache clusters. Redis provides many desired features such as data persistence and high-level data structures. A detailed introduction of Redis is out of the scope of this book.

11.3 Microsoft Azure CDN

Microsoft Azure Content Delivery Network (CDN) is another SaaS to help you improve your service performance. It caches your website contents on strategically located cache nodes around the global to speed up delivery of the contents to your customers.

CDN improves your system performance in two different ways. First, because CDN can serve up contents from its cache nodes that are closer to the final users, it can provide better response time compared to the case when contents are directly served from your application servers. For example, your service may have been deployed in the US West data center. When a user from the east coast tries to access the site, many of the contents, such as JavaScript files, images, styles, and static pages, can be delivered directed by the CDN nodes on the east coast instead of being retrieved from the west coast servers. Second, CDN can also take some workload off your application servers. Because users can request not only static contents, but also cached dynamic page outputs from the CDN nodes, many requests can be satisfied by the CDN nodes instead of your application servers.

Example 11.3: Use CDN to distribute contents in your BLOB storage

Difficulty: **

In this example, we enable CDN for one of our Microsoft Azure Storage accounts. We assume you have already created a storage account with a BLOB container, which holds a number of pictures. See Section 6.3 for details on how to use BLOB storage service.

1. Log in to Microsoft Azure Management Portal.
2. Click the NEW icon on the command bar. Then, select **APP SERVICES**→**CDN**→**QUICK CREATE**. In the **ORIGIN DOMAIN** field, you can select a storage account endpoint to cache contents from your storage account, or **cloud services** to cache static contents and page outputs from your cloud services. Here we will pick a storage account. Click on the **CREATE** link to create the CDN endpoint (Figure 11.10).

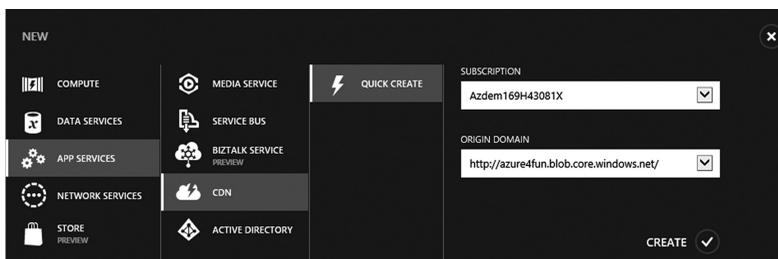
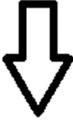


Figure 11.10 Creating a new CDN endpoint.

cdn



NAME	STATUS	SUBSCRIPTION	URL	ORIGIN DOMAIN	
az495340	✓ Enabled	Azdem169H43081X	http://az495340.vo.msecnd.net/	http://azure4fun.blob.core.windo...	

Figure 11.11 CDN URL on the portal.

- Once the CDN endpoint is created, you can observe its endpoint URL on the portal. Now, you can use URL `http://[CDN endpoint]/[BLOB container]/[BLOB]` to access blobs in your BLOB container (it may take several minutes before the contents can be accessed by the CDN endpoint) (Figure 11.11).

As mentioned in the previous example, in addition to caching for storage accounts, you can use CDN to cache cloud service contents and outputs. For example, let us assume you have a cloud service named **myservice**, whose endpoint is `http://myservice.cloudapp.net/`. You have created a CDN endpoint for this service, with the address `http://somevalue.vo.msecnd.net/`. Then, all files under your cloud service are accessible by replacing the host in their URLs with the CDN host name. For instance, a script file under your cloud service root folder, `scripts/myscript.js`, can be accessed by the address `http://somevalue.vo.msecnd.net/scripts/myscript.js`. An ASP.NET file, `pages/mypage.aspx`, can be accessed by the address `http://somevalue.vo.msecnd.net/pages/mypage.aspx`. Note that in this case, the user is not invoking the page, but is accessing cached output from the caching node. Furthermore, you can enable caching by query strings (you can enable this option by clicking on the **ENABLE QUERY STRING** button on the command bar of the cache's general page). Once caching by query string is enabled, page outputs based on different query string parameters are saved separately. For example, `pages/mypage.aspx?parm=value1` and `pages/mypage.aspx?parm=value2` correspond to different cached contents.

Finally, you can define custom domains for your CDN endpoints to provide friendlier addresses to your customers.

11.4 Asynchronous Operations and Parallel Operations

In addition to reducing I/O operations and using cache, asynchronous and parallel operations are also important techniques to improve system performance. Asynchronous operations separate complex tasks from UI to keep UI responsive. Parallel operations increase system throughput by splitting complex tasks into smaller tasks that can be executed in parallel by multiple threads, processes, or servers.

As mentioned at the beginning of this chapter, we will not be able to cover asynchronous operations and parallel operations in detail in this book. The following is a brief summary of different techniques you can apply at different levels.

- .NET multithreading

Under .NET environment, you can use **Thread** type to explicitly manage threads. You can also use **ThreadPool** type to queue work items into a managed thread pool. You can use methods such as **Parallel.For** to convert a liner loop to a parallel loop. You can use Task Parallel Library (TPL) to achieve parallelization. Starting with .NET 4.5, you can also use keywords such as **async** and **await** to implement asynchronous methods. You may refer to related C# and CLR documentations for further details.

- ASP.NET asynchronous controllers

You can use the **AsyncController** type to implement asynchronous controllers under ASP.NET MVC. An asynchronous controller offloads the handling of a web request to a separate thread so that the web server is not blocked by lengthy requests.

A web server maintains a thread pool. When a request arrives, the web server retrieves one thread from the pool to handle the request, and releases the thread back to the pool only when the request has been handled. Because there are a limited number of threads in the thread pool, when the server is busy with many complex requests, the thread pool might be drained. This phenomenon is called Thread Starvation. When Thread Starvation happens, the server will put new requests into a queue. When the queue is filled up, the server will start to reject new queries by returning a Server Busy (503) status code.

Asynchronous controllers use threads differently. When a new request arrives, the web server acquires a thread from the thread pool as usual. But the thread is immediately returned to the thread pool once an asynchronous operation is started. When the asynchronous operation is completed, the web server acquires another thread from the thread pool (which could be different from the original thread) to return the final response.

- Multiple service entities and multiple subscriptions

We have learned that we can scale out our cloud service to share system workload using multiple instances. However, our services often rely on other external services such as storage services and Service Bus. Microsoft Azure imposes quotas and throttling limits on most of these services. When a single service entity, such as a single Table or a single Queue, cannot provide sufficient throughput, you can segment your workload across multiple service entities or even multiple subscriptions to gain additional throughputs. Using multiple subscriptions can also facilitate billing to different customers. You can group resources required by a single customer to a dedicated subscription so the cost can be separately calculated and monitored.

- Service Bus best practices

While introducing Microsoft Azure Table storage, we discussed how to use batched operations on Table storage service to improve performance and to reduce transactional costs. Service Bus also supports batched processing of messages for similar purposes. In addition, you can use techniques such as prefetching and asynchronous send/receive to further improve performance. For different scenarios, you may want to pick and combine several different techniques to get the best performance. For a complete guide of Service Bus best practices for performance, you may consult the MSDN document: <http://msdn.microsoft.com/en-us/library/windowsazure/hh528527.aspx>.

11.5 Summary

In this chapter, we discussed several different aspects to improve system performance: reduce I/O operations, improve I/O speed, and use asynchronous operations and parallelization. The chapter focused on Microsoft Azure Cache service and Microsoft Azure CDN. It also covered some of the asynchronous techniques briefly.

Chapter 12

Claim-Based Architecture

Authentication and authorization are fundamental capabilities needed by most systems, especially by cloud-based systems. The Internet is a hostile environment, with many hackers and ill-purposed introducers wandering around like sharks looking for victims. In terms of figuring out ways to attack, they are very creative and determined. Unfortunately, to create an efficient, reliable, and easy-to-use authentication and authorization system requires deep knowledge on network security and rich experience in antihacking techniques. Obviously, this is beyond the capability of most cloud service developers. Then, what is the solution?

When you need to lock a door, do you make a lock yourself, or do you buy a lock from a store? I believe most people will choose the latter for obvious reasons. First, making a reliable lock requires specialized knowledge, materials, and tools. Moreover, the time investment to make a lock is simply unacceptable to most of us. Second, the quality of the homemade lock is not guaranteed. Is it strong enough? Is it easy to be picked? Is it durable? To answer any of these questions with high assurance is not a simple matter. The lock manufacturers, on the other hand, have perfected the manufacturing processes over years to produce high-quality locks. These locks have passed rigid tests before they are released to the market, and they are battle-tested in the field. So, buying a lock is obviously a smart choice. Similarly, to enable authentication and authorization on a system, instead of spending large amounts of time and money to design and implement a customized system without high-quality assurance, a smart way is to “outsource” authentication and authorization to a trusted party. This is the fundamental idea of claim-based architecture.

Note: About *Absolutely* Secure Systems

During discussions with customers, a question sometimes comes up asking if claim-based architecture is *absolutely* secure—of course not. Just like a lock can be picked, any security system can potentially be compromised. Why? This is because no matter what kind of protection we put around a system, we still need to make the system accessible. This access mechanism causes unavoidable vulnerability. This is just like making a maze. The maze can be very complex, but there is always a path that allows you to navigate through

the maze. As long as this path exists, the maze is solvable. To make the system *absolutely* secure, the system has to be inaccessible. However, an inaccessible system is useless. For instance, you can seal a server in concrete and launch it to the Moon, in which case we may assume the system is *absolutely* secure (if no aliens capture it!), but it becomes completely useless as well.

The architecture that we introduce in this chapter is the result of tens of years of collective experience and effort. The level of security and efficiency it provides is beyond the reach of most individual cloud service developers. Over the years, claim-based authentication and authorization have been proven to be effective and have been widely adopted by many large systems, such as Microsoft Office 365 and Microsoft Azure Management Portal itself.

12.1 Claim-Based Authentication and Authorization

Before we study the architecture, let us review some of the basic concepts:

- User and attributes

A user refers to an entity, such as a human user or a program, which consumes a service. A user commonly has one or more associated attributes, such as user name, email address, and phone numbers.

- Claim

A claim is an assertion made on an attribute of an entity. For example, “This is Mr. Smith” is an assertion made on the last name attribute of a person. Or, more formally, the assertion states that the last name attribute of the person (entity) has the value “Smith.” Any party can make claims. For instance, when I introduce myself to you, I would say, “I’m Haishi”—this is me claiming that my first name attribute is “Haishi.” On the other hand, the name printed on my ID card is a claim made by a third-party (such as the state office) about me.

- Security token

A security token is a collection of claims. A security token is often digitally signed and encrypted to ensure its integrity and authenticity.

- Service provider/relying party

Service provider (SP) refers to the service to be accessed. Under the claim-based architecture, an SP is often called a relying party (RP), because it relies on a third-party to provide authentication.

- Identity provider

An identity provider (IdP) provides the service of authenticating users and issuing security tokens. An SP relies on trusted IdPs to authenticate users. Then, it authorizes or denies user access requests based on the claims, which are contained in the security token issued by a selected IdP. There are several commonly used standard protocols that enable SPs and IdPs to interact with each other, including ws-Federation, SAML, and Open ID Connect. In this book, we focus on ws-Federation.

- Trust

Trust refers to the trust relationship between SPs and IdPs. An SP can trust multiple IdPs; and an IdP can trust multiple SPs. The trust relationship has to be mutual. All trusted

parties together form a circle of trust. Within a circle of trust, SPs can provide users the single sign-on (SSO) experience, which means a user only needs to log in once to gain access to all trusted services within the circle.

- Authentication

Authentication is the process of verifying user identity. The authentication process usually requires the user to provide certain proofs, such as a secret key, a digital certificate, or an issued security token, to prove that he or she is the legit entity whose attributes can be released to a trusted SP. Some IdPs require users to provide multiple proofs. This is the so-called multifactor authentication process. On the contrary, if a service does not require authentication, we say this service allows anonymous access.

- Authorization

Authorization confirms if an authenticated user has access to certain functionalities (or resources). Authorization decisions are often made based on user attributes. For example, some administrative pages of a website may require a user's role attribute to be "Administrator."

Authentication and authorization are two related but different concepts. Authentication answers the question "Who are you?" and authorization answers the question "What are you allowed to do?"

Now with the basic concepts clarified, we can move on to study the basic process of claim-based authentication and authorization.

12.1.1 Basic Authentication and Authorization Process

The claim-based authentication and authorization process involves three parties: the user, the SP (or relying party), and the IdP. The SP outsources authentication to the IdP and authorizes user access based on the claims issued by the IdP. The entire workflow is shown in Figure 12.1.

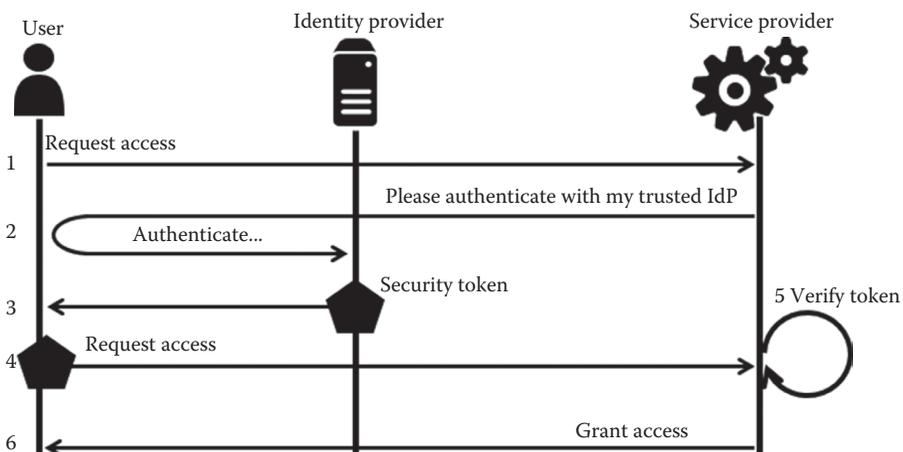


Figure 12.1 Claim-based authentication and authorization.

1. A user requests access to a resource (such as a web page) provided by an SP.
2. The SP detects that the user has not been authenticated. It redirects the user to a trusted IdP for authentication.
3. The user provides the required proofs to the IdP to complete the authentication process. Once the user is authenticated, the IdP packages requested user attributes as claims in a security token, and returns the token to the user.
4. The user requests the resource again and attaches the security token to his or her request.
5. The SP verifies the security token to ensure its authenticity. Then, it retrieves the attribute values from the claims.
6. The SP authorizes or denies access to the resource based on the attribute values.

12.1.2 Authentication and WIF

Over the past decades, people have designed different protocols for authentication processes, such as SAML, ws-Federation, OpenID, Kerberos, and NTLM for Windows. Although these protocols are widely used, to fully understand them in detail requires much time and effort, which are often beyond what most developers would like to invest in. Microsoft Windows Identity Foundation (WIF) provides a simplified programming model that encapsulates the details of these protocols. Developers do not need to understand any details of underlying protocols but can still write complying codes. On the other hand, WIF also allows developers to dig into the protocols and implement various customizations and extensions when necessary. WIF used to be a separate download but with the release of .NET 4.5, claim-based authentication has been embedded deeply into .NET runtime. Many WIF classes have been moved into the *System* namespace, showing the confidence .NET has in claim-based architecture. We use WIF in the examples of this chapter.

12.1.3 Authentication Broker

The authentication process introduced in Section 12.1.1 is only a basic workflow. In more complex scenarios, an authentication broker might be involved (see Figure 12.2). As an intermediate between SPs and IdP, authentication brokers provide the following functionalities:

- Simplify SP development and maintenance. An authentication broker can hide protocol details of interacting with different IdPs. The SP only needs to work with a single authentication broker, who in turn works with different IdPs. As for the SP, it only needs to manage one trust relationship, which is the trust with the authentication broker. The authentication broker then includes more IdPs into the circle of trust by managing trust relationships with each IdP.
- Allow an SP to support multiple IdPs at the same time. An authentication provider provides the necessary mechanisms and user interfaces for end users to pick the IdP they need to use for authentication. The user registration process is always a barrier for a user to sign up for a new service. Allowing users to directly log in using their existing accounts, such as their Microsoft account or Facebook account, is a great way for an SP to build up a bigger customer base with little friction.
- An authentication broker can also modify the claims provided by an IdP, such as filtering or enriching the claims and transforming claim formats. For example, in the case where an IdP cannot provide claims required by an SP, an authentication broker can transform the claims so that the needs of the SP can be satisfied.

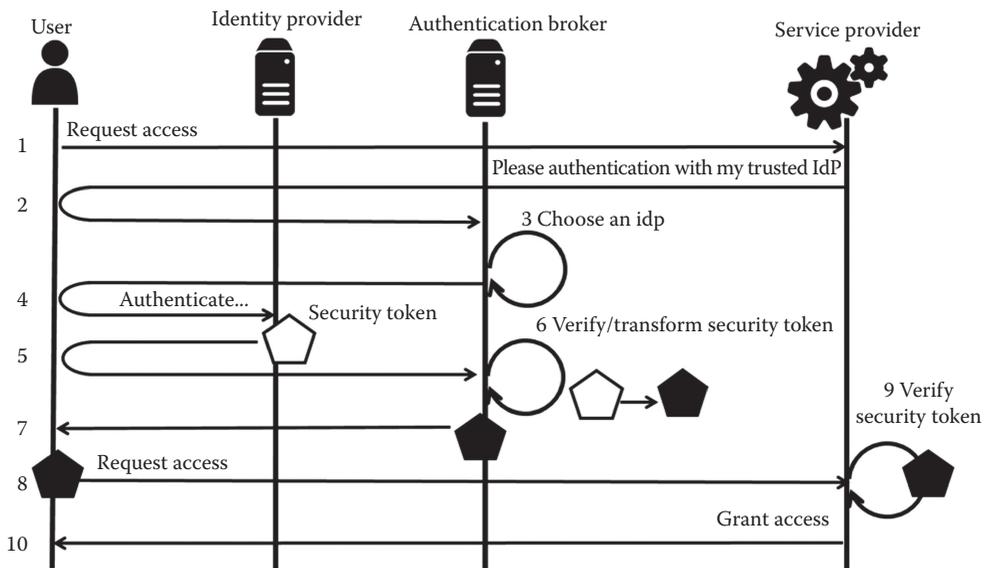


Figure 12.2 Authentication via an Authentication Broker.

Figure 12.2 shows the authentication process when an authentication broker is involved.

1. A user requests access to a resource (such as a web page) provided by an SP.
2. The SP detects that the user has not been authenticated. It redirects the user to a trusted IdP for authentication. In this case, the trusted IdP is the authentication broker.
3. [Optional] If the SP supports multiple IdPs, the authentication provider allows users to pick the IdP they want to use. This process is called home realm discovery. For example, the authentication broker can provide a web page with links to different IdPs for users to choose from. Or, it can automatically discover the intended IdP based on the information carried by the user id (such as email domains when emails are used as user ids).
4. The user is redirected to the selected IdP. He or she provides the required proofs to the IdP to complete the authentication process.
5. Once the user is authenticated, the IdP packages requested user attributes such as claims in a security token and returns the token to the authentication broker. Note that the authentication broker does not perform actual authentication. Instead, it requests a token from the IdP as a replaying party.
6. The authentication broker verifies the token. Based on its configuration, the broker can transform the claims as needed, such as modifying claims, adding new claims, or filtering out some claims.
7. The authentication broker packages the modified claims as a new security token and returns it to the user.
8. The user requests the resource again and attaches the security token to his or her request.
9. The SP verifies the security token to ensure its authenticity. Then, it retrieves the attribute values from the claims.
10. The SP authorizes or denies access to the resource based on the attribute values.

Note: Authentication and authorization in real life

The claim-based authentication and authorization process has great similarity with many real-life scenarios. For example, when you take an airplane trip, the first step is to present your id to the ticket counter staff to get a boarding pass. Expressed in terms of claim-based architecture, your id card is a security token issued by an IdP (your DMV or other state offices) that is trusted by an SP (the airline). The ticket counter staff acts as authentication broker because it verifies one token (your id card) and issues a new token (your boarding card). The boarding card has a different set of claims. It does not contain information such as your age, address, height, and eye color, but contains additional information such as your seat number. Once you board the plane, the boarding pass is your proof of being authorized to sit on a designated seat.

This introduction to the authentication and authorization processes is simplified, but with sufficient details that most developers need to understand to work with claim-based authentication and authorization. After all, the goal of claim-based architecture is to free developers from protocol details and concentrate on business logics, so we will not dig into more details here. Regardless, because there are multiple parties involved, and their configurations have to match up exactly for everything to work together, it is essential to keep a clear picture of relationships among these parties. When you try to modify configurations, you should avoid modifying multiple settings at the same time. Instead, you should try to modify and verify configuration changes step by step.

12.2 Introduction to Microsoft Azure AD

Microsoft Azure AD (WAAD) is a REST-style SaaS provided by Microsoft Azure. It provides identity management and access control services to your services. Many Microsoft online services, such as Microsoft Office 365, Dynamics CRM Online, Windows Intune, and Microsoft Azure, support WAAD as an IdP. By using WAAD, you can not only outsource user management and authentication, but also achieve SSO with the previous services and an increasing number of third-party services (including yours). So, all Microsoft Office 365 users are your potential customers. That is a tremendous advantage in building up a customer base. Moreover, WAAD also allows you to sync users and groups in your local Active Directories (AD) to WAAD so that your existing AD users can directly log in to your cloud services with the existing credentials.

WAAD is a multitenant system. You can provision new tenants for managing your users. The users in your tenants are isolated from the users of other tenants. The user information saved on Microsoft Azure is guarded by world-class security measures, which is far more secure comparing to a custom user database that you manage yourself. Microsoft saving its users into WAAD proves the reliability of the service. So, although the user data are not physically “in your hand,” they are much more secure. This is similar to the difference between saving money in a bank and saving money in a shoebox at home.

WAAD authentication service is free. However, Microsoft Azure Multifactor Authentication service, which we will introduce later in this chapter, charges a small fee for authentication requests.

WAAD also provides a RESTful API called Graph, which allows you to access objects such as Users, Groups, and Roles as well as to discover their relationships.

At the time of writing this book, WAAD adds application access enhancements as well. This feature allows you to easily manage access to SaaS applications, such as Salesforce, Box, and other Microsoft and your own services, with a centralized view. You can easily configure SSO for your users, and manage user access rights to all applications.

Note: WAAD is not a Domain Controller. WAAD only provides user management and authentication service; it does not manage other resources such as computers.

12.2.1 Managing Microsoft Azure Tenants and Users

You can manage Microsoft Azure tenants and users using Microsoft Azure Management Portal. You automatically get a WAAD tenant with your Microsoft Azure subscription. You can also create additional ones via the portal. The following steps will help you create a new tenant:

1. Sign on to Microsoft Azure Management Portal.
2. Click on the **ACTIVE DIRECTORY** link in the left panel to open the Active Directory page. Then, click the **ADD** button on the command bar to create a new directory, as shown in Figure 12.3.
3. On the **Add directory** dialog, enter the domain name of your tenant (domain name has to be globally unique), the region it belongs to (you should select the region where your company is located), and a descriptive name, as shown in Figure 12.4. Click the check button to continue.
4. Once the directory is created, click on its name to navigate to its details page. Then, click on the **USERS** link to switch to the user's page. On the page, you can see your Microsoft Account is listed in the user's list. This is because by default the user is automatically

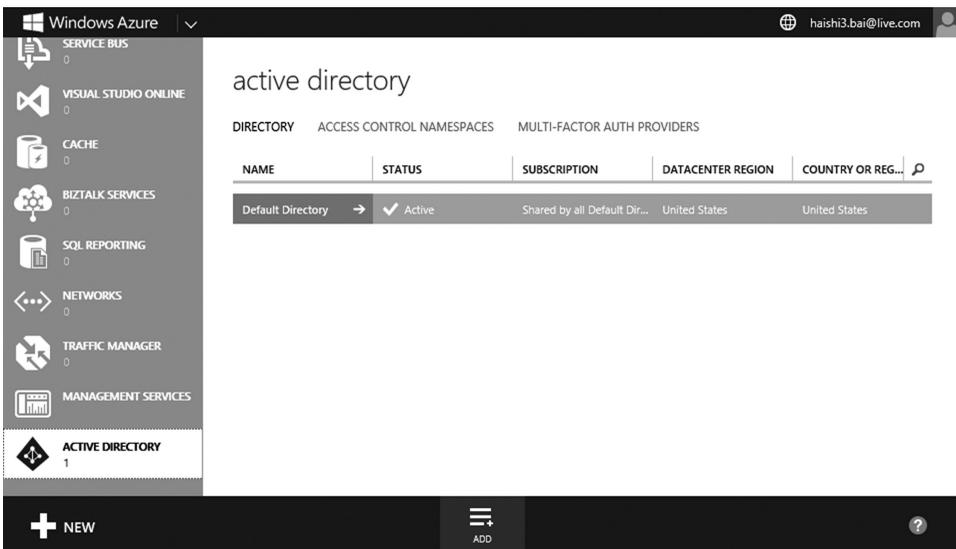


Figure 12.3 Active Directory page.

Figure 12.4 Create a new directory.

configured as a Global Administrator of the directory. However, this user account is not a member of your new tenant—you can observe that the user is a Microsoft account in its **SOURCED FROM** column. You cannot remove this user from the tenant using the management portal (nor can you remove a tenant at this point). To add a new user, click on the **ADD USER** button on the command bar, as shown in Figure 12.5.

5. On the **ADD USER** dialog, choose **New user in your organization** as **TYPE OF USER**, and enter a user name, as shown in Figure 12.6. The new user name will have the format `[user name]@[tenant name].onmicrosoft.com`. You can use your own custom domain as well (more on this later).

DISPLAY NAME	USER NAME	SOURCED FROM
Haishi Bai	haishi3.bai@live.com	Microsoft account

Figure 12.5 User view.

ADD USER

Tell us about this user

TYPE OF USER

New user in your organization

USER NAME

haishi @ zenofcloud.onmicrosoft.com

2 3

Figure 12.6 Add user dialog.

6. On the next screen, enter the user name and display name, and select a role for the user. You can create either a regular User or a Global Administrator. Click the next button to continue, as shown in Figure 12.7.
7. On the **Get temporary password** screen, click on the **create** button to create a temporary password for the new user, as shown in Figure 12.8. When the user logs in for the first time, he or she will have to change the password.
8. Once the temporary password is generated, you can select to send the password in clear text to the corresponding user via email. You can send the password to up to five email addresses, as shown in Figure 12.9.

ADD USER

user profile

FIRST NAME LAST NAME

Haishi Bai

DISPLAY NAME

Haishi Bai

ROLE

User

1 3

Figure 12.7 Configure user name and role.

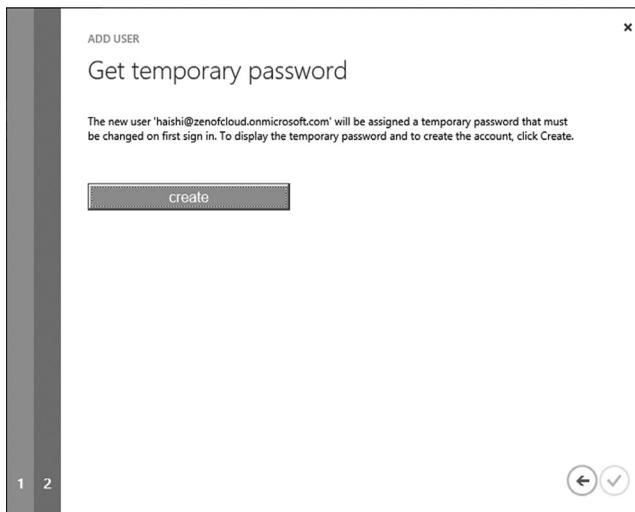


Figure 12.8 Temporary password screen.

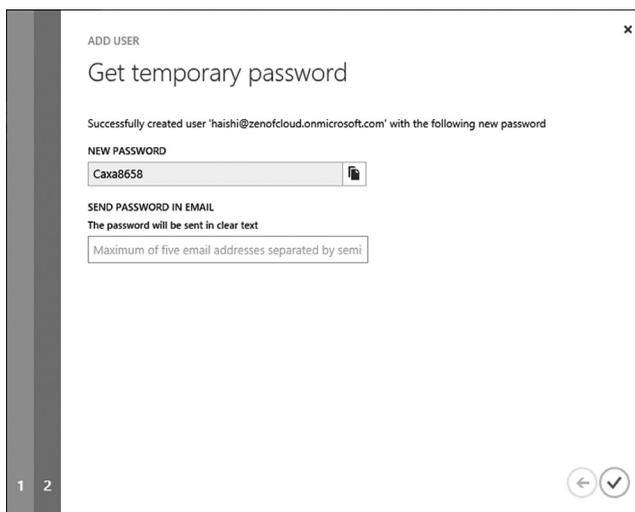


Figure 12.9 Send temporary password screen.

Note: You will not be able to view this password once you leave this screen. So be sure the password is sent or noted down before you close this window. If the temporary password is lost, you can click on the **MANAGE PASSWORD** button on the command bar of the **USERS** page to reset the user's password. This will generate a new temporary password, which the user has to change the next time he or she logs on.

9. Now the new tenant, and the new user, is ready for use.

With WAAD, WIF, Visual Studio tooling, and Microsoft Azure Management Portal, Microsoft provides a comprehensive set of tools and services for implementing claim-based authentication scenarios. Next, we will learn the development processes of a couple of typical scenarios, starting with the most basic one.

Example 12.1: Use Microsoft Azure AD for user authentication

Difficulty: *****

In this example, we create a simple Web PI website, and then configure it as one of the WAAD Relying Parties. Before a user can access this website, he or she has to authenticate with WAAD first. To run this example, you will need Visual Studio 2013 Professional or Ultimate edition. If you do not have either version, you can go to Microsoft Visual Studio's download page

<http://www.microsoft.com/visualstudio/eng/downloads>

to download a free trial.

Before we carry out the exact steps, let us recap the relationships among the participating components. The system diagram is shown in Figure 12.10.

First, we need to establish the trust relationship between the WAAD tenant and the application. As mentioned earlier, the trust relationship has to be mutual: the application needs to trust the WAAD tenant as its IdP, and the WAAD tenant has to enlist the application as one of its trusted Relying Parties. The configurations on both sides have to match up exactly in order for the relationship to be established. Furthermore, we need to configure WIF into our application pipeline so that we can leverage WIF to handle the authentication process for us. WIF provides IIS modules that intercept the HTTP traffic and enables the workflow depicted in Figure 12.1. Obviously, getting everything configured correctly is not necessarily an easy task. Fortunately, Visual Studio 2013 tooling makes this process extremely simple. In the following steps, you will see that we only need to go through a simple wizard to get everything in place.

Now, let us get started.

1. Before we start, we will create a new **Global Administrator** on the new WAAD tenant we have just created. The identity tooling needs *Global Administrator* access to complete the configuration steps for us.

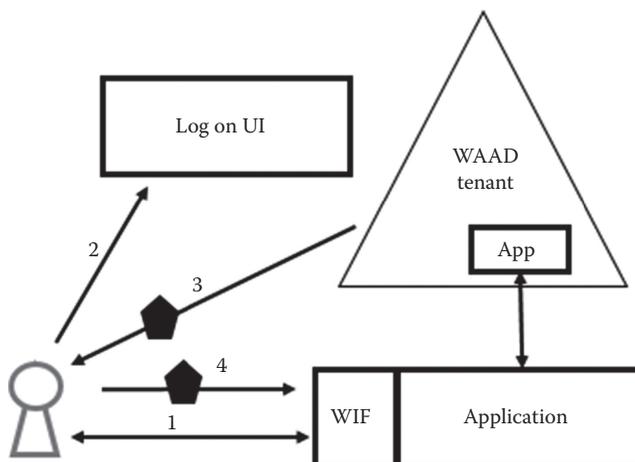


Figure 12.10 System diagram of Example 12.1.

2. Launch Visual Studio 2013.
3. Create a new **ASP.NET Web Application**. On the **New ASP.NET Project** dialog, select the **MVC** template. Before we continue to create the application, click on the **Change Authentication** button to start the process of configuring the trust relationship between our WAAD tenant and the application, as shown in Figure 12.11.
4. On the **Change Authentication** dialog, select the **Organizational Accounts** option, and enter the domain name of your WAAD tenant into the **Domain** field, as shown in Figure 12.12. Click the **OK** button to continue.

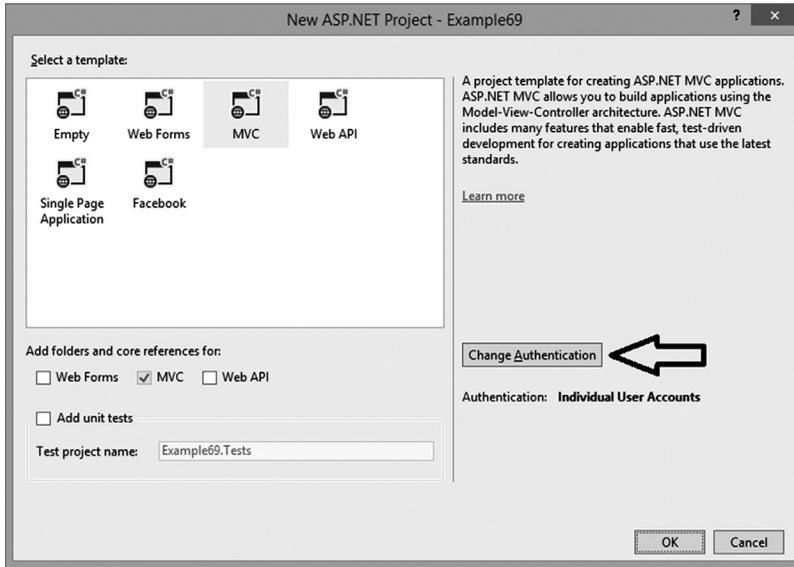


Figure 12.11 Change application authentication method.

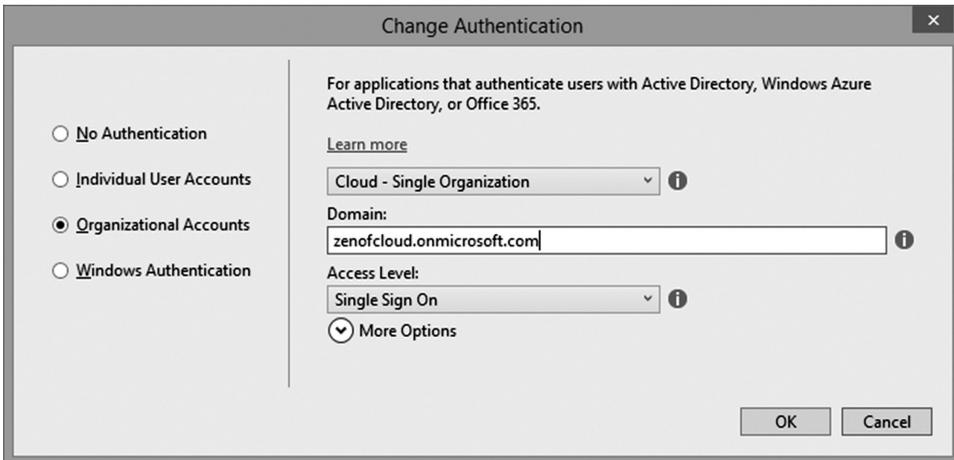


Figure 12.12 Set up organizational account.

5. You will see a log on dialog shows up. Log on to your WAAD tenant as a *Global Administrator*.

Note: If the account you use is a new account, you will see an extra page asking you to set a new password before you can log in.

6. That is all! Once you log in successfully, you are back to the **New ASP.NET Project** dialog, where you can click the **OK** button to finish creating the new ASP.NET application.
7. Press **F5** to launch the application. You will notice that the Visual Studio tooling has enabled SSL on your site and has changed the launch address to use HTTPS. Because we do not have an appropriate certificate in place, you will see the certificate warning as shown in Figure 12.13. Click on **Continue to this website (not recommend)** to continue.
8. You will be redirected to your tenant's sign-in page for authentication. Enter a user name and password from your tenant (not necessarily the Global Administrator account) to sign in, as shown in Figure 12.14.
9. Once the user has been authenticated, the browser is redirected back to your web application, with authenticated user displayed at the upper-right corner of the page, as shown in Figure 12.15.
10. Signing out is autoconfigured as well (compared to earlier versions where you had to do some additional coding). Click on the **Sign out** link to sign out. You will be redirected to a sign out callback page after the user has been successfully signed out (see Figure 12.16). Now, try to click on any of the links at the top. Because you have already signed out, you will need to sign in again before you can access any of the pages.

Now that is almost magical. There are many things that happened behind the scenes. We will guide you through the places where configurations have been changed without digging into too much detail on specifics. This will help you easily locate where to check when you go deeper into the authentication workflow.

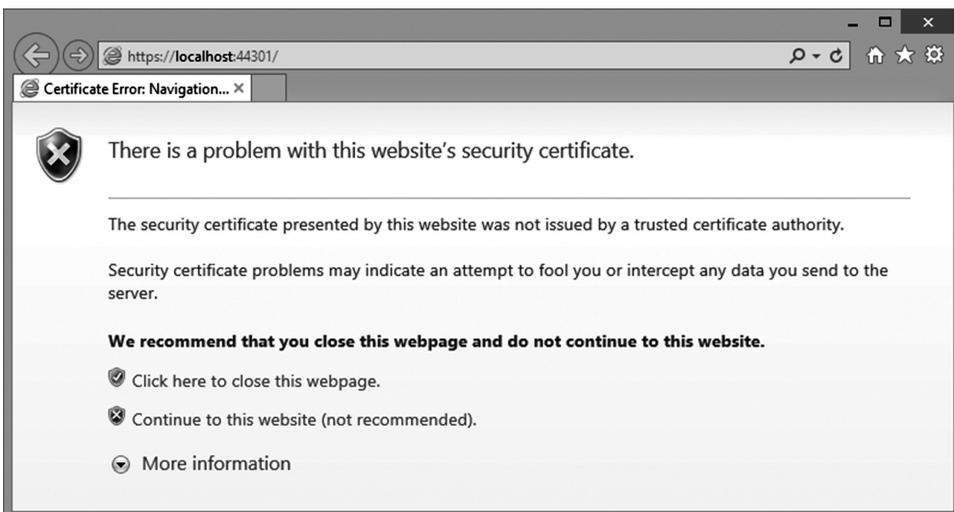


Figure 12.13 Certificate warning.

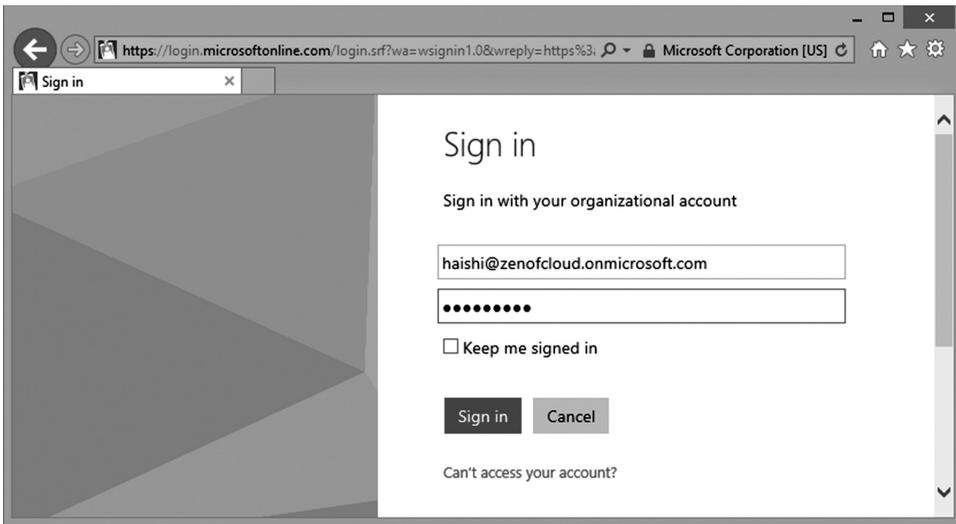


Figure 12.14 Sign in to your WAAD tenant.



Figure 12.15 Web page with a signed-in user.

First, let us take a look at the **Web.config** file. You will see that two WIF modules (FAM and SAM) are injected into the ASP.NET pipeline:

```
<system.webServer>
  <modules>
    <add name="WSFederationAuthenticationModule"
      type="System.IdentityModel.Services.WSFederation
        AuthenticationModule,
        System.IdentityModel.Services, Version=4.0.0.0,
        Culture=neutral,
        PublicKeyToken=b77a5c561934e089" preCondition="managed
        Handler" />
```

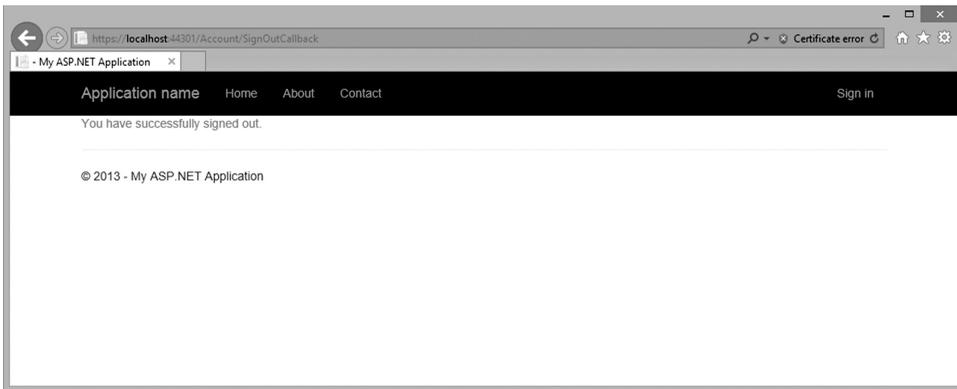


Figure 12.16 Sign out callback page.

```
<add name="SessionAuthenticationModule"
type="System.IdentityModel.Services.
SessionAuthenticationModule,
System.IdentityModel.Services, Version=4.0.0.0,
Culture=neutral,
PublicKeyToken=b77a5c561934e089" precondition="managed
Handler" />
</modules>
</system.webServer>
```

You will also find a new `system.identityModel.services` section added to the configuration file:

```
<system.identityModel.services>
  <federationConfiguration>
    <cookieHandler requireSsl="true" />
    <wsFederation passiveRedirectEnabled="true"
      issuer="https://login.windows.net/zenofcloud.onmicrosoft.com/
      wsfed"
      realm="https://zenofcloud.onmicrosoft.com/Example69"
      requireHttps="true" />
  </federationConfiguration>
</system.identityModel.services>
```

The *passiveRedirectEnabled* attribute specifies that the ws-Federation workflow (as shown in Figure 12.1) should be enabled. Within the *wsFederation* element, the *issuer* attribute specifies the address of the trusted IdP—our WAAD tenant in this case. The *realm* attribute, on the other hand, identifies the application itself. This identifier has to be enlisted as a Relying Party with the IdP before the IdP issues any token for the application.

Another attribute to observe is the *ida:FederationMetadataLocation* in *appSettings*. The value of this attribute points to the IdP's federation metadata address. When the IdP changes its configuration, your application refreshes its own settings to match up with the new changes by reading this metadata file.

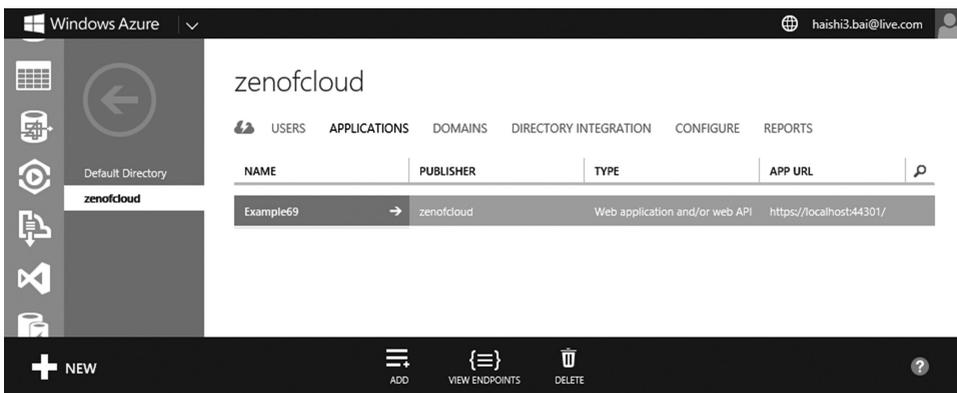


Figure 12.17 Trusted applications by a WAAD tenant.



Figure 12.18 APP URL of an application.

On the WAAD side, your application has been registered as a Relying Party. Log on to Microsoft Azure Management Portal, and you will see the application enlisted in the **APPLICATIONS** view of your WAAD tenant, as shown in Figure 12.17.

Open the application **CONFIGURE** page, where you can observe the application identifier (the APP ID URI field) and a REPLY URL, as shown in Figure 12.18. After authentication is completed, your browser will be redirected to this address with posted back token. Obviously, this will only work on your own local machine, and not on a deployed site. We will see how to fix this next.

Visual Studio tooling allows you to update your authentication settings when you publish your website to Microsoft Azure. When you publish a website, you can update authentication settings on the **Publish Web** wizard. You set this up on the **Settings** tab, where you can enter the WAAD tenant you want to use as the IdP, as shown in Figure 12.19. You should note that to make your published website work, you will need to update the **TenantDbContext** connection to an SQL database instead of the default localDB connection. Then, after you publish your website, you will see another application registered on your WAAD tenant with REPLY URL correctly configured.

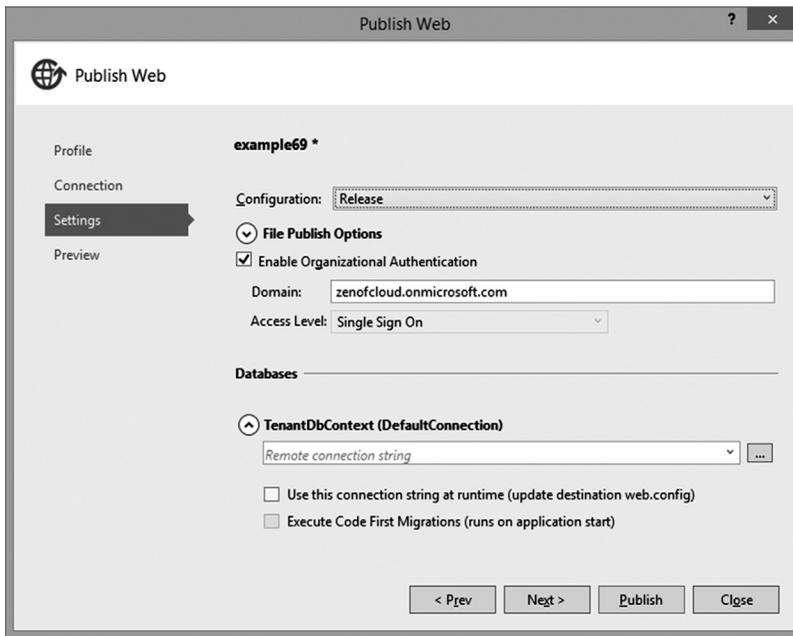


Figure 12.19 Update authentication setting on Publish Web wizard.

Before we move on to the next topic, let us take a quick look at what you need to do in your application code. WIF sets the current thread's user identity for you, so you can directly use the identity in your code. For example, in *_LoginPartial.cshtml*, to display the name of the current user, the code simply writes:

```
<li class="navbar-text">
    Hello, @User.Identity.Name!
</li>
```

You can also cast *User.Identity* as a *ClaimsIdentity* type to access its associated claims. For example:

```
var roles = ((ClaimsIdentity)User.Identity).Claims
    .Where(c => c.ValueType == ClaimTypes.Role)
    .Select(c => c.Value);
```

12.2.2 Graph API

WAAD Graph API is a RESTful API provided by WAAD for querying and updating directories, including reading user lists and group lists; reading, updating, and validating users and user groups; and updating passwords.

Note: WAAD is a directory on cloud, which is different from your on-premise active directories. You cannot query WAAD directories using LDAP.

Example 12.2: Use Graph API

Difficulty: ****

In this example, we will modify the application in Example 12.1 to use Graph API to query the directory. To complete this example, you will need do the following:

- First, download and install WCF Data Service 5.6.0 RTM Tools from the following address: <http://www.microsoft.com/en-us/download/download/confirmation.aspx?id=39373>.
- Then, download Microsoft Azure AD Graph API Helper Library from the following MSDN address: <http://code.msdn.microsoft.com/Windows-Azure-AD-Graph-API-a8c72e18>. This library provides some helper classes for using Microsoft Azure AD Graph API. The downloaded file is a compressed source code package. Expand the code to a local folder of your choice.

Note: This example is adapted from

<http://msdn.microsoft.com/en-us/library/windowsazure/dn151791.aspx>.

1. In Example 12.1, we only granted SSO access to the service principle (see step 4 of Example 12.1). In order to use Graph API, we first need to grant corresponding access rights.
2. On Microsoft Azure Management Portal, open the application configuration page. Then, click the MANAGE ACCESS icon on the command bar. Next, on MANAGE ACCESS dialog, click on the **Change the directory access for this app** link, as shown in Figure 12.20.
3. Check the **SINGLE SIGN-ON, READ AND WRITE DIRECTORY DATA** option. This allows the application to read and write the directory. Click the check button to apply the change (Figure 12.21).
4. On the CONFIGURE page, scroll to the **keys** section. Drop down the **Select duration** dropdown box and select **2 years**. Then, click the **SAVE** button on the command bar to save the change, as shown in Figure 12.22. This will create an access key that is good for 2 years.
5. Once the key is saved, you can view and copy the key from the page (you can also look up the **CLIENT ID** from the same page about the **keys** section), as shown in Figure 12.23.

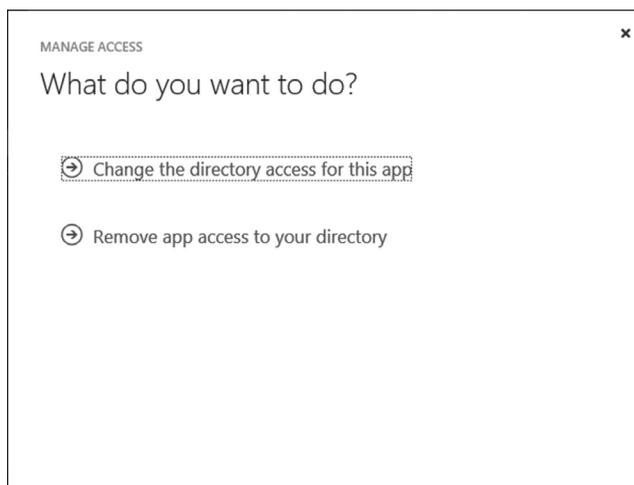


Figure 12.20 Update directory access for an application.

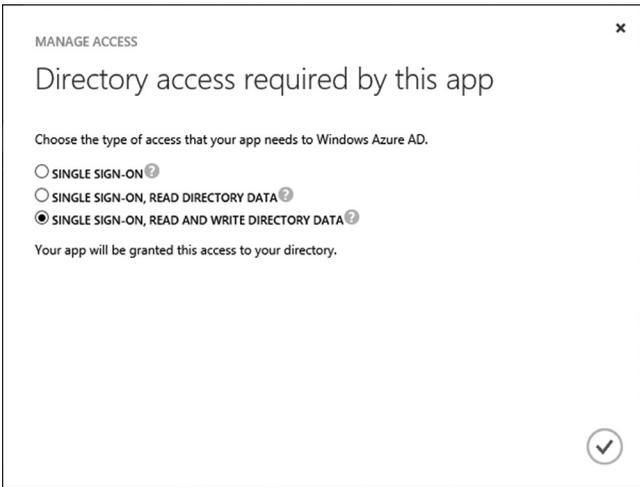


Figure 12.21 Grant directory access to an application.



Figure 12.22 Change access key expiration.

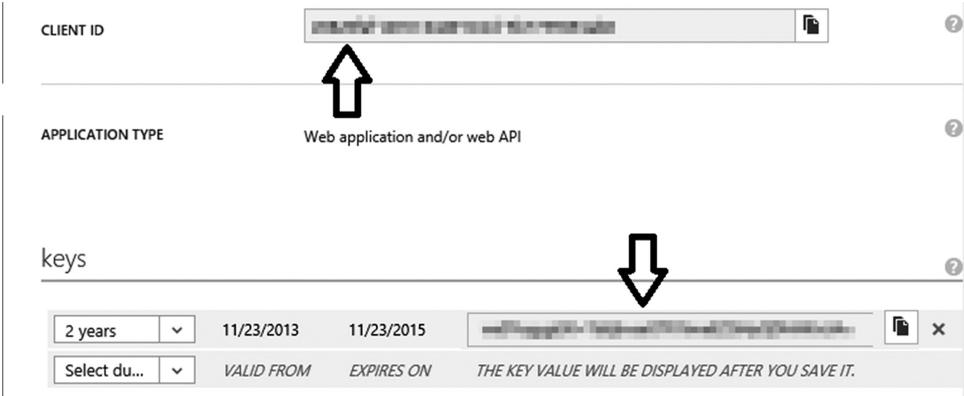


Figure 12.23 Get client id and access key from the configuration page.

6. In Visual Studio Solution Explorer, add the expanded Microsoft Azure AD Graph API Help Library project (*Microsoft.WindowsAzure.ActiveDirectory.GraphHelper*) to the solution.
7. Rebuild the solution.
8. In the web application, add a reference to the newly added project.
9. Then, in the web application, add a reference to *Microsoft.Data.Services.Client* (5.6.0.0 from the *Extensions* group).
10. Modify the **Web.config** file to add new application settings:

```
<appSettings>
  <add key="ClientId" value="[your client id]" />
  <add key="Password" value="[Your access key]" />
  ...

```

11. Because the helper library uses 5.3.0.0, define two assembly mappings to map them to 5.6.0.0:

```
assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
  <dependentAssembly>
    <assemblyIdentity name="Microsoft.Data.Services.Client"
      publicKeyToken="31bf3856ad364e35" />
    <bindingRedirect oldVersion="5.0.0.0-5.6.0.0"
      newVersion="5.6.0.0" />
  </dependentAssembly>
  <dependentAssembly>
    <assemblyIdentity name="Microsoft.Data.Edm"
      publicKeyToken="31bf3856ad364e35" />
    <bindingRedirect oldVersion="5.0.0.0-5.6.0.0"
      newVersion="5.6.0.0" />
  </dependentAssembly>
  ...

```

12. Modify the **Index()** method of the **HomeController** to read the directory to get a user list:

```
public ActionResult Index()
{
    string tenantName = ClaimsPrincipal.Current.FindFirst
        ("http://schemas.microsoft.com/identity/claims/tenantid").
        Value;
    string clientId = ConfigurationManager.AppSettings["ClientId"];
    string password = ConfigurationManager.AppSettings["Password"];
    AADJWTToken token = DirectoryDataServiceAuthorizationHelper
        .GetAuthorizationToken(tenantName, clientId, password);
    DirectoryDataService graphService =
        new DirectoryDataService(tenantName, token);
    QueryOperationResponse<User> response = graphService.users.
        Execute()
        as QueryOperationResponse<User>;
    return View(response.ToList());
}

```

13. Modify the **Views\Home\Index.cshtml** file to display the user list, as shown in the following code list:

```
@model IEnumerable<Microsoft.WindowsAzure.ActiveDirectory.User>

...

<div class="row">
  <table>
    <tr>
      <th>Name</th>
      <th>UPN</th>
    </tr>
    @if (Model != null)
    {
      foreach (var user in Model)
      {
        <tr>
          <td>@user.displayName</td>
          <td>@user.userPrincipalName</td>
        </tr>
      }
    }
  </table>
</div>
```

14. Launch the application again and you will see the user list displayed on the page, as shown in Figure 12.24.

The screenshot shows a web application interface. At the top, there is a navigation bar with links for 'Application name', 'Home', 'About', and 'Contact'. On the right side of the navigation bar, it says 'Hello, haishi@zenofcloud.onmicrosoft.com!' and a 'Sign out' link. Below the navigation bar is a large section with the heading 'ASP.NET' and the text 'ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.' There is a 'Learn more »' button. At the bottom of the page, there is a table with two columns: 'Name' and 'UPN'. The table contains three rows of user information. A white arrow points to the table.

Name	UPN
System Admin	admin@zenofcloud.onmicrosoft.com
Haishi Bai	haishi@zenofcloud.onmicrosoft.com
Haishi Bai	haishi3.bai_live.com#EXT#@zenofcloud.onmicrosoft.com

Figure 12.24 User list displayed on home page.

12.3 Microsoft Azure AD New Features

Just as other Microsoft Azure services, WAAD is under constant development. At the time of writing this book, WAAD is adding some new exciting features, which we briefly summarize here.

12.3.1 Azure Authentication Library

In the previous examples, we used a browser as the client, and the authentication workflow was based on browser address redirections. This type of authentication paradigm is called Passive Client Authentication. On the other hand, Windows desktop applications, including the Windows 8 Store applications, are considered Active Clients. Active Clients cannot directly leverage browser redirection to complete the authentication process. Instead, they can leverage Microsoft Azure Authentication Library (AAL) to do the job. By using AAL, your active clients can rely on WAAD or other IdPs for authentication. AAL is a NuGet library, which you can download and install from the official NuGet site:

```
Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory
```

For an example of how to use AAL, you may consult the following MSDN example page: <http://msdn.microsoft.com/en-us/library/windowsazure/79e09d59-08b9-446a-8ead-209134a4326d>.

12.3.2 Microsoft Azure Active Directory Premium

Microsoft Azure Active Directory Premium is built upon the free WAAD service and provides a set of advanced features that enterprises need. These features include the following:

- **User self-service password reset**
For an enterprise that has tens of thousands of users, self-service is critical for relieving IT staff from small tasks such as resetting passwords that can be carried out by end users themselves.
- **Group-based application access**
Managing application access by groups is an essential requirement of many enterprises. For example, you can grant access to sales applications to the sales group, and access to marketing applications to the marketing group.
- **Company branding**
Sometimes, little touches can go a long way. Company branding allows you to use your own logos and color schemes on your Access Panel, providing the user a more seamless experience navigating among the enterprise applications.
- **Additional security reports**
More detailed security reports will help you gain new insights to improve access security and respond to potential threats.

At the time of writing this book, the Premium feature is a preview service that you need to sign up for separately.

12.4 Summary

This chapter started with a brief introduction of authentication and authorization. Then, it moved on to introduce the basic features of Microsoft Azure Active Directory, including sign on, sign off, and Graph API. Then, the chapter covered the new developments of the service. Of course, merely the fundamentals were presented here. Microsoft's WIF hides the details of underlying protocols, so you can implement authentication scenarios without much specialized knowledge. For interested readers who want to dig deeper, we recommend reading *Programming Windows Identity Foundation* by Vittorio Bertocci.

DEVICES AND CLOUD



The explosive development of mobile devices has had a profound impact on how we work, communicate, entertain, and learn. Mobile devices have not only become an integral part of our personal lives, but also an invaluable assistant for us to access business functionalities from anywhere, at any time. Most successful mobile applications provide access to some sort of services. According to my own survey on application galleries in 2012, about 85% of the top mobile applications are connected to Internet-based or cloud-based services. The result does not include single-player games, which often save a certain amount of information, such as player info and rankings, on backend services as well. So, devices plus cloud has become the main architecture choice for building powerful mobile applications.

Chapter 13

Mobile Service

Microsoft Azure Mobile Service is a turnkey solution for building devices plus cloud applications. It allows mobile developers to quickly build a mobile application with back-end support on popular mobile platforms such as iPhone, Android, Windows Phone, and Windows 8/8.1 devices.

13.1 Mobile Service Overview

Mobile Service provides a set of services that help mobile developers to build back-end services for their mobile applications without needing to understand how to host and maintain high-availability, cloud-based back-end services by themselves. Mobile Service provides the following features:

- **Data storage.** You can easily save data to an SQL database. Mobile Service automates database managements and schema updates so that you do not need any deep SQL knowledge to read and write data.
- **Back-end business logic.** Mobile Service allows you to write JavaScript backend or .Net backend to respond to various data operation events. For example, you can define scripts that respond to data insertions, updates, deletions, and queries to perform custom operations. In addition to data-driven scripts, Mobile Service also supports creating APIs for mobile clients to call directly.
- **User authentication.** At the time of writing this book, Mobile Service provides built-in support for authentication with WAAD.
- **Push notifications.** Mobile Service provides support for sending push notifications to Windows devices, iOS devices, and Android devices.
- **Scheduled tasks.** Mobile Service also allows you to define periodically triggered business logics.

Now, let us learn how to create a simple Windows Store application using Mobile Service.

Example 13.1: Use Mobile Service to build a Windows Store application

Difficulty: *

In this example, we will create a simple Windows Store application to manage to-do items using Mobile Service.

1. Log in to Microsoft Azure Management Portal.
2. On the command bar, select **NEW**→**COMPUTE**→**MOBILE SERVICE**→**CREATE**.
3. On the **NEW MOBILE SERVICE** window, enter a URL for your Mobile Service. In the **DATABASE** field, you can choose to use an existing SQL database or create a new one (you can take advantage of the free 20 MB SQL database offering). Select a **REGION** where you want the service to be provisioned and click the next button to continue, as shown in Figure 13.1.
4. On the **Specify database settings** screen, select to create a new SQL database server. Set up database login credentials, and then click the check button to complete the operation, as shown in Figure 13.2.
5. After the Mobile Service has been created, click on its name to open its details page. On the page, you will find links to generate an application for popular platforms, including Windows Store, Windows Phone 8, iOS, Android, HTML/Javascript, Xamarin and PhoneGap. Here we will choose **Windows Store**, as shown in Figure 13.3.
6. [Optional] You can click on the **Install Visual Studio Express 2013 for Windows** link to install a free Visual Studio Express 2013 tailored for developing Windows Store applications. We will use the Visual Studio Ultimate edition on our machine.
7. Now you can create the database table for your data. Simply click on the **Create TodoItem Table** button (see Figure 13.3) to create a default to-do items table.
8. Once the table is created (see Figure 13.4), you can click on the **Download** button to download an autogenerated application. The default language is C#, but you can select a JavaScript-based version as well.
9. Extract the downloaded zip file to a local folder of your choice.
10. Open the Visual Studio solution file (the .sln file). When you see the security warning as shown in Figure 13.5, click on the **OK** button to continue.

Figure 13.1 Creating a new Mobile Service.

NEW MOBILE SERVICE

Specify database settings

NAME

SERVER

SERVER LOGIN NAME

SERVER LOGIN PASSWORD

CONFIRM PASSWORD

REGION

CONFIGURE ADVANCED DATABASE SETTINGS

Figure 13.2 Specify database settings.

mymobilesample2

DASHBOARD DATA API SCHEDULER PUSH IDENTITY CONFIGURE SCALE LOGS

Your mobile service was created.
 Now let's connect it to an app.

Skip Quick Start the next time I visit

CHOOSE A PLATFORM

Windows Store Windows Phone 8 iOS Android HTML/JavaScript Xamarin PhoneGap

GET STARTED

CREATE A NEW WINDOWS STORE APP

Follow these steps to create a simple Todo app that connects to your mobile service.

- 1 Get the tools

Install Visual Studio Express 2013 for Windows
 (If you are using an older version of Windows or Visual studio, follow this tutorial)
- 2 Create a table

To store data in your mobile service, you need a table. Click the button below to create a TodoItem table for your starter project. To add additional tables later, click the Create button on the Data tab.

Create TodoItem Table

Figure 13.3 Mobile Service details page.

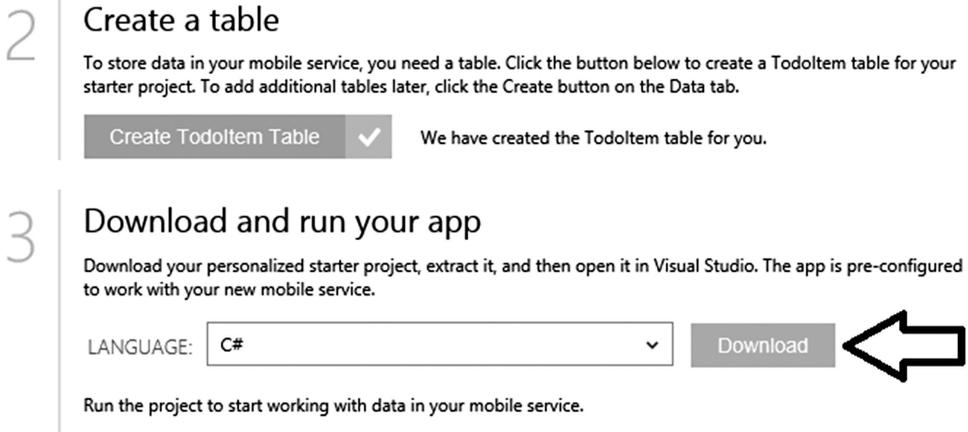


Figure 13.4 Download autogenerated application.

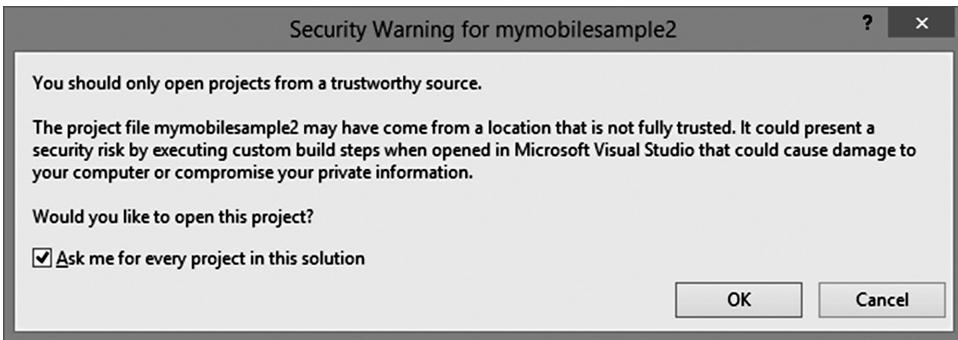


Figure 13.5 Security warning.

11. Visual Studio will prompt you to get a Windows 8.1 developer license. If you agree with the terms, click on the **I Agree** button to continue, as shown in Figure 13.6.
12. You will be prompted to log in using your Microsoft Account.
13. Once you have logged in, you will be notified that you have acquired a developer license. Click on the **Close** button to continue, as shown in Figure 13.7.
14. In Visual Studio, select the **BUILD→Rebuild solution** menu to rebuild the solution.

Note: If you receive a “Package restore is disabled by default” error, it is because the NuGet package restore feature is turned off. To fix the problem, select the **PROJECT→Enable NuGet Package Restore** menu to enable NuGet package restore. Then, rebuild the solution.

15. Press **F5** to launch the application. You can enter new to-do items in the text box to the left, and click on the **Save** button to add them to the list to the right, as shown in Figure 13.8.



Figure 13.6 Developer license prompt.



Figure 13.7 Notification of acquired developer license.

In just a few minutes, you have created a complete mobile application with a front end and a back end. The system is fully functional with all necessary configurations in place. You can use this application as the starting point of your own application.

Next, we will keep building on the application and make some changes in both the front end and the back end.

Example 13.2: Mobile Service back-end programming

Difficulty: *

This example continues from the previous example. We first add simple data validation on the server side and then add a new timestamp field.

WINDOWS AZURE MOBILE SERVICES

mymobilesample2

1 Insert a TodoItem
Enter some text below and click Save to insert a new todo item into your database

2 Query and Update Data
Click refresh below to load the unfinished TodoItems from your database. Use the checkbox to complete and update your TodoItems

Introduce Push Notification

Figure 13.8 Running to-do item application.

Note: This example is based on a sample from the official Microsoft Azure site: <http://www.windowsazure.com/en-us/develop/mobile/tutorials/validate-modify-and-augment-data-dotnet/>.

1. Log in to Microsoft Azure Management Portal. Open the **DATA** page of the Mobile Service you created in Example 13.1, as shown in Figure 13.9. You can observe the number of records in the **TodoItem** table. Click on the table name to open its details.
2. On the details page, you can browse table data, edit data trigger scripts, examine table schema, and modify access rights. Click on the **SCRIPT** link to open the online script editor, where you can write scripts that respond to various operations, such as insert, update, delete, and read, as shown in Figure 13.10.

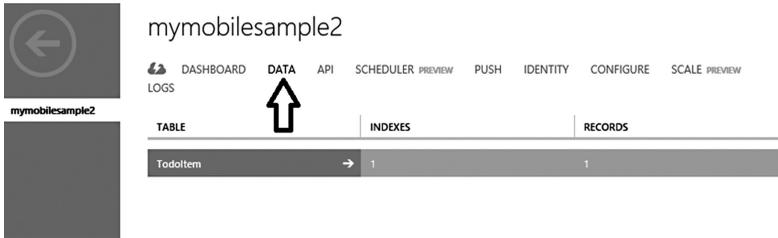


Figure 13.9 Mobile Service data page.



Figure 13.10 Online script editor.

3. Replace the insert script with the following script, which verifies if the text property of the submitted item is less than 30 and rejects the request if the string is too long. After you have entered the new code, click the **SAVE** button on the command bar to save the script. In addition to providing the online editor, Mobile Service also allows you to edit the scripts directly in Visual Studio. You can locate and edit the scripts in Server Explorer.

```
function insert(item, user, request) {
    if (item.text.length >30) {
        request.respond(statusCodes.BAD_REQUEST,
                        'String can\'t be longer than 30
                        characters. ');
    } else {
        request.execute();
    }
}
```

4. Launch the application you created in the last example again. Try to enter a to-do item that is longer than 30 characters, and you will receive a *MobileServiceInvalidOperationException*, as shown in Figure 13.11.
5. Modify the **InsertTodoItem** method to handle this exception. The modified code is shown in Code List 13.1.
6. Run the application again. Enter a to-do item that is longer than 30 characters. Click on the **Save** button to observe how the error message is displayed (see Figure 13.12).
7. Next, we will add a new timestamp field to the system. In the online script editor, modify the **Insert** method again to replace the code with Code List 13.2. Note that line 6 of the code adds a new *createdAt* field to the record. Mobile Service supports dynamic table schema so that you can directly add a new field in the code like this without needing to worry about maintaining table structure. Before you put your service into production, however, you should turn off dynamic schema (in the Mobile Service **CONFIGURE** page, set **ENABLE DYNAMIC SCHEMA** to **OFF**).
8. Launch the application again. Insert a few new records. Then, on the **DATA** page of the Mobile Service, you will observe the new records with the additional *createdAt* field.

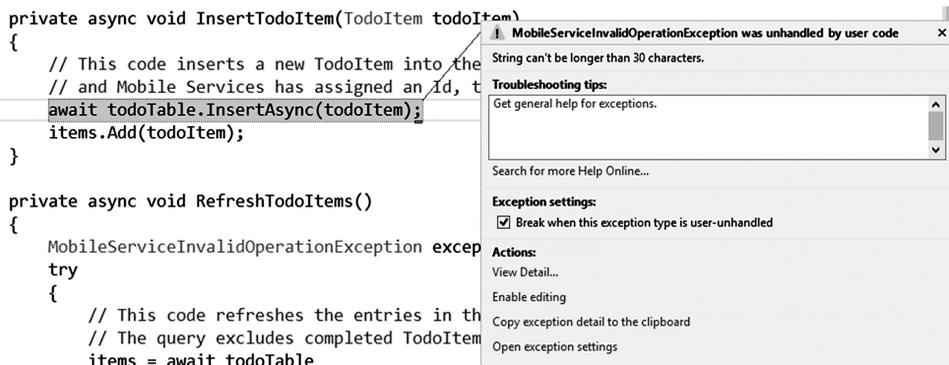
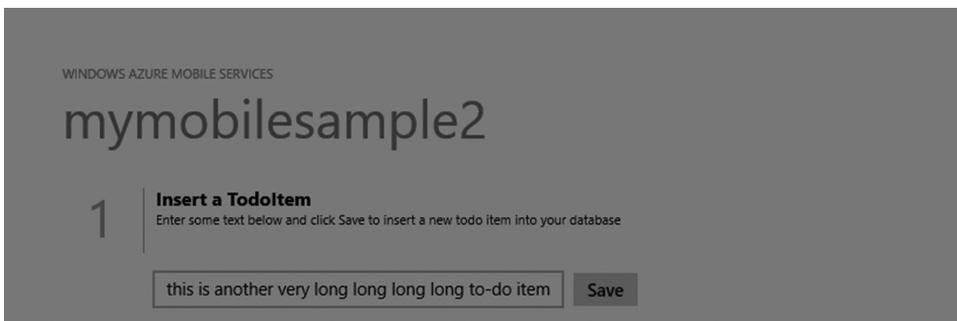


Figure 13.11 MobileServiceInvalidOperationException.

CODE LIST 13.1 CLIENT-SIDE EXCEPTION HANDLING

```
private async void InsertTodoItem(TodoItem todoItem)
{
    // This code inserts a new TodoItem into the database.
    // When the operation completes and Mobile Services has
    // assigned an Id, the item is added to the collection.
    try
    {
        await todoTable.InsertAsync(todoItem);
        items.Add(todoItem);
    }
    catch (MobileServiceInvalidOperationException e)
    {
        MessageDialog errorMsg = new MessageDialog(e.Message,
            string.Format("{0} (HTTP {1})",
                e.Response.ReasonPhrase,
                e.Response.StatusCode));
        var ignoreAsyncResult = errorMsg.ShowAsync();
    }
}
```

**Bad Request (HTTP BadRequest)**

String can't be longer than 30 characters.

Close

Figure 13.12 Input validation error on the client.

CODE LIST 13.2 ADDING A NEW TIMESTAMP FIELD

```

1: function insert(item, user, request) {
2:     if (item.text.length >30) {
3:         request.respond(statusCodes.BAD_REQUEST,
4:             'String can\'t be longer than 30
              characters.');
```

```

5:     } else {
6:         item.createdAt=new Date();
7:         request.execute();
8:     }
9: }
```

9. Now let us modify the client to display the additional field. First, modify the **TodoItem** class in the **MainPage.xaml.cs** file to add the field:

```

public class TodoItem
{
    ...
    [JsonProperty(PropertyName = "createdAt")]
    public DateTime? CreatedAt {get; set;}
}

```

10. Modify the **MainPage.xaml** to display the new **CreatedAt** field. The modified UI code is shown in Code List 13.3.

CODE LIST 13.3 MODIFIED UI

```

1: ...
2: <ListView Name="ListItems" Margin="62,10,0,0" Grid.Row="1">
3: <ListView.ItemTemplate>
4: <DataTemplate>
5: <StackPanel Orientation="Horizontal">
6: <CheckBox Name="CheckBoxComplete"
7:     IsChecked="{Binding Complete, Mode=TwoWay}"
8:     Checked="CheckBoxComplete_Checked" Content="{Binding
      Text}"
9:     Margin="10,5" VerticalAlignment="Center"/>
10: <TextBlock Name="WhenCreated" Text="{Binding CreatedAt}"
11:     VerticalAlignment="Center"/>
12: </StackPanel>
13: </DataTemplate>
14: </ListView.ItemTemplate>
15: </ListView>
16: ...

```

11. Modify the **RefreshTodoItems** method in the **Mainpage.xaml.cs** file to sort the data by the **CreateAt** field in descending order:

```
...
items = await todoTable
    .Where(todoItem => todoItem.Complete == false
        && todoItem.CreatedAt != null)
    .OrderByDescending(todoItem => todoItem.CreatedAt)
    .ToCollectionAsync();
...
```

12. [Optional] Modify the **InsertToDoItem** method and change **items.Add(todoItem)** to **items.Insert(0, todoItem)** so the new to-do items are displayed at the top of the list.
13. Launch the application again to observe the updated display.

13.2 Push Notifications

One of the advantages mobile clients have over traditional desktop clients is to be able to interact with end users anywhere at any time. The close relationship between mobile devices and end users makes mobile devices perfect for improving user engagements. A cloud service is designed to serve a wide user base. An available and scalable service design requires the service to be designed to have least dependencies on operational contexts. On the other hand, personalized service is a common expectation of end users. Mobile devices supplement cloud services by providing service personalization and added stickiness. In other words, mobile devices put cloud services into users' hands so that users can enjoy these services at any time. In addition, mobile devices can also proactively engage with users by sending notifications to them.

13.2.1 Push Notification Overview

Most popular mobile platforms support push notifications. However, each platform has its own Platform Notification System (PNS). For instance, Windows mobile devices use Windows Notification Service (WNS); iOS devices use Apple Push Notification Service (APNS); and most Android devices use Google Cloud Messaging for Android (GCM) or polling. Although these platforms differ from each other, they work in a similar fashion, as shown in Figure 13.13.

1. The mobile device contacts the PNS and gets a handle. The exact name and format of the handle varies from platform to platform. WNS calls it URL or Push Channel. APNS calls it Token, and GCM calls it Registration Id.
2. The handle is submitted to back-end service and saved for later use.
3. When sending a notification, the back-end service provides the handle to the PNS service.
4. The PNS service sends the message to the final target: the combination of a device and an application.

Now let us learn how to push notifications to Windows devices by an example.

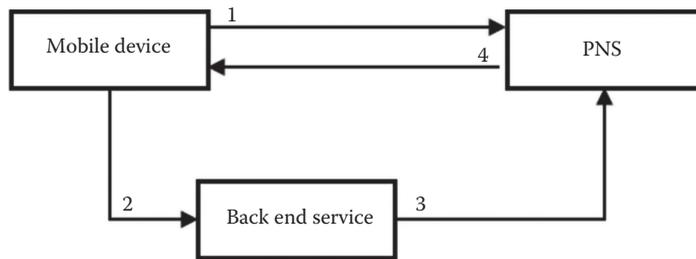


Figure 13.13 General workflow of push notification.

Example 13.3: Push notifications to Windows devices

Difficulty: *

This example continues from Example 13.2. We use WNS service to push messages to Windows devices. We go through the complete process of device registration, handle management, and push notification.

Note: This example is based on a sample from official Microsoft Azure site

<http://www.windowsazure.com/en-us/develop/mobile/tutorials/push-notifications-to-users-dotnet/>

and a tutorial from the same site

<http://www.windowsazure.com/en-us/develop/mobile/tutorials/get-started-with-push-dotnet/>.

1. First, you need to submit your application to Windows Store. Open a browser and log in to Windows Store apps developer center (<https://appdev.microsoft.com/StorePortals/en-US/Home/Index>). If you do not have a subscription, you will need to create a paid subscription first.
2. Click on the **Submit an app** link to start submitting your application, as shown in Figure 13.14.
3. Click on the **App name** link. Then, on the **Submit an app** page, enter a name for your application. Click **Reserve app name** to reserve the application name (Figure 13.15).
4. After the name has been reserved, click the **Save** button.
5. Open the application in Example 13.2 with Visual Studio. In **Solution Explorer**, right-click the project, and select the **Store→Associate App with the Store** menu, as shown in Figure 13.16.
6. On Associate You App with the Windows Store dialog, click on the Sign In button.
7. Log in using Microsoft Account.
8. On the **Select an app name** page, select the application name you just reserved, and click the **Next** button to continue, as shown in Figure 13.17.
9. On the last screen, click on the **Associated** button to complete the operation.
10. Go back to the Windows Store developer center, and click on the **Services** button, as shown in Figure 13.18.
11. On the **Services** page, click on the **Live Services site** link (as indicated by the arrow in Figure 13.19).

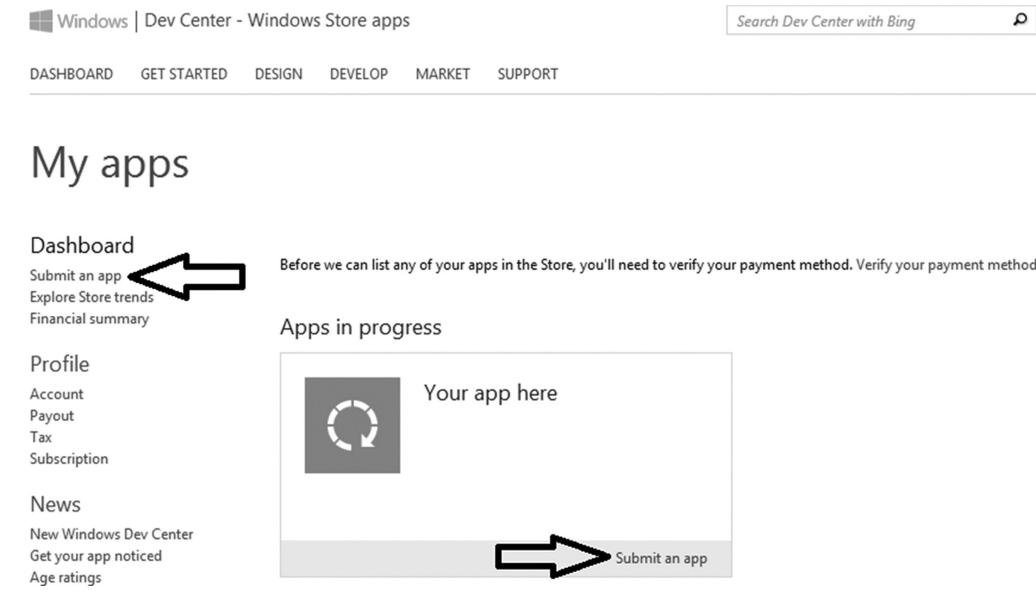


Figure 13.14 Submitting a new application.

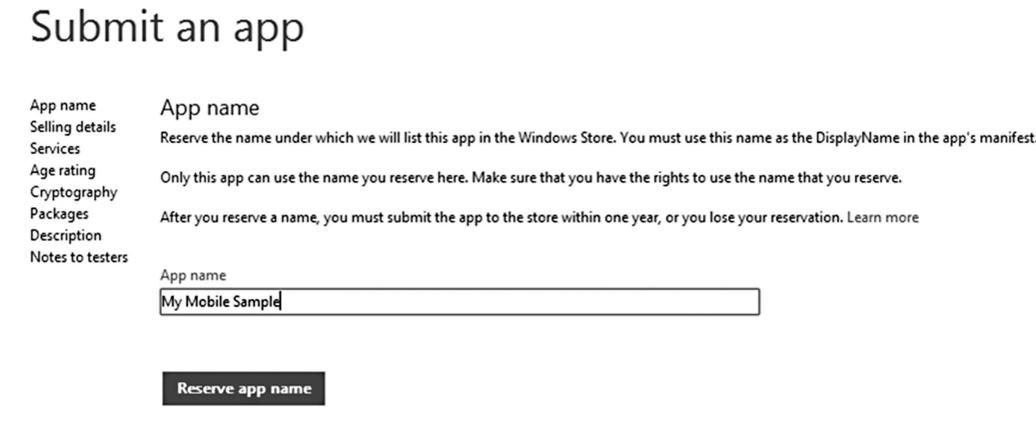


Figure 13.15 Set up application name.

12. On the Push notifications and Live Connect services information page, click on the Authenticating your service link, as shown in Figure 13.20.
13. Now you can copy your Package Security Identifier (SID) and the Client secret from this page, as shown in Figure 13.21.
14. Return to Microsoft Azure Management Portal. Go to the **PUSH** page. Paste in your SID and client secret, and then click on the Save button, as shown in Figure 13.22.

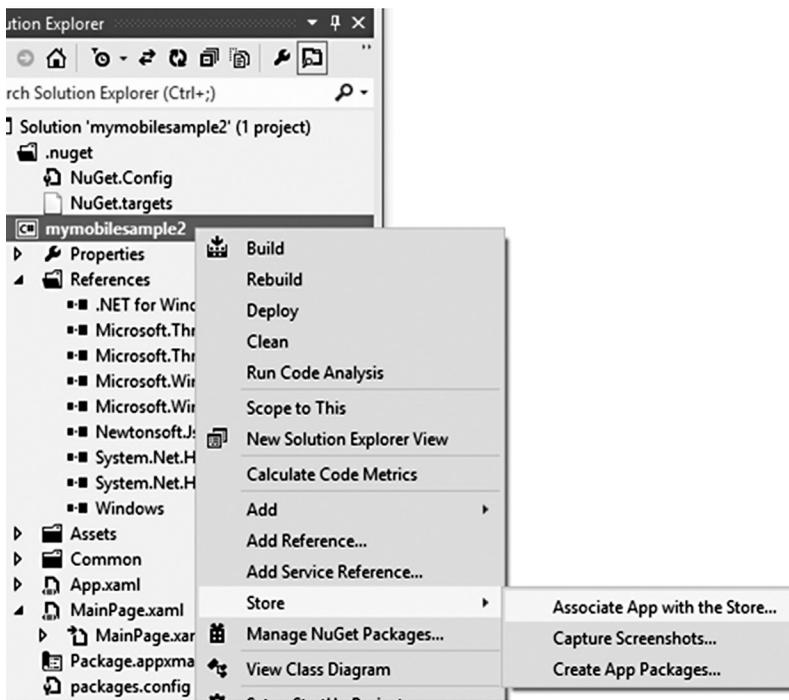


Figure 13.16 Associate app with the Store.

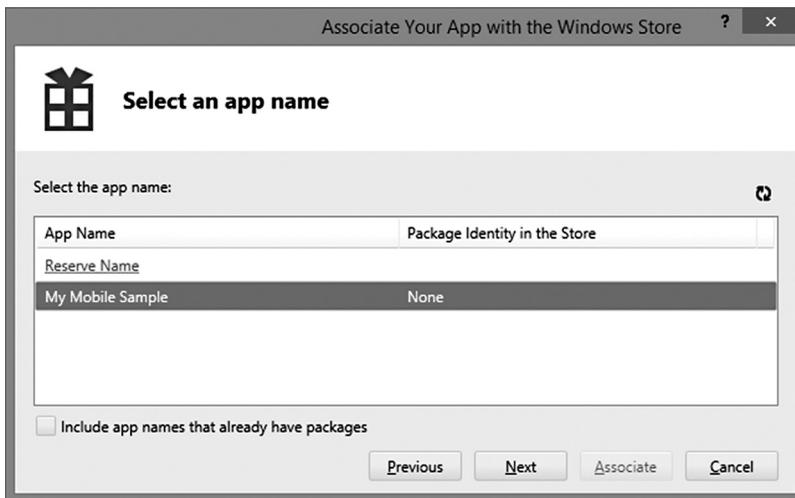


Figure 13.17 Select submitted application.



Services

Add push notifications, authenticate users, enable cloud storage, and define in-app offers.

Learn more

Figure 13.18 Services button.

My Mobile Sample: Release 1

App name	Services
Selling details	Add services to bring connected, integrated experiences to your app and make it more engaging, dynamic, and appealing to yo can also provide in-app offers to let customers make additional purchases from within your app.
Services	
Age rating	
Cryptography	
Packages	Windows Azure Mobile Services
Description	You can use Mobile Services to send push notifications, authenticate and manage app users, and store app data in the cloud. L
Notes to testers	
News	Sign in to your Windows Azure account. Or sign up now to add services to up to ten apps for free.
New Windows Dev Center	If you have an existing WNS solution or need to update your current client secret, visit the Live Services site. 
Get your app noticed	

Figure 13.19 Live Service site link.

Push notifications and Live Connect services info

Overview	Overview
Identifying your app	
Authenticating your service 	You can add Microsoft Cloud Services to your app to give it live app tiles and access to the customer's data. Windows Push (WNS) provides notifications so your app can display dynamic info on the app tile and Live Connect provides access to serv sign-on (SSO), SkyDrive, Hotmail, and Windows Live Messenger.
Representing your app to Live Connect users	
	Before you test or upload your app to the Store, review the following sections that apply to the services your app uses.
	If your app uses WNS for push notifications, review:
	Identifying your app
	Authenticating your service

Figure 13.20 Push notification and live connect service information page.

Push notifications and Live Connect services info

Overview	Authenticating your service
Identifying your app	To protect your app's security, Windows Push Notification Services (WNS) and Live Connect services use client sec communications from your server.
Authenticating your service	
Representing your app to Live Connect users	Package Security Identifier (SID) 
	Client secret 

Figure 13.21 SID and client secret.

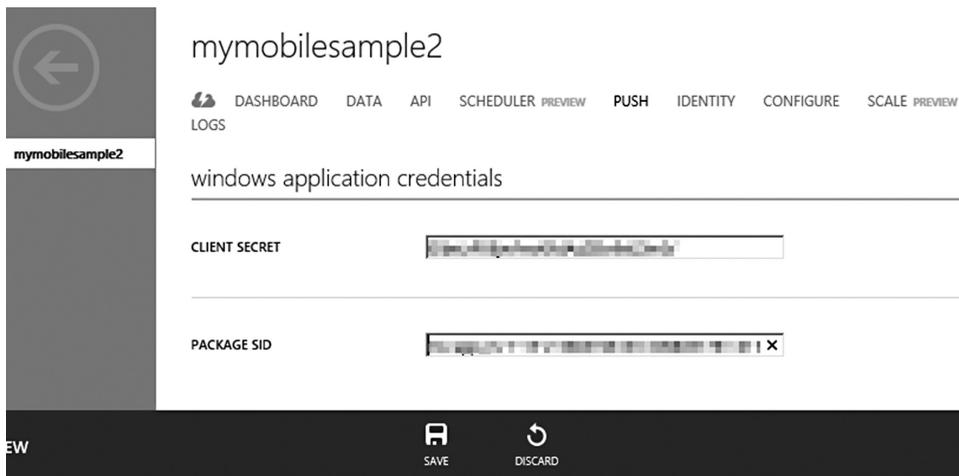


Figure 13.22 Paste client secret and SID into Microsoft Azure Management Portal.

15. In Visual Studio, paste the following code to **App.xaml.cs**:

```
using Windows.Networking.PushNotifications;
...
sealed partial class App: Application
{
    public static PushNotificationChannel CurrentChannel {get;
        private set;}
    private async void AcquirePushChannel()
    {
        CurrentChannel = await PushNotificationChannelManager.
            CreatePushNotificationChannelFor
                ApplicationAsync();
    }
    ...
}
```

16. Add the following line on the top of the **OnLaunched** method:

```
AcquirePushChannel();
```

17. Add a Channel field to the **TodoItem** class in **MainPage.xaml.cs**:

```
[JsonProperty(PropertyName = "channel")]
public string Channel { get; set; }
```

18. Replace the `ButtonSave_Click` method with the following code:

```
private void ButtonSave_Click(object sender, RoutedEventArgs e)
{
    var todoItem = new TodoItem
    {
        Text = TextInput.Text,
        Channel = App.CurrentChannel.Uri
    };
    InsertTodoItem(todoItem);
}
```

19. In the script online editor, replace the Insert script with the following code:

```
function insert(item, user, request) {
    request.execute({
        success: function() {
            request.respond();
            push.wns.sendToastText04(item.channel, {
                text1: item.text
            }, {
                success: function(pushResponse) {
                    console.log("Push Notification: ", pushResponse);
                }
            });
        }
    });
}
```

20. Press **F5** to launch the application. Add a new to-do item, and observe the notification message appearing at the upper-right corner of the screen, as shown in Figure 13.23.

Note: Visual Studio 2013 simplifies these steps.

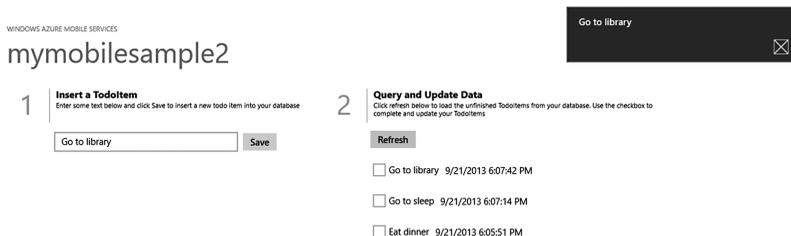


Figure 13.23 Push notification on client.

13.3 Scheduler and API

Scheduler allows you to periodically trigger back-end logics. Let us see how it works with an example.

Example 13.4: Scheduler

Difficulty: *

This example continues from Example 13.3. We add a simple scheduled job to the system.

1. On Microsoft Azure Management Portal, switch to the **SCHEDULER** view of your Mobile Service. Click on the **CREATE** button on the command bar to create a new scheduled job.
2. On the **Create new job** dialog, enter a job name, and then click the check icon to continue, as shown in Figure 13.24.
3. The new job is disabled by default. Click on the name of the job to open its details page. Switch to **SCRIPT** view, where you can enter the Node.js script that gets periodically invoked. Figure 13.25 shows a modified script that logs a message when called. After you have modified the script, click on the **SAVE** button on the command bar to save the changes.
4. Then, click on the **RUN ONCE** button to test the script. Once the script is executed, you can go back to the LOG view of the Mobile Service and observe the logged entry, as shown in Figure 13.26.

In addition to scheduled jobs, Mobile Service also allows you to define REST-style APIs for client application to call directly. It is fairly easy to define and use API, which we will see in the following example.

MOBILE SERVICES: SCHEDULER ✕

Create new job

JOB NAME

SCHEDULE

Every ▼

On demand

You can add a schedule later.

Figure 13.24 Create a new scheduled job.

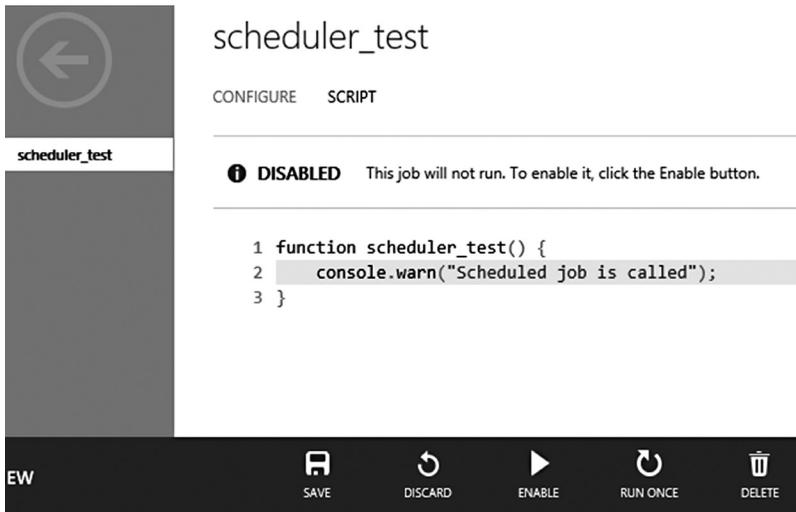


Figure 13.25 Modified script.

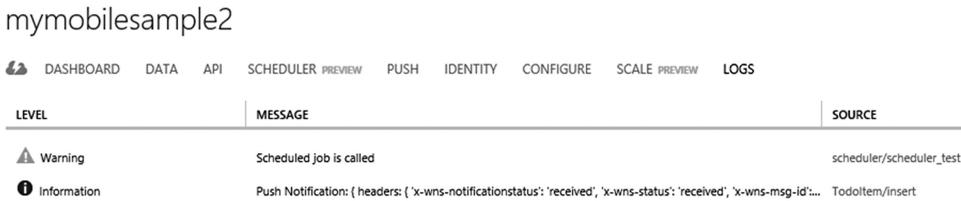


Figure 13.26 The log entry generated by the scheduled job.

Example 13.5: API

Difficulty: *

This example extends Example 13.4 and adds a simple “hello world” API to the system.

1. On the Mobile Service’s API page, click on the **CREATE** icon on the command bar.
2. On the **Create a new custom API** dialog, enter *api_test* as the API name. Your API will have the address *http://[Mobile Service name].azure-mobile.net/api/[API name]*. For simplification, set **GET PERMISSION** to **Everyone** to allow anonymous HTTP GET requests; otherwise, you will need to attach a valid authentication token with your request. Click the check icon to complete the operation, as shown in Figure 13.27.
3. After the API is created, click on the API name to open the script editor. Mobile Service provides sample implementations for both POST method and GET method:

```
exports.post = function(request, response) {
  // Use "request.service" to access features of your mobile
  // service, e.g.:
  //   var tables = request.service.tables;
```

```
// var push = request.service.push;
response.send(statusCodes.OK, { message : 'Hello World!' });
};

exports.get = function(request, response) {
  response.send(statusCodes.OK, { message : 'Hello World!' });
};
```

4. Modify the GET method implementation to return some HTML contents instead of the JSON object:

```
exports.get = function(request, response) {
  response.send(statusCodes.OK, "<h1>Hello World!</h1>");
};
```

5. Open a browser and navigate to *http://[Mobile Service name].azure-mobile.net/api/[API name]*, and you will see the returned <h1> tag, as shown in Figure 13.28.

MOBILE SERVICES: API

Create a new custom API

API NAME

You can set a permission level against each HTTP method for your custom API.

GET PERMISSION

POST PERMISSION

PUT PERMISSION

PATCH PERMISSION

DELETE PERMISSION

✓

Figure 13.27 Create a custom API.

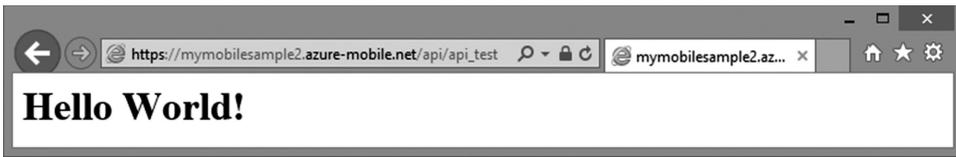


Figure 13.28 Calling Mobile Service API.

13.4 Summary

Microsoft Azure Mobile Service provides a turnkey solution for mobile application developers to quickly create supporting back-end services to provide data storage, push notification, authentication, API, and scheduled jobs. In this chapter, we learned the basic steps of provisioning a new Mobile Service, saving data to the back-end server, sending push notifications, and defining scheduled jobs and APIs.

Chapter 14

Internet of Things

When we talk about “devices,” we should not constrain ourselves to smart phones and tablets. With the development of the Internet and telecommunication technologies, the number and types of devices that are capable of connecting and communicating with each other are constantly growing. These devices are playing increasingly important roles in various areas such as telemetry, automatic control, logistics, robotics, and artificial intelligence. In this chapter, we first provide a brief overview of Internet of Things (IoT). Then, we analyze the relationship between IoT and cloud. Finally, we study IoT developer experiences provided by Microsoft.

14.1 IoT Overview

The term Internet of Things was proposed by Kevin Ashton in 1999. Till today, IoT is a relatively new concept and people have not agreed on a precise definition of it. As the name suggests, IoT refers to a large number of “things” that are connected by a network. A “thing” in IoT can be many things, such as a smart phone, a tablet, or a sensor. However, some common characteristics of “things” can be summarized:

- Uniquely identifiable communication nodes. Although the “things” in IoT differ from each other in a thousand ways, they can all be uniquely identified and most of them can participate in communications independently. Here “communication” is not limited to data exchanges over the Internet, but also in other ways such as telephone networks, satellites, inductions, and even chemical reactions.
- Automatic data exchanges. Data exchanges in IoT often happen without human interventions. Automatic data acquisition, analysis, exchange, and publication eliminate error-prone human factors and ensure efficient, precise system operations.
- Real-world contexts. The “things” on IoT are closely connected to real-world environments and objects. The data collected by the “things” are often relevant only to a specific time, location, and object. For example, a speedometer attached to a car only reflects the temporal speed of the specific car. The reading is most meaningful only when it is read in real time.

In the following sections, we introduce several typical IoT application scenarios, through which we can gain more insights into different characteristics of the IoT.

14.1.1 Radio Frequency Identification

Radio Frequency Identification (RFID) is a noncontact wireless data exchange technology. RFID tags, especially passive RFID tags, have been widely used in our lives, such as ID cards, public transportation cards, and hotel key cards. However, the most important application of RFID is in the field of logistics. For example, the ISO/PAS 18186 RFID Cargo Tag, invented by Bao Qifan from China, has been widely used in major waterways around the globe to seamlessly track the movement of containers. Another example is Walmart, which has been advocating and applying RFID technology throughout its systems. It uses RFID to optimize inventory levels, improve turnover of stock, and increase processing accuracy and efficiency.

RFID is also used in positioning within a controlled environment. Active RFID tags can be used to provide position tracking with higher precision in a local environment, especially in an indoor setting, which is often a blind area of satellites. Furthermore, the Near Field Communication (NFC) technology, which is developed on RFID, is gaining wider adoption as well.

Note: Windows Phone 8 and some of the Windows 8–based laptops and tablets have built-in NFC capabilities. Interested readers may consult Windows Store applications developer center: <http://msdn.microsoft.com/en-us/library/windows/apps/hh465205.aspx>.

RFID has all the IoT characteristics summarized earlier: RFID tags uniquely identify objects; readers exchange data with RFID tags without user inputs; and RFID tags are associated with real-world objects. RFID also shows that a “thing” in IoT does not necessarily need much (or any, in case of passive tags) possessing power like that of smart phones or tablets to participate in data exchanges.

14.1.2 Artificial Intelligence Equipment

The equipment on IoT is not limited to data acquisition; it may also process the data intelligently and directly interact with the physical environment. For example, many car manufacturers and high-tech companies are developing self-driving cars. These cars can plan their maneuvers based on real-time image analysis, distance sensors, and digital maps. Multiple self-driving cars can also exchange data with each other to coordinate smoother traffic. It is easy to anticipate that the wide adoption of self-driving cars will have a profound impact on road transportation by reducing accidents, increasing speed limits, optimizing energy consumption, and avoiding jams.

We can consider self-driving cars a part of IoT because of the ambiguous definition of “things” in IoT: as long as a component can identify itself and participate in data exchange, it can be considered a part of IoT. Equipment on IoT can be as simple as an RFID tag or as complex as a car, which is a combination of mechanical and electrical parts. The inclusiveness of IoT allows a very rich set of scenarios. We will take a peek into the examples later in this chapter.

14.1.3 Wearable Devices

Wearable device is not a new concept. It can be traced back to the 1990s, when Steven Mann first introduced the concept. Wearable device is one form of Ubiquitous Computing, which

allows humans to interact with computing services in natural, intuitive ways without explicit engagement with a computer. For example, a pair of smart glasses that has face recognition capability can use Augmented Reality (AR) to superimpose recognized names onto the user's view. It can also directly present surrounding information such as nearby restaurants, shops, and activities to the user without the user explicitly operating a computer—all the user needs to do is to look around.

Many people believe wearable devices will lead the next wave of consumer electronics. We shared the same view, but believe the next wave will include not just wearable devices but also other technologies such as immersive gaming and spatial display.

14.1.4 Wireless Sensor Network

Wireless Sensor Network (WSN) is a network of connected sensors that are capable of collecting environmental data from a given region. The main application areas of WSN include air-quality monitoring, forest fire detection, landslide detection, and water-quality monitoring. Such data are collected to centralized locations for storage and analysis either by direct connections to the server or through multiple hops among sensors to finally reach the server.

14.2 Devices and Cloud

We are living in the era when Ubiquitous Computing is quickly becoming a part of our everyday lives. With the development of cloud technology, Ubiquitous Computing also means Ubiquitous Cloud, which means cloud-based services are becoming natural extensions of our lives to provide enhancement to our day-to-day activities. The combination of devices and cloud provides unprecedented opportunities and unlimited energy for the development of Ubiquitous Computing. They complement each other so well that together they empower many innovative scenarios to make our lives better.

14.2.1 Importance of Devices for Cloud

One of the main characteristics of cloud service is its high availability to a broad audience. In other words, as far as a cloud service is concerned, a conversation with a customer is merely to respond to a request. However, to an end user, the perception of a conversation with the services has a much longer span and much richer contexts. For example, a user who is arranging weekend activities may use a search engine to search for restaurants, an online ticketing site for movie tickets, a navigation service to find directions, and a payment service to pay for the expenses. This real-life activity happens at different locations, over a couple of days, and involves a number of cloud services. None of the involved services actually understands or captures this context. However, a carefully designed mobile application can create the connections among them and provide a complete end-to-end user experience. In other words, mobile devices make cloud service more relevant to our lives by providing the connections to our real-life contexts and anytime access, anywhere, to the services while the situation is still relevant. Another example of such relevancy is a user searching for a restaurant. A search from a mobile phone can be automatically augmented by location and time so that the user can find a nearby restaurant that is open at the moment: obviously, if you were in Seattle, finding even the best restaurant in Sydney provides little value to solve your dinner problem.

Another importance of devices for cloud is their added stickiness. Many cloud-based services have a very different business model compared to the traditional license-based software. With the traditional license + maintenance contract sales model, the cycle of selling and implementing a software system is quite lengthy, requiring considerable upfront investments and serious commitments. The difficulties of implementing a new system, however, make giving up a system a painful process as well. On the contrary, many cloud services use a subscription model, where users can provision and use services via self-service. The increased agility and lowered risk of subscribing to a new service is definitely beneficial for a new service to grab market shares. The steady, predictable cash flow under the subscription model is favorable to most service providers and investors. However, just because the risk of adopting a new service is lowered, giving up a service is not as painful, either. So, it is not surprising to see cloud service customers being less loyal to traditional software users. A service client installed on a mobile device encourages the user to use the service more frequently. The sense of ownership improves the stickiness of the service. Many media services and social network services are making all the effort to offer free clients on various mobile platforms, because they understand the value that mobile clients can have in terms of creating a tighter relationship with customers and improving user loyalty.

The rich data acquisition and feedback capabilities of devices provide opportunities for cloud service providers to offer value-added functionalities and richer end user experiences on the clients. As mentioned in earlier chapters, cloud services to end users are endpoints that provide some sort of service. Their relevancy to real life is decided by the rich contexts provided by the devices. Their reach is broadened by the devices. The end user experience they provide is enhanced by the expressiveness of the devices.

14.2.2 Importance of Cloud for Devices

As mentioned at the beginning of Section III, most top mobile applications have a web-based or cloud-based back end. The percentage of this architectural choice is simply too high to be coincidental. Rather, this is a solid proof of how cloud and devices complement each other. Cloud services can aggregate and analyze data collected by multiple devices, provide communication channels among devices, and coordinate distributed workflows across different devices. For example, a temperature sensor captures only a point temperature at a given time. However, when a cloud service collects temperature readings from many strategically located temperature sensors, it will be able to plot a heat map of a region. By storing and comparing historical data, it can calculate trends in temperature changes. Moreover, by comparison with other data, it can detect correlations between temperature changes and other factors such as human activities. All these functionalities are impossible to be achieved by a single device.

Although the computational power and storage capacity on mobile devices are constantly increasing, mobile device capabilities are still constrained by the weight, dimension, and battery life requirements. Cloud services provide unlimited extensions to these devices. For instance, backing up files to cloud storage has become common practice in many mobile phones. With future developments in networking technologies, an increasing number of tasks that were traditionally handled by client devices, such as game AI and screen rendering, can be performed on cloud. It is easy to anticipate that with the power of cloud, the virtual world created by computers will have unprecedented detail and fidelity. When human beings no longer perceive the difference, the world will enter a new era where the virtual world and reality will fuse. We shall not digress further here, but that is our true belief.

Most of us take pictures using our cell phones. Mobile phone manufacturers have been competing to provide a better photo experience. However, there is an interesting dilemma here: on the one hand, the resolution of cameras keeps increasing to capture images and videos with higher qualities; on the other hand, the larger data files keep adding pressure to the storage. This is a catch-22 situation that the device itself finds hard to resolve. Many users have tried to take on the matter themselves by backing up pictures to PCs and external devices. However, the complicity of managing these backups is often unacceptable. Cloud provides an elegant solution to this problem. User pictures can be safely preserved on cloud, and additional storage capacity can be acquired at any time without affecting existing archives. In addition, cloud can provide additional value-added services such as sharing, printing, and image processing. The trend is clear that eventually data management will become totally transparent to end users. End users will no longer need to worry where and how much data are to be saved, and they will be able to access their data from any device just as if the data have been magically replicated everywhere for their convenience.

Note: We have to admit that here we are mixing the concepts of web services and cloud services. Obviously, not all web services are hosted on cloud. However, as the Internet can be viewed as a huge pool of resources that provides services via endpoints, it shares many characteristics of cloud platforms. For devices, it really does not matter how these services are hosted as long as they are accessible.

14.3 Challenges of IoT

The development of IoT brings new challenges to system architecture, design, development, and operation. Effectively managing a large number of devices and a large amount of data creates challenges in system availability, scalability, maintainability, performance, and throughput. For example, addressing a large number of devices alone is a real problem. People have been seeking solutions since the 1980s. The widely used IPv4 network can provide 2^{32} (4,294,967,296) unique addresses in theory. However, because many address spaces are reserved for special purposes, with the number of devices exploding, IPv4 addresses can be exhausted. Although technologies such as NAT and CIDR can provide some relief, IPv6, which provides 128-bit address spaces, is being widely adopted.

Managing a large number of devices is a challenge by itself—how would you effectively manage tens of thousands of devices? Obviously, manual management is not an option. We will have to find ways to automate the process. For example, devices can automatically register and configure themselves once they are connected. They keep themselves updated by periodically checking for updates. They should be able to be monitored remotely and should send alerts when their self-diagnosis fails. Moreover, they should employ a modular design so that they can be easily maintained, fixed, or replaced. On the server side, capabilities such as unified management view, centralized policy management, and security key distribution are all essential features for an efficient management system.

Note: One example of such a modular design is Field Replaceable Units (FRU). They are parts that can be easily pulled out and replaced with new ones to minimize the cost and complexity of hardware maintenance.

Data acquisition is another challenge. As a matter of fact, the data acquisition problem is one of the major problems to be solved for IoT. It requires not only enhancement in networking technologies and adoption of industrial standards, but also carefully designed architectures that fully take network bandwidth, storage, and computation needs into consideration. All cloud services hosted on Microsoft Azure are subject to capacity limitations. For instance, different compute node sizes have different bandwidth allocation associated with them. Each storage account has a quota on maximum transactions that can occur per second (the limitation is 20,000 transactions per second at the time of writing this book). Such quotas should be sufficient for most applications. However, they become overconstrictive when you need to work with a huge number of devices. To work around these limitations, you may need to horizontally segment your workload to multiple service entities such as multiple storage accounts or even multiple subscriptions. In addition, storage of excessive amount of data incurs excessive cost. So you will need to make cautious choices of where and how data are stored. For example, you can store log files that do not need complex queries into table storage service, and extracted key features into an SQL database for complex queries and BI. Temporal data can be saved in cache clusters for faster access and lowered storage cost.

14.4 .NET Micro Framework

As this book is designed primarily for developers using Microsoft technologies, we will present a brief introduction of .NET Micro Framework in this section. For many developers who develop PC-based systems using high-level languages such as C# and Java, coding against an embedded system is a very foreign concept. Microsoft's Micro Framework allows .Net developers, especially C# developers, to write a code for embedded systems using the language and tools that they are familiar with.

14.4.1 .NET Micro Framework Overview

.NET Micro Framework is an open-source environment for developing software running on embedded systems. .NET Micro Framework allows .NET developers to write a code for embedded systems using languages they know (such as C#) in the environment they are familiar with (such as Visual Studio) so that they can quickly ramp up on embedded system developments without needing to learn low-level languages and acquiring device-specific knowledge.

Note: You can download Micro Framework from CodePlex.

If you use Visual Studio 2010, you should use .NET Micro Framework SDK 4.2 (download the address <http://netmf.codeplex.com/releases/view/91594>). If you are using Visual Studio 2012 or higher, you should use the 4.3 version (download the address <http://netmf.codeplex.com/releases/view/81000>).

Seeing is believing. Now we will go through the development process using .NET Micro Framework with an example. You do not need any special hardware to run through the following example. All you need is Visual Studio and .NET Micro Framework.

Example 14.1: Hello, Embedded System!

Difficulty: *

This is a simple example of using Micro Framework. Here we assume you have downloaded and installed .NET Micro Framework 4.3 SDK and Visual Studio 2012. The development process is the same with Visual Studio 2010.

Note: This example is adapted from <http://msdn.microsoft.com/en-us/library/ee435413.aspx> and is updated for Visual Studio 2012.

1. In Visual Studio 2012, select the **FILE**→**New**→**Project** menu.
2. In the **New Project** dialog, select the **Visual C#**→**Micro Framework** template in the left panel, and select **Window Application** in the middle panel. Then, click on the **OK** button to create the project, as show in Figure 14.1.
3. Press **F5** to launch the program. This brings up an emulator, as shown in Figure 14.2. The emulator has five buttons and a screen, where it displays “Hello World!” Close the emulator window to stop the program.
4. In **Solution Explorer**, double click the **Resources.resx** file to open the project resource editor. Select the **Add Resource**→**New Image**→**BMP Image** menu. Enter “helloImage” as the new resource name, and then click on the **Add** button.
5. In the image editor, create a new image. The image we chose to use is shown in Figure 14.3.
6. Modify the **Program.cs** file to insert lines 9–14 from Code List 14.1. Then modify `mainWindow.Child = text` to `mainWindow.Child = img` (line 17). By modifying the code yourself, you can get a first-hand experience of writing an embedded code using C#. We believe that even without any explanation, an experienced C# developer can easily understand the code and apply the changes.

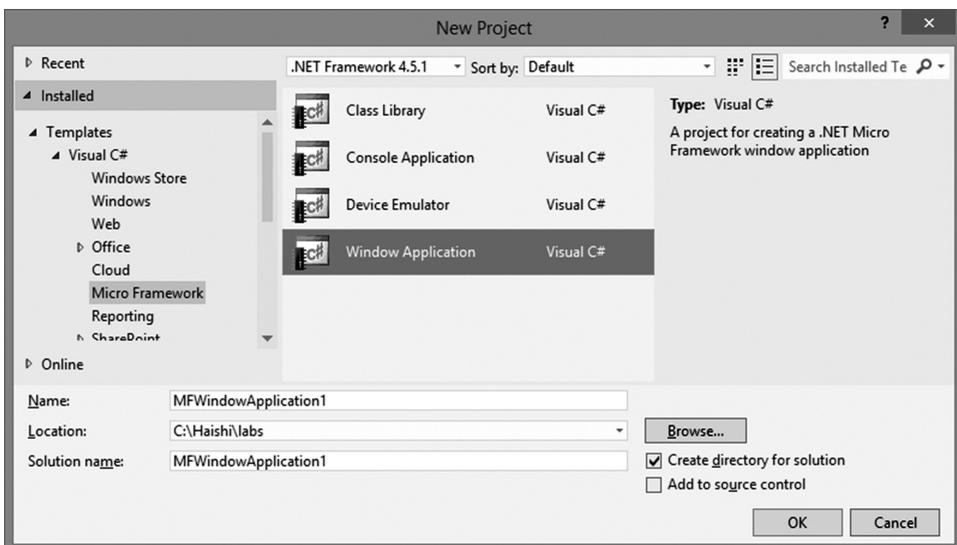


Figure 14.1 New Project dialog.



Figure 14.2 Device emulator.



Figure 14.3 New image resource.

CODE LIST 14.1 MODIFIED PROGRAM.CS

```

1: ...
2: Text text = new Text();
3: text.Font = Resources.GetFont(Resources.FontResources.small);
4: text.TextContent = Resources.GetString(Resources.StringResources.
   String1);
5: text.HorizontalAlignment =
6:     Microsoft.SPOT.Presentation.HorizontalAlignment.Center;
7: text.VerticalAlignment = Microsoft.SPOT.Presentation.
   VerticalAlignment.Center;
8:
9: Image img = new Image(new
10: Bitmap(Resources.GetBytes(Resources.BinaryResources.helloImage),
11: Bitmap.BitmapImageType.Bmp));
12: img.HorizontalAlignment = Microsoft.SPOT.Presentation.
13:     HorizontalAlignment.Center;
14: img.VerticalAlignment = Microsoft.SPOT.Presentation.
15:     VerticalAlignment.Center;
16: // Add the text control to the window.
17: mainWindow.Child = img;
18: ...

```

Note: In Visual Studio 2012, bitmap resources are loaded as binary resources instead of bitmaps. This has been reported as a bug and might be addressed in a future version.

7. Launch the application again, and you will see updated UI as shown in Figure 14.4.

14.4.2 .NET Gadgeteer Overview

We believe that for many curious readers, playing with emulators is not enough fun. So, we are going to introduce Microsoft .NET Gadgeteer, which allows you to play with some real hardware. Microsoft .NET Gadgeteer is an open-source tool kit for developing small appliance prototypes using .NET Microsoft Framework. You can purchase various hardware parts that can combine them into small devices via standard connectors.

At the .NET Gadgeteer home page (address: <http://www.netmf.com/gadgeteer/>), you can find purchase information for various hardware parts, such as mother boards, cameras, network adapters, buttons, touch screens, and SD card readers. Here, we choose to use the Fez Spider Starter Kit from GHI Electronics. The kit is a hardware package that includes a mother board, a camera, two buttons, a network card, a multicolor LED light, an SD card reader, and a color touch LCD screen.

Example 14.2: Simple video camera

Difficulty: ***

This example shows typical steps of .NET Gadgeteer development using Visual Studio 2012 and Fez Spider Starter Kit. Even if you do not have the hardware, you can still get a glimpse of the



Figure 14.4 Updated UI.

process through the detailed steps in this example. Now, let us use Fez Spider Starter Kit to build a simple video camera.

In order to complete this example, you will need the following:

- Visual Studio 2012
- .NET Micro Framework SDK 4.3
- GHI Software Package 4.2 (you need to download from GHI Electronics' support page after registration)
- You should also download and run FEZ Config program to update the firmware. Before upgrading, connect the Power/USB module to the port 1 on the motherboard, and then connect the circuit to your computer. Next, launch FEZ Config on the computer to update the firmware, as shown in Figure 14.5.
 1. In Visual Studio 2012, select **Gadgeteer**→**.Net Gadgeteer Application** to create a new Gadgeteer application.
 2. On the **.NET Gadget Application Wizard** dialog, select **FEZSpider**, and click the **Create** button.
 3. Once the project has been created, you will see a circuit design surface, where you can drag and drop various parts, just as if you were designing a Windows from UI, and connect them into a complete circuit. While making connections, you can directly draw lines between part ports, and the designer will prompt you to the right ports to use as you draw the line. An easier approach is to right-click on the design surface, and select the **Connect all modules** menu to automatically connect all parts. On the design surface, add the following parts:
 - a. Camera (Camera)
 - b. Button (Button)
 - c. Display (Display_T35)
 - d. USB/Power module (UsbClientDP)

Then, use the **Connect all modules** menu to connect them, as shown in Figure 14.6.

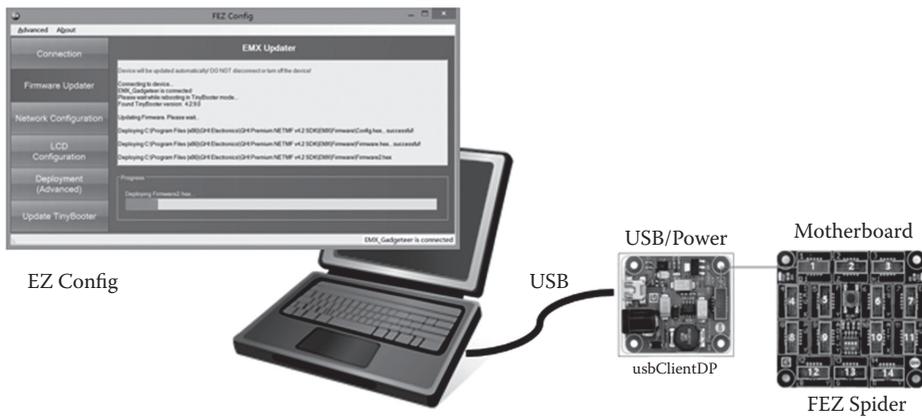


Figure 14.5 Update firmware.

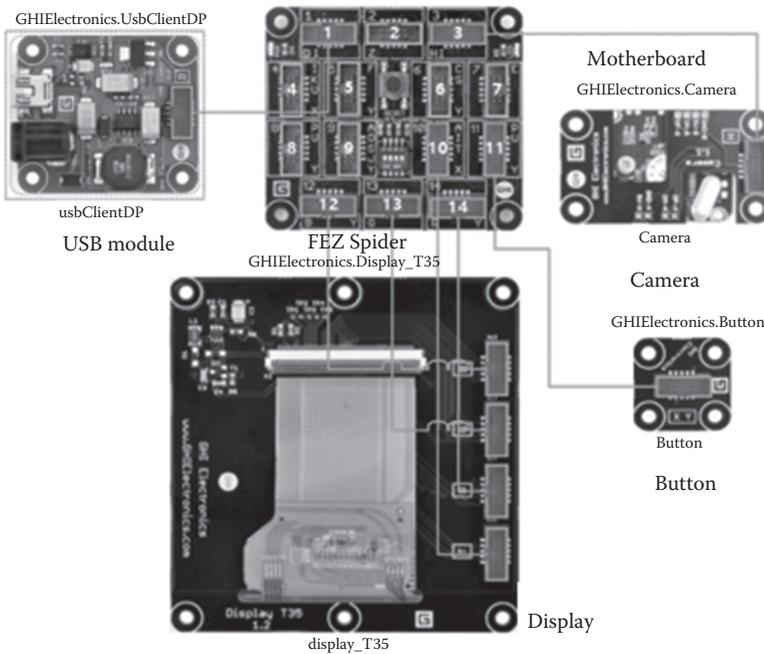


Figure 14.6 Completed circuit.

- Now we can start to code against the circuit. Edit the **Program.cs** file and enter the lines in Code List 14.2. If you are familiar with C#, the following code should be fairly straightforward to understand: we respond to the button pressed event to turn on/off the camera (lines 13–29). When a frame is captured, we will send the image directly to the display (lines 31–34). Note that to get a higher frame rate, we have created a Bitmap local variable (line 3) to serve as a shared buffer between the camera and the display. This works well when the camera is stationary. However, the image is distorted when the camera

CODE LIST 14.2 CAMERA CONTROL PROGRAM

```

1: public partial class Program
2: {
3:     Bitmap mBitmap; //Image buffer
4:     void ProgramStarted()
5:     {
6:         mBitmap = new Bitmap
7:         (camera.CurrentPictureResolution.Width,
8:         camera.CurrentPictureResolution.Height); //Initial buffer to
           camera view size
9:         camera.BitmapStreamed += camera_BitmapStreamed; //Frame
           captured event
10:        button.ButtonPressed += button_ButtonPressed; //Button pressed
           event
11:            button.TurnLEDOff(); //Turn off LED to indicate "off"
           state
12:        }
13:        void button_ButtonPressed(GTM.GHIElectronics.Button sender,
14:            GTM.GHIElectronics.Button.ButtonState state)
15:        {
16:            if (button.IsLedOn) // "on" state
17:            {
18:                camera.StopStreamingBitmaps(); //Stop streaming
19:                button.TurnLEDOff(); //Clear "on" state
20:            }
21:            else
22:            {
23:                if (camera.CameraReady) //Check if camera is
           ready
24:                {
25:                    camera.StartStreamingBitmaps(mBitmap); //
           Start streaming
26:                    button.TurnLEDOff(); //Set "on" state
27:                }
28:            }
29:        }
30:
31:        void camera_BitmapStreamed(GTM.GHIElectronics.Camera sender,
           Bitmap bitmap)
32:        {
33:            display_T35.SimpleGraphics.DisplayImage(mBitmap, 0,
           0); //display frame
34:        }
35:    }

```

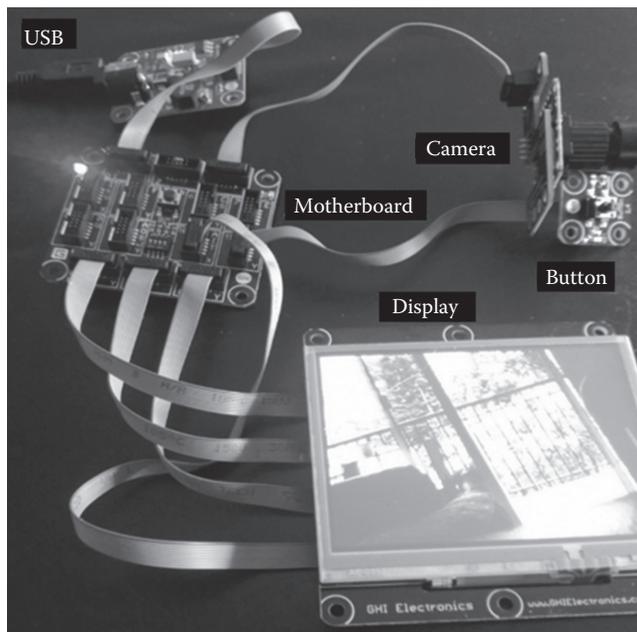


Figure 14.7 Finished product.

moves. Interested readers may attempt to fix the distortion by using a double-buffer, but probably the performance will suffer.

5. Connect the parts as shown in Figure 14.6. Then, connect the whole circuit to the computer via a USB.
6. Now, it is the magical moment. In Visual Studio, press F5 to launch the program. The code will be compiled and deployed to your device. Wait till the LED light dims. Press the button, observe the LED light turning on, and the display displaying the images captured by the camera. Press the button again to capture a still image. Press the button once again to return to live streaming. Figure 14.7 shows a working system.
7. [Optional] You can also set up breakpoints in the code and experience the debugging process. You can step through a code, watch the variable values, and examine the exception details just as if you were debugging a regular PC application. Isn't this convenient?

14.4.3 Device Integration Sample Scenario

A cloud service can connect separate devices together and orchestrate them into a complete, innovative solution. In this section, we will introduce a sample scenario that builds a simple intruder detection system by combining Micro Framework, Windows tablet, and Microsoft Azure Mobile Services together. The system creates a small camera that can be hidden at a certain location in your house. The camera captures and analyzes environmental images, and inserts a record into a Mobile Service data table once it detects any motion. In turn, the Mobile Service notification mechanism is triggered by the record and sends a push notification to your smart phone or tablet. The system diagram is shown in Figure 14.8.

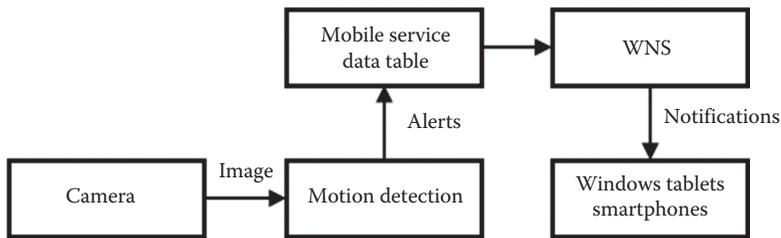


Figure 14.8 System diagram of the intruder detection system.

Although the system is simple, it is a good example of how to combine the advantages of cloud and devices to create useful solutions. First, the system takes advantage of the rich data acquisition capability of devices to collect environmental data. Second, the system takes advantage of the high availability of cloud services to link different devices together. Finally, the system uses the cloud-based push notification service to send alerts to users regardless of their location. Now, let us study how to build such a system.

Example 14.3: Intruder detection system

Difficulty: *****

This example continues from the previous example. We use Visual Studio 2012, Fez Spider Starter Kit from GHI Electronics, Microsoft Azure Mobile Services, and Windows Store application to implement an intruder detection system. This example assumes that you have learned about Microsoft Azure Mobile Services and push notifications. As the process is a bit long, I will omit some of the steps.

1. Continue with Example 14.2. Extend the circuit to include a network card. On the design surface, add an **ethernet_J11D** component, and then connect it to the motherboard, as shown in Figure 14.9.
2. Then, let us implement a simple motion detection algorithm. Code List 14.3 shows the modified *camera_BitmapStreamed* method. The method is simple: it observes the RGB value of the center pixel of the screen. When it sees a sudden change (lines 17 and 18), it sends an alert. The program throttles the alerts (line 22) so that a single event (series of motions within 5 s) is not reported multiple times. Lines 36–42 display the captured images as well as the generated alerts on the screen—this part is optional. If you would like to simplify the circuit by eliminating the display (and display-related code), you are welcome to do so—it does not affect the rest of the system.
3. Next, we need to insert the alert into a Mobile Service data table. Here, we assume you have created a standard to-do application (see Example 13.1). Before we can perform the insertion, we need to solve three problems: syncing up system time on the embedded system; enabling SSL on the embedded system; and authenticating with the Mobile Service. We need to do these because Mobile Service requires HTTPS and authentication for REST API calls. It requires a client clock to be within the allowed clock skew.
4. First, let us enable SSL. We can deploy an SSL seed to the device using the **MFDeploy** tool from the Fez Spider Starter Kit. Run `c:\Program Files (x86)\Microsoft .NET Micro Frameworks\v4.2\Tools\MFDeploy.exe`. Then, select the **Target→Manage Device Keys→UpdateSSLSeed** menu, as shown in Figure 14.10.
5. The update takes only a few seconds, as shown in Figure 14.11.
6. Next, we will use an Internet time server to update the embedded system clock when it is started. The code is shown in Code List 14.4.

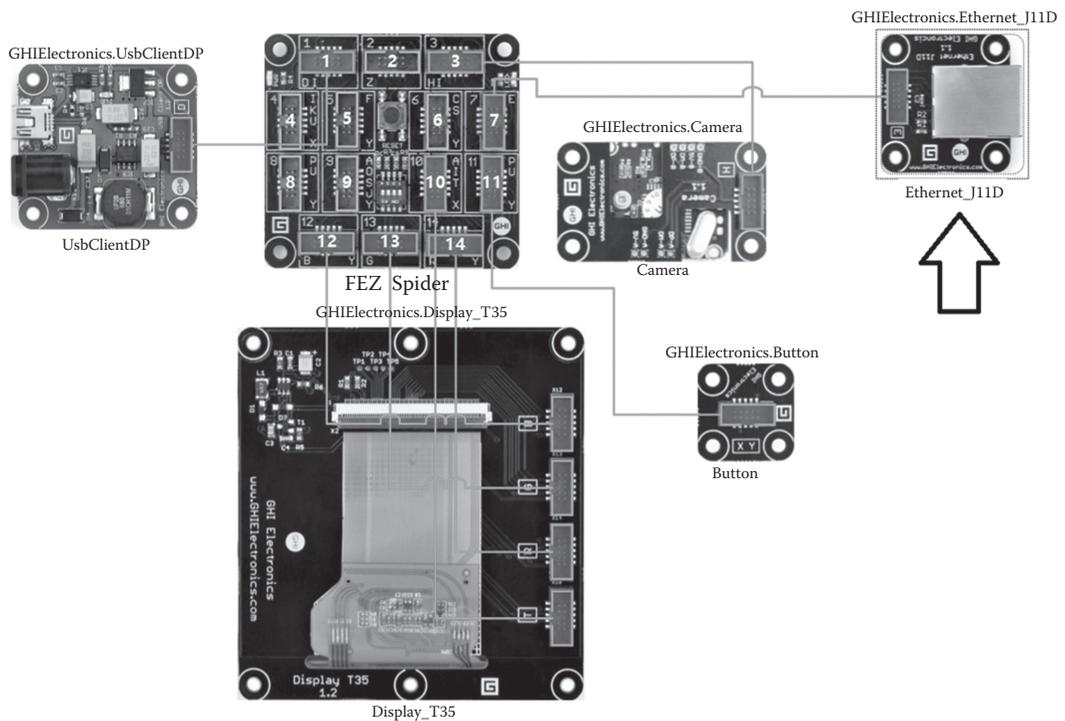


Figure 14.9 Modified circuit.

Note: This code is adapted from

<http://weblogs.asp.net/mschwarz/archive/2008/03/09/wrong-datetime-on-net-micro-framework-devices.aspx>, which uses a time server at time-a.nist.gov.

7. Now, we are ready to invoke Mobile Service's REST API to insert alerts to a Mobile Service data table. Then, Mobile Service will push the alert as a push notification to mobile clients. Here, we assume your data table has the name *ToDoItem*, and its access permission is set to *Anyone with an Application Key*. Note that the push notification part is omitted here. You may refer to Example 13.3 if needed (Code List 14.5).
8. Launch the program as well as the mobile application that accepts the push notification. Wave your hand in front of the camera to trigger an alert, and observe the push notification on the mobile application.

Note: You can change table access permissions on a table's PERMISSIONS page, as shown in Figure 14.12.

To view your application key, use the **MANAGE KEYS** icon on the command bar, as shown in Figure 14.13.

CODE LIST 14.3 SIMPLE MOTION DETECTION ALGORITHM

```

1: bool mFirst = true; //indicating the first pixel read
2: byte mLastR = 0;
3: byte mLastG = 0;
4: byte mLastB = 0;
5: DateTime mLastWarn = DateTime.MinValue;
6: ...
7: void camera_BitmapStreamed(GTM.GHIElectronics.Camera sender,
    Bitmap bitmap)
8: {
9:     var newColor = bitmap.GetPixel(bitmap.Width / 2, bitmap.
    Height / 2);
10:    var r = ColorUtility.GetRValue(newColor);
11:    var g = ColorUtility.GetGValue(newColor);
12:    var b = ColorUtility.GetBValue(newColor);
13:    if (mFirst)
14:        mFirst = false;//first pixel read, record without
        analysis
15:    else
16:        {
17:            if (System.Math.Abs(r - mLastR) + System.Math.Abs(g
    - mLastG)
18:                + System.Math.Abs(b - mLastB) >= 100)//sudden
                change
19:                {
20:                    try
21:                    {
22:                        if ((DateTime.Now - mLastWarn).Seconds > 5)
23:                            {
24:                                mLastWarn = DateTime.Now;
25:                                //TODO: send alert
26:                            }
27:                    }
28:                    catch
29:                    { //TODO: handle exception
30:                    }
31:                }
32:        }
33:    mLastR = r;//record last observed pixel to allow slow drifting
34:    mLastB = b;
35:    mLastG = g;
36:    display_T35.SimpleGraphics.DisplayImage(mBitmap, 0, 0); //
    display image
37:    //optional - display the alert on screen for 5 seconds

```

```

38:     if ((DateTime.Now - mLastWarn).Seconds <= 5)
39:         display_T35.SimpleGraphics.DisplayText("ALERT!",
40:         Resources.GetFont(Resources.FontResources.NinaB),
41:         GT.Color.Red,
42:         (uint)bitmap.Width / 2, (uint)bitmap.Height / 2);
43: }

```

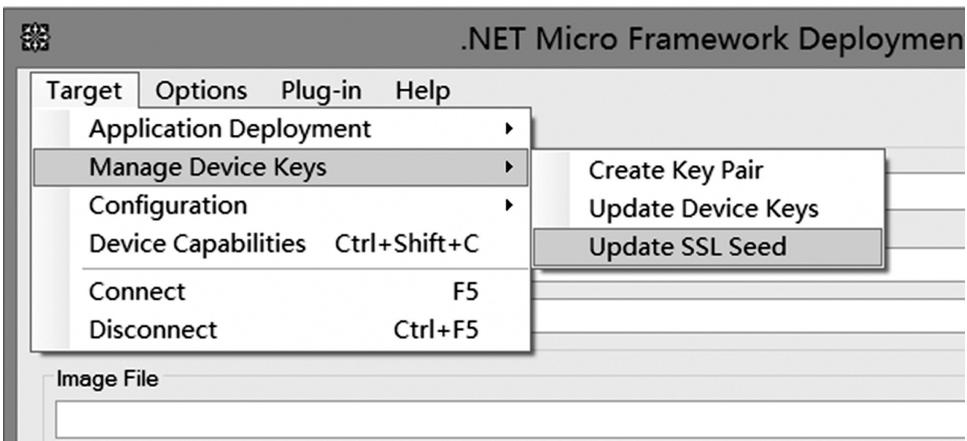


Figure 14.10 MFDeploy tool.

```

Updating SSL seed...
Chk signature
Signature PASS
Update Complete!

```

Figure 14.11 MFDeploy tool run result.

CODE LIST 14.4 UPDATE EMBEDDED SYSTEM CLOCK

```

1: void ProgramStarted()
2: {
3:     ...
4:     Utility.SetLocalTime(GetNetworkTime());
5: }
6: private DateTime GetNetworkTime()
7: {
8:     IPEndPoint ep = new IPEndPoint(Dns.GetHostEntry
9:         ("time-a.nist.gov").AddressList[0], 123);
10:    Socket s = new Socket(AddressFamily.InterNetwork,
11:        SocketType.Dgram, ProtocolType.Udp);
12:    s.Connect(ep);
13:    byte[] ntpData = new byte[48]; // RFC 2030
14:    ntpData[0] = 0x1B;
15:    for (int i = 1; i < 48; i++)
16:        ntpData[i] = 0;
17:    s.Send(ntpData);
18:    s.Receive(ntpData);
19:    byte offsetTransmitTime = 40;
20:    ulong intpart = 0;
21:    ulong fractpart = 0;
22:    for (int i = 0; i <= 3; i++)
23:        intpart = 256 * intpart + ntpData[offsetTransmit
24:            Time + i];
25:    for (int i = 4; i <= 7; i++)
26:        fractpart = 256 * fractpart +
27:            ntpData[offsetTransmitTime + i];
28:    ulong milliseconds = (intpart * 1000 + (fractpart * 1000) /
29:        0x100000000L);
30:    s.Close();
31:    TimeSpan timeSpan = TimeSpan.FromTicks
32:        ((long)milliseconds * TimeSpan.TicksPerMillisecond);
33:    DateTime dateTime = new DateTime(1900, 1, 1);
34:    dateTime += timeSpan;
35:    TimeSpan offsetAmount = TimeZone.CurrentTimeZone.
36:        GetUtcOffset(dateTime);
37:    DateTime networkDateTime = (dateTime + offsetAmount);
38:    Debug.Print(networkDateTime.ToString());
39:    return networkDateTime;
40: }

```

CODE LIST 14.5 SEND ALERTS TO MOBILE SERVICES

```

1: var timestamp = DateTime.Now;
2: string data = "{\"text\": \"Intruder at \" + timestamp.Hour + \":\" +
3:   timestamp.Minute + \":\" + timestamp.Second + "\",
4:   \"complete\": false}";
5: byte[] bytes = System.Text.Encoding.UTF8.GetBytes(data);
6: HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create
7:   ("https://[your Mobile Service].azure-mobile.net/tables/
8:   TodoItem");
9: request.Method = "POST";
10: request.Accept = "application/json";
11: request.ContentType = "application/json";
12: request.ContentLength = bytes.Length;
13: request.Headers.Add("X-ZUMO-APPLICATION", "[your
14:   application key]");
15: using (var stream = request.GetRequestStream())
16: {
17:   stream.Write(bytes, 0, bytes.Length);
18: }
19: var response = request.GetResponse();
20: response.Close();

```

todoitem

BROWSE SCRIPT COLUMNS PERMISSIONS

table permissions

INSERT PERMISSION



UPDATE PERMISSION



DELETE PERMISSION



READ PERMISSION



Figure 14.12 Mobile Service table access permissions.

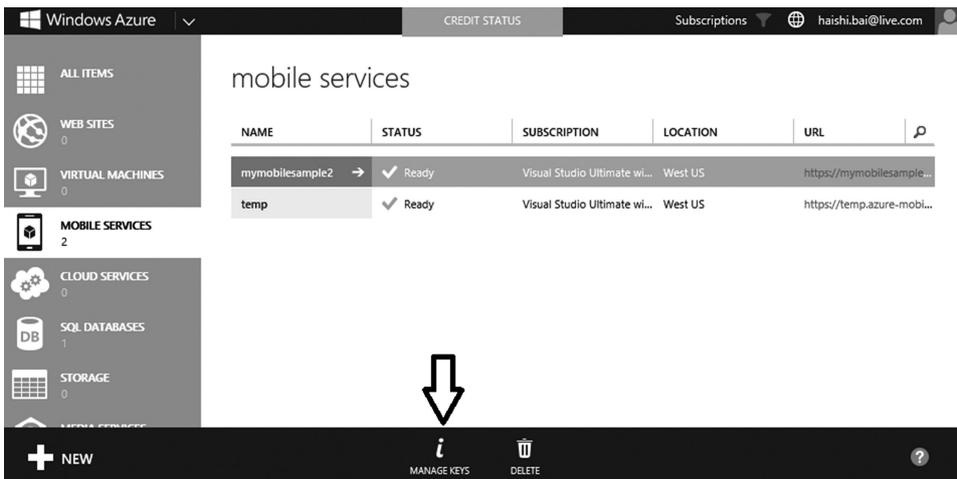


Figure 14.13 View application key.

14.5 Summary

In this chapter, we provided a brief overview of IoT. Then we discussed how devices and cloud complement each other, and how they can create powerful solutions when combined together. Through several examples, we learned how to develop an embedded code using Microsoft Micro Framework and .NET Gadgeteer. Finally, we worked on a scenario that combined .NET Gadgeteer, Windows Store application, and Mobile Services to provide a complete scenario.

SYSTEM INTEGRATION AND PROJECT MANAGEMENT

IV

What is the number one benefit of cloud? Agility. Cloud allows you to go to the market faster, to respond to market changes quicker, and to innovate at an unprecedented pace. This section of the book discuss this pheromone from two different views. First, we discuss integration, focusing on how to orchestrate on-premises workloads and cloud workloads together to construct complete solutions. A flexible integration strategy allows different pieces of a large system to move at different paces so that as you embrace the agility of the cloud you can still work with legacy systems that are staying behind for various reasons. Then, we study how Microsoft tools and services enable DevOps scenarios to help you to build an agile innovation team to produce next-generation cloud-based solutions.

Chapter 15

Message-Based System Integration

15.1 System Integration

Many enterprises, especially large and medium-sized enterprises, often have many software and hardware solutions deployed and used at the same time. How to effectively integrate these systems to provide streamlined workflows across the enterprise is a practical problem faced by many of these enterprises. Many factors contribute to IT fragmentation within an enterprise. For example, different departments may have different paces of adopting new technologies, conflicting interests, diverse working styles, and strategies, especially when different teams originate from different companies because of acquisitions, reorganizations, and mergers.

When a cloud service provider introduces Cloud to enterprises, it should be ready to address the needs of integrating the new cloud services with hundreds of existing systems. It needs to provide a smooth transition path for enterprise users to adopt cloud services over time. At the same time, when it extends existing enterprise applications to cloud, it often needs to migrate existing systems piece by piece. So, for a considerable period of time, it will face the situation that different parts of a bigger system work together over different environments, such as on-premise, private cloud, community cloud, and public cloud.

Although there are many different strategies and methodologies for system integration, we can still abstract some common rules and patterns. In this chapter, we will first introduce some common integration patterns, and then focus on how to integrate systems using message-based system integration. Message-based integration is a strategy that is the result of tens of years of collective experience in system integration, and has been proven to be a flexible, maintainable, and extensible integration strategy. The majority of this chapter will focus on implementing system integration patterns using Microsoft Azure Service Bus.

It is worth pointing out that system integration is not just a technical problem, but also a problem that involves business workflow and sometimes strategy-level changes. This book discusses only the technical aspects of the problem.

Common system integration patterns include the following:

15.1.1 *Integration by Data*

Integration by data means different systems are integrated by sharing the same data. This pattern has the following common implementation methods:

- **Data import/export**

Data from a system are exported in a common format, such as CSV and XML. Then, the data are imported to another system by either automatic or manual operations. Because data import/export happens in batches, this method does not provide real-time integration. The downside of the method includes unreliability, inefficiency, and unresponsiveness, especially when manual operations are involved. The upside of such methods includes simplicity, minimum impact on existing systems, and flexible data transfer mechanism (including transferring data using a USB key!).

- **Data replication**

Data replication automates the import/export process, so it is a much more reliable method. Many modern databases and data services provide data replication or synchronization capabilities to keep separate data repositories in sync. Data repositories participating in the replication processes can share the same schema, or have their schemas mapped via some sort of transformation service, which is often provided by a middleware instead of the database itself. The shared schema provides higher efficiency; however, it creates a tighter coupling between the participating systems.

- **Shared repository**

Shared repository avoids data import/export and synchronization altogether. Systems simply access the same data repository to work together. Obviously, this method provides a simple, real-time integration solution. However, in this case, systems are tightly coupled together. The shared repository can easily become a system bottleneck and a Single Point of Failure (SPoF).

15.1.2 *Shared Business Functions*

With Shared Business Functions, some systems expose common business functions that can be used across the enterprise. This is a common integration pattern, and many techniques have been developed to support it. Essentially, all techniques for distributed systems and remote calls can be leveraged to implement Shared Business Functions, such as DCOM, CORBA, JavaRMI, .NET Remoting, Web Service, and Wep API. For the Shared Business Functions to work, systems need to agree on addressing, contracts, and protocols. Direct communications require all participants to be online at the same time, and their throughputs to match up. Whenever any of the addressing, contracts, or protocols change, all related systems need to be updated to reestablish the connection.

Shared Business Functions are sometimes offered by one of the systems that participate in the integration. However, as system integration often has impact on workflows across multiple departments, many enterprises choose to isolate these shared functions from existing systems, and to create an independent department that is responsible for the shared functions.

15.1.3 *Enterprise Service Bus*

Enterprise Service Bus (ESB) consolidates the functionalities of Service-Oriented Architecture (SOA), Enterprise Application Integration (EAI), and Message-Oriented Middleware. Instead of focusing on an abstract definition of ESB, we can understand it from the following aspects.

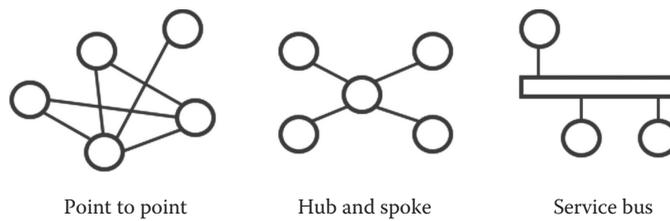


Figure 15.1 System topologies in integration.

■ SOA

ESB provides a candidate architecture for SOA implementations. SOA emphasizes that systems be loosely coupled by contracts. However, SOA does not mandate the topology of the participating systems. Roughly speaking, we can put system topologies into three categories: Point to Point, Hub and Spoke, and Service Bus, as shown in Figure 15.1.

The major disadvantage of the Point-to-Point topology is maintainability. For example, n full connected systems require a total of $n(n - 1)/2$ connections, while the other two topologies only need n and $n - 1$ connections, respectively. The main disadvantage of the Hub and Spoke topology is that the Hub is an SPoF of the integrated system, jeopardizing performance and reliability of the overall system. Service Bus is a Hub and Spoke topology as well. The main difference resides in how the Hub is constructed. With Service Bus, the Hub is constructed by distributed, redundant services so that the availability, scalability, and reliability of the Hub itself can be assured. This overcomes the main shortcomings of the Hub and Spoke topology and provides a flexible, maintainable, and available integration solution.

■ Enterprise Application Integration

ESB allows systems built on different platforms to be integrated together via common message brokers. With ESB, the message exchange, transformation, and routing rules are defined independently from participating systems. This is a very important characteristic, if you think about it. With ESB, application developers can focus on applications themselves and leave system integration details to ESB. All they need to do is to define input/output endpoints that the applications expose. Obviously, when individual applications are developed, the exact requirements of system integration are often undefined. Instead of asking developers to guess how the systems would be integrated in the future, it is better to hand the integration off to a dedicated party. Furthermore, because integration rules can be maintained independently from loosely coupled applications, the integrated system as a whole is more adaptive to requirement changes.

■ Message-Oriented Middleware

ESB also provides systems with a message-oriented middleware, which provides message publishing, filtering, monitoring, auditing, and throttling. ESB provides a common message exchange platform for all participating systems, allowing them to exchange data without knowing any details of each other. In addition, message-oriented middleware provides capabilities such as message transformation, routing, and buffering. These features allow systems with different message formats, different throughputs, and different availabilities to work together.

15.2 Message-Based System Integration

We have given an introduction to Microsoft Azure Service Bus Queue and Topic/Subscription in Section 8.3. In this section, we will introduce four specific integration patterns using Microsoft Azure Service Bus. There are hundreds of message-based system integration patterns, which cannot be covered in this book. Here we have picked the five most representative patterns that show different characteristics of message-based integration. Interested users should consult related books listed in the reference material at the end of this book.

15.2.1 Content-Based Routing

In many integration scenarios, messages do not follow fixed paths. For example, a message sender may need to send messages to different business partners based on message contents. In other words, a message sender should be able to route messages to different message receivers based on message contents. Here are some typical ways of implementing content-based routing:

- Custom-logic router

A custom-logic router maintains a list of all possible message recipients. It forwards messages to different recipients based on message contents and predefined routing rules. Although custom-logic routers have the power of supporting arbitrary routing logics, they are less flexible and harder to maintain because they have to keep the recipient list up-to-date. A custom-logic router pattern is depicted in Figure 15.2.

- Message filters

With this implementation, a message sender broadcasts messages to all recipients, and the recipients use their corresponding filters to filter out unwanted messages (see Figure 15.3). There are two disadvantages of this approach. First, because the messages are broadcasted, a greater number of messages flow through the system, while most of them will be eventually discarded (filtered). Second, the message sender needs to ensure that the filters are not tampered with by ill-purposed recipients to gain access to messages that are not meant for them. However, the method also has obvious advantages. First, because there is no recipient list to be maintained, it is easier to maintain and to extend the system. Second, the message sender in this case defines the routing rules (filters) based on business scenarios instead of defining the rules per recipient.

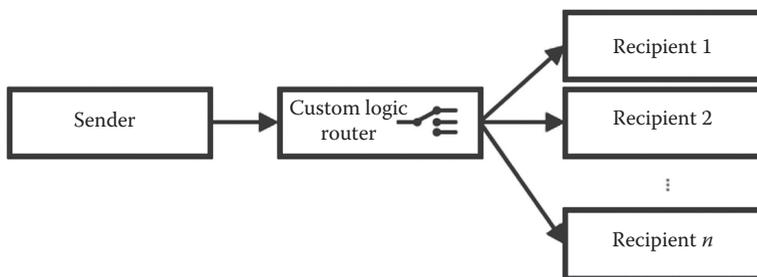


Figure 15.2 Custom-logic router.

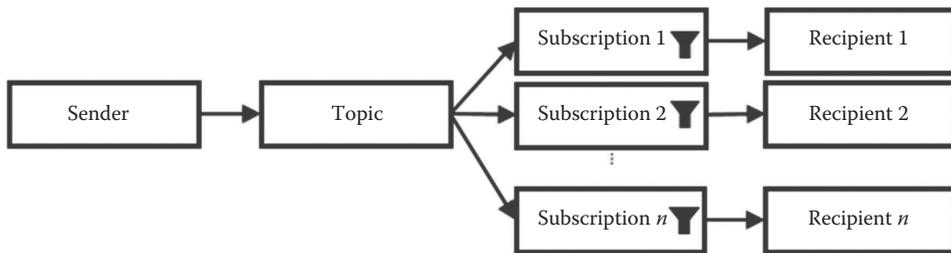


Figure 15.3 Message filters.

Example 15.1: Content-based message routing

Difficulty: *

In this example, three enterprises need to exchange order information with each other: Enterprise A needs to send some orders to either Enterprise B or C, based on order values; Enterprise B is capable of handling high-valued orders; and Enterprise C handles only smaller orders. In order to implement such integration, Enterprise A has created an Order topic, with two levels of subscriptions—“high-valued” and “low-valued.” The high-valued subscription is assigned to Enterprise B and the low-valued subscription is assigned to Enterprise C. Enterprises B and C are allowed to use the subscriptions, but they do not have rights to alter the subscriptions or associated filters. This is to ensure that each partner gets only the messages that are intended for him or her.

1. Create a new **Orders** topic in your Microsoft Azure Service Bus namespace.
2. Launch Visual Studio, and create a new Windows Console application named **EnterpriseA**.
3. Add a reference to the **WindowsAzure.ServiceBus** NuGet package.
4. Modify the **Microsoft.ServiceBus.ConnectionString** setting in **App.Config** to enter the connection string for your Service Bus namespace.
5. Add a reference to the **System.Configurations** assembly.
6. Modify the **Main()** method of the program. The modified code is shown in Code List 15.1.
7. Add another Windows Console application named **Partner** to the solution.
8. In the **Partner** project, add a reference to the **WindowsAzure.ServiceBus** NuGet package.
9. Modify the **Main()** method. The modified code is shown in Code List 15.2.
10. Set both projects as startup projects, and then press F5 to launch the applications. Copy and paste the secret key for Enterprise B to the receiver program, and press enter to start listening for incoming orders. In the sender program, send several orders with different values. As shown in Figure 15.4, orders #2 and #4 went through because their values were larger or equal to 10,000 dollars. Orders #1 and #3 were filtered because of their low values.

15.2.2 Priority Queue

Items in a priority queue are assigned with different priorities. Items with the highest priority values are retrieved from the queue first, ahead of items with lower priorities. Although enterprises vastly differ from each other and their businesses vary greatly, arranging work according to some sort of priority is a very common scenario. At this point of time, Microsoft Azure Service Bus does not have built-in support for priority queueing, but we can easily simulate a priority queue with Topics and Subscriptions. We create a separate subscription for each priority level. When a client checks for messages, it loops through subscriptions in the order of priorities so that high-priority messages can be read first. This method suits only the situations where the number of priority levels is limited. Because the architecture and code are quite similar to those of Example 15.1, we will leave implementing a priority queue as an exercise for interested readers.

CODE LIST 15.1 MESSAGE SENDER

```

static void Main(string[] args)
{
    string topicName = "Orders";
    NamespaceManager nm = NamespaceManager.CreateFromConnectionString
        (ConfigurationManager.AppSettings
         ["Microsoft.ServiceBus.ConnectionString"]);
    if (nm.TopicExists(topicName))
        nm.DeleteTopic(topicName);
    TopicDescription topic = new TopicDescription(topicName);
    string keyB = SharedAccessAuthorizationRule.GenerateRandomKey();
    string keyC = SharedAccessAuthorizationRule.GenerateRandomKey();
    //Create SAS for partners
    topic.Authorization.Add(
        new SharedAccessAuthorizationRule("ForCompanyB", keyB,
        new AccessRights[] { AccessRights.Listen }));
    topic.Authorization.Add(
        new SharedAccessAuthorizationRule("ForCompanyC", keyC,
        new AccessRights[] { AccessRights.Listen }));
    nm.CreateTopic(topic);

    Console.WriteLine("Enterprise B secret key: " + keyB);
    Console.WriteLine("Enterprise C secret key: " + keyC);
    //Subscription B accepts orders that worth greater or equal to
    10,000 dolloars;
    //Subscription C accepts orders that worth less than 10,000
    dollors;
    nm.CreateSubscription(topicName, "high_value",
        new SqlFilter("value >= 10000")); //Filter by the value
        property
    nm.CreateSubscription(topicName, "low_value",
        new SqlFilter("value < 10000")); //Filter by the value
        property

    TopicClient client = TopicClient.CreateFromConnectionString
        (ConfigurationManager.AppSettings
         ["Microsoft.ServiceBus.ConnectionString"], topicName);
    int orderNumber = 1;
    while (true)
    {
        Console.Write("Enter the amount of Order #" + orderNumber +
            ": ");
        string value = Console.ReadLine();
        int orderValue = 0;
        if (int.TryParse(value, out orderValue))
        {
            var message = new BrokeredMessage();
            message.Properties.Add("order_number", orderNumber);
        }
    }
}

```

```

        message.Properties.Add("value", orderValue); //Define the
            value property
        client.Send(message);
        Console.WriteLine("Order #" + orderNumber + " has been
            sent!");
        orderNumber++;
    }
    else
    {
        if (string.IsNullOrEmpty(value))
            break;
        else
            Console.WriteLine("Invalid input, please try
                again.");
    }
}
}
}

```

CODE LIST 15.2 MESSAGE RECEIVER

```

static void Main(string[] args)
{
    Console.Write("Please enter your secret key: ");
    string key = Console.ReadLine();
    //Access the subscription using given SAS
    MessagingFactory factory = MessagingFactory.Create(
        ServiceBusEnvironment.CreateServiceUri("sb", "[your service bus
            namespace",
                string.Empty),
                TokenProvider.CreateSharedAccessSignatureTokenProvider(
                    "ForCompanyB", key));
    SubscriptionClient client = factory.CreateSubscriptionClient(
        ("Orders", "high_value");
    Console.WriteLine("Waiting for messages...");
    while (true)
    {
        var message = client.Receive();
        if (message != null)
        {
            Console.WriteLine(string.Format("Received Order:  #{0}.
                Value: {1}",
                    message.Properties["order_number"],
                    message.Properties["value"]));
            message.Complete(); //mark the message as complete
        }
    }
}
}

```



Figure 15.4 Content-based routing example.

15.2.3 Request/Response

A request/response pattern is also a common integration pattern. In many situations, a message sender is interested not only in sending messages to a receiver, but also in receiving results. When message-based integration is used, because message senders and receivers work asynchronously, a message sender cannot simply wait for the results, but needs to use an additional channel for retrieving results. A simple solution is to use two queues, one for sending messages and the other for receiving messages, as shown in Figure 15.5.

The problem is, however, if we use a single queue to return all results, and one of the clients stops working with a result for it is at the top of the queue, all other clients are blocked because of the FIFO nature of the queue. On the other hand, if we use a separate queue for each client, we will have to manage a large number of queues. Microsoft Azure Service Bus supports sessions, which can be used to solve this problem nicely. We can create multiple sessions on the same queue, and each client can use its own session to communicate with the server. When a client sends a message, it can mark the message with a session id and then simply listen to the same session id for response. When the server sends a response back, it marks the returning message with the same session id so it can be retrieved only by the designated client.

Now let us see how this is implemented with an example.

Example 15.2: Request/response pattern

Difficulty: **

In this example, an order-processing system takes and processes orders from multiple clients. For simplicity, we will use a single Windows Console application to simulate one order-processing service and four attached clients. Each client sends order requests to the service via an order queue, and the service returns the processing results to the client. Clients are isolated using Service Bus queue sessions.

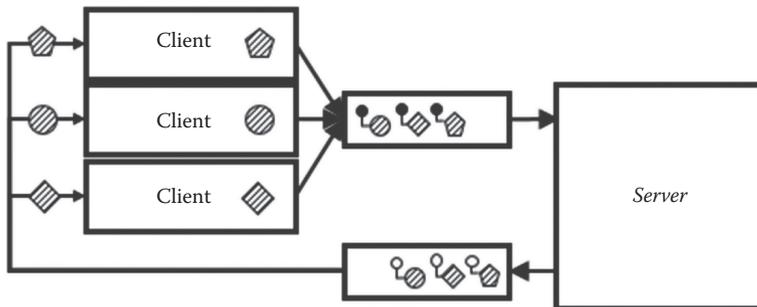


Figure 15.5 Request/response pattern.

1. Create a new Windows Console application.
2. Add a reference to the **WindowsAzure.ServiceBus** NuGet package and a reference to the **System.Configuration** assembly.
3. Modify the **Microsoft.ServiceBus.Connection** setting in the **App.config** file to enter the connection string to your Service Bus namespace.
4. Modify the **Main()** method. You may consult the comments in Code List 15.3 for further details.
5. Launch the program. Figure 15.6 shows the screenshot of a sample run. The application uses a five-column display, where the first four are for the clients and the last one is for the server. You can read each column separately from top to bottom. As you can see, although events from different clients are interleaved, each client communicates with the server independently without interrupting the others.

15.2.4 Dead Letter Queue

Dead Letter Queue is for archiving messages that are not processed by a receiver under the designated time limit or other constraints. These messages are kept in the Dead Letter Queue to be processed later by a cleaner program (see Figure 15.7). Readers should note that in this situation the messages become invalid (such as expired, or they exceed maximum allowed failures or errors), so they should not be directly fed to receivers anymore. During the cleaning process, the system should extract related information from these messages for auditing, debugging, or regenerating the messages before purging them.

Microsoft Azure Service Bus provides built-in support for dead letters in its Queues and Topics. It supports the following ways for placing a message into a Dead Letter Queue:

- **Explicit dead lettering**
A message receiver can put a message into the Dead Letter Queue by explicitly calling the **DeadLetter** method on the **BrokeredMessage** class.
- **Dead lettering by expiration**
When creating a Queue or Subscription, you can set the **EnableDeadLetteringOnMessageExpiration** property to true on the corresponding entity description. This enables expired messages to be automatically put into the Dead Letter Queue.
- **Dead lettering after maximum number of deliveries**
A message is put into the Dead Letter Queue once it exceeds the maximum number of deliveries.
- **Dead lettering by filter exceptions**
For a Subscription, if the **EnableDeadLetteringOnFilterEvaluationException** property of its corresponding **SubscriptionDescription** is true, then the message is put into the Dead Letter Queue when a filter throws an exception.

Now let us see an example of using a Dead Letter Queue.

Example 15.3: Dead Letter Queue

Difficulty: *

In this example, we will write a simple demo program. First, we will simulate a message sender who sends 10 messages to a message queue. Then, we will simulate a receiver receiving and processing these messages. The receiver will deliberately put every other message into a Dead Letter Queue. Finally, we will simulate a cleaner who goes through the Dead Letter Queue for dead letters.

CODE LIST 15.3 REQUEST/RESPONSE PATTERN

```

static void Main(string[] args)
{
    string requestQueue = "requestQueue";//request queue
    string responseQueue = "responseQueue";//response queue
    string connString = ConfigurationManager.AppSettings
        ["Microsoft.ServiceBus.ConnectionString"];
    object padLock = new object();
    ConsoleColor[] colors = new ConsoleColor[] {
        ConsoleColor.Green, ConsoleColor.Cyan,
        ConsoleColor.Red, ConsoleColor.Yellow }; //identiy
        clients with                                //different colors
    NamespaceManager nm = NamespaceManager.CreateFromConnection
        String(connString);
    if (nm.QueueExists(requestQueue))
        nm.DeleteQueue(requestQueue);
    if (nm.QueueExists(responseQueue))
        nm.DeleteQueue(responseQueue);
    //the request queue doesn't need session support
    nm.CreateQueue(new QueueDescription(requestQueue));
    //the response queue requires sessions
    nm.CreateQueue(new QueueDescription(responseQueue) {
        RequiresSession = true });
    //spawn 4 threads to simulate 4 clients
    for (int i = 0; i < 4; i++)
    {
        ThreadPool.QueueUserWorkItem((o) =>
        {
            QueueClient sender = QueueClient.
                CreateFromConnectionString
                    (connString, requestQueue);
            QueueClient receiver = QueueClient.
                CreateFromConnectionString
                    (connString, responseQueue);
            //create reception session
            var session = receiver.AcceptMessageSession
                (o.ToString());
            for (int j = 0; j < 5; j++)
            {
                lock (padLock)
                {
                    Console.ForegroundColor = colors[(int)o];
                    Console.WriteLine(string.Format("{0} Sending
                        Order #{1}.",
                            new string('\t', (int)o), j));
                }
                //send order message
                sender.Send(new BrokeredMessage

```

```

        {
            ReplyToSessionId = o.ToString(), //mark session id
            MessageId = j.ToString()
        });
        var message = session.Receive(); //receive result on
        session
        if (message != null)
        {
            lock (padLock)
            {
                Console.ForegroundColor = colors[(int)o];
                Console.WriteLine(string.Format
                    ("{0} Order #{1} Processed.",
                    new string('\t', (int)o), message.
                    MessageId));
            }
            message.Complete();
        }
    }
    }, i);
}
//start server thread
ThreadPool.QueueUserWorkItem((o) =>
{
    QueueClient receiver = QueueClient.CreateFromConnectionString
        (connString, requestQueue);
    QueueClient sender = QueueClient.CreateFromConnectionString
        (connString, responseQueue);
    while (true)
    {
        var message = receiver.Receive(); //receive order
        if (message != null)
        {
            //send returning message
            sender.Send(new BrokeredMessage
            {
                SessionId = message.ReplyToSessionId, //mark
                session id
                MessageId = message.MessageId
            });
            lock (padLock)
            {
                Console.ForegroundColor = ConsoleColor.White;
                Console.WriteLine(string.Format
                    ("{0} Processing Order #{2} from Client {1}",
                    new string('\t', 4), message.
                    ReplyToSessionId,
                    message.MessageId));
            }
        }
    }
}

```

```

        message.Complete();
    }
}
});
Console.ReadLine();
}

```

```

Sending Order #0.      Sending Order #0.
Sending Order #0.      Sending Order #0.
Sending Order #0.      Processing Order #0 from Client 0
Order #0 Processed.   Processing Order #0 from Client 2
Sending Order #1.      Order #0 Processed.
                        Sending Order #1.
                        Order #0 Processed.
                        Processing Order #0 from Client 3
                        Sending Order #1.
                        Processing Order #0 from Client 1
Order #0 Processed.   Order #0 Processed.
Sending Order #1.      Sending Order #1.

```

Figure 15.6 Request/response pattern example.

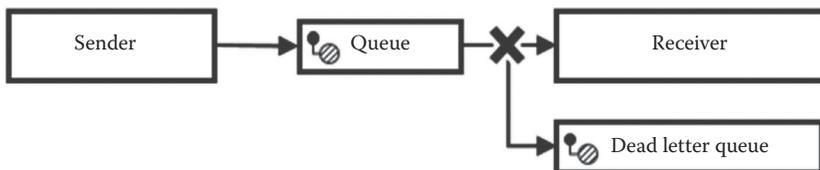


Figure 15.7 Dead Letter pattern.

1. Create a new Windows Console application.
2. Add a reference to the **WindowsAzure.ServiceBus** NuGet package and a reference to the **System.Configuration** assembly.
3. Modify the **Microsoft.ServiceBus.ConnectionString** setting in the **App.config** file to enter the connection string to your Service Bus namespace.
4. Modify the **Main()** method. For details, you may see the comments in Code List 15.4.
5. Launch the program and observe the Dead Letter Queue at work, as shown in Figure 15.8.

15.2.5 Event-Driven Consumer

When participating in message exchange using a Queue or a Topic/Queue, a message receiver often periodically polls the corresponding messaging entity for new messages. This is the method we have used in all previous examples. Such consumers who periodically poll for messages are

CODE LIST 15.4 DEAD LETTER QUEUE DEMO

```

static void Main(string[] args)
{
    string connString = ConfigurationManager.AppSettings
        ["Microsoft.ServiceBus.ConnectionString"];
    string queueName = "samplequeue";
    NamespaceManager nm = NamespaceManager.CreateFromConnectionString
        (connString);
    if (nm.QueueExists(queueName))
        nm.DeleteQueue(queueName);
    nm.CreateQueue(queueName);
    //Simulate a message sender sending 10 messages
    QueueClient sender = QueueClient.CreateFromConnectionString
        (connString, queueName);
    for (int i = 0; i < 10; i++)
        sender.Send(new BrokeredMessage { MessageId =
            i.ToString() });
    //Simulate a message receiver
    QueueClient receiver = QueueClient.CreateFromConnectionString
        (connString, queueName);
    for (int i = 0; i < 10; i++)
    {
        var message = receiver.Receive();
        if (message != null)
        {
            if (i % 2 == 0)
            {
                Console.WriteLine(string.Format("Message {0} is
                    processed.",
                        message.MessageId));
                message.Complete();
            }
            else
            {
                //Add every other message to the dead letter queue,
                //and provide a text scription of the reason of dead
                lettering.
                message.DeadLetter("Unprocessed", "Dead letter queue
                    test.");
            }
        }
    }
    //Simulate a cleaner to clean up the dead letter queue. Note
    QueueClient
    //provides a FormatDeadLetterPath method which helps to find the
    corresponding
    //dead letter queue of a Service Bus queue.

```

```

QueueClient deadLetterClient = QueueClient.
    CreateFromConnectionString
        (connString, QueueClient.FormatDeadLetterPath(queueName),
        ReceiveMode.ReceiveAndDelete);
BrokeredMessage deadLetter;
while ((deadLetter = deadLetterClient.Receive()) != null)
{
    Console.WriteLine(
        string.Format("Retrieved message {0} from the dead
            letter queue.",
            deadLetter.MessageId));
    Console.WriteLine(
        string.Format("            Message {0} is dead-lettered
            because of: {1}",
            deadLetter.MessageId,
            deadLetter.Properties["DeadLetterErrorDescription"]));
}
}

```

```

Message 0 is processed.
Message 2 is processed.
Message 4 is processed.
Message 6 is processed.
Message 8 is processed.
Retrieved message 1 from the dead letter queue.
    Message 1 is dead-lettered because of: Dead letter queue test.
Retrieved message 3 from the dead letter queue.
    Message 3 is dead-lettered because of: Dead letter queue test.
Retrieved message 5 from the dead letter queue.
    Message 5 is dead-lettered because of: Dead letter queue test.
Retrieved message 7 from the dead letter queue.
    Message 7 is dead-lettered because of: Dead letter queue test.
Retrieved message 9 from the dead letter queue.
    Message 9 is dead-lettered because of: Dead letter queue test.

```

Figure 15.8 Dead Letter Queue demo.

called Polling Consumers. A Polling Consumer runs a continuous loop, which works fine with a backend program. However, for a client application that often uses event-driven programming, the continuous polling does not fit in very well. The so-called event-driven Consumers do not poll for changes. Instead, they register callbacks with the corresponding messaging entities and wait for the callbacks to be invoked, as shown in Figure 15.9.

Next, let us see this pattern in action with an example.

Example 15.4: Event-Driven Consumer

Difficulty: *

In this example, we will write a simple WPF client, which acts as both a message sender and a message receiver. A user can click on the “Send” button on the UI to send a message, and observe messages appearing on the screen when new message events are triggered.

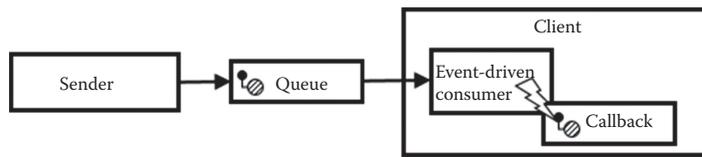


Figure 15.9 Event-driven consumer pattern.

1. Create a new WPF application.
2. Add a reference to the **WindowsAzure.ServiceBus** NuGet package.
3. Modify the **MainWindow.xaml** file to implement a simple messaging UI:

```

<Window x:Class="Example15.4.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
            presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525" FontSize="18">
<StackPanel>
    <DockPanel HorizontalAlignment="Stretch">
        <TextBlock Margin="5">Message:</TextBlock>
        <Button Margin="5" x:Name="sendMessage" DockPanel.
            Dock="Right"
                Click="sendMessage_Click_1">Send</Button>
        <TextBox x:Name="messageText" Margin="5"/>
    </DockPanel>
    <ListBox x:Name="messageList" />
</StackPanel>
</Window>
    
```

4. Modify the **MainWindow.xaml.cs** file. Lines 18–25 in Code List 15.5 listen to queue events and add corresponding messages to the display list. Specifically, the **MaxConcurrentCalls** parameter in line 25 specifies how many concurrent calls can be made on the callback. If this value is larger than 1, then the callback function may be concurrently called multiple times when many messages arrive at the same time.
5. Press F5 to launch the program. Send several messages to add them to the display list, as shown in Figure 15.10.

15.3 Advanced Message Queuing Protocol

Traditionally, most middleware uses proprietary protocols and private formats, which lead to various interoperability issues. Nowadays, more and more software vendors and service providers are taking a more open approach by creating and supporting common standards. Advanced Message Queuing Protocol (AMQP) is an open protocol for message-based middleware. It defines a common standard for enterprises to exchange data safely, reliably, and efficiently.

AMQP was conceived by JPMorgan Chase in 2003. The purpose of the standard is primarily to facilitate exchange of financial data. Soon, many famous enterprises, such as Bank of America, Goldman Sachs, Microsoft, Cisco, and Red Hat, joined the effort to define and promote the protocol. In August 2011, AMQP workgroup became a formal member of OASIS. OASIS released

CODE LIST 15.5 MESSAGE SEND/RECEIVE LOGIC

```

1: public partial class MainWindow : Window
2: {
3:     public MainWindow()
4:     {
5:         InitializeComponent();
6:         //recreate the queue
7:         NamespaceManager nm = NamespaceManager.
            CreateFromConnectionString
8:             (connString);
9:         if (nm.QueueExists(queueName))
10:            nm.DeleteQueue(queueName);
11:         nm.CreateQueue(queueName);
12:         //create sender and receiver
13:         mSender = QueueClient.CreateFromConnectionString
14:             (connString, queueName);
15:         mReceiver = QueueClient.CreateFromConnectionString
16:             (connString, queueName);
17:         //listen to queue events
18:         mSender.OnMessage((m) =>
19:             {
20:                 messageList.Dispatcher.Invoke(() =>
21:                     {
22:                         messageList.Items.Add(string.Format("{0}: {1}",
23:                             m.SequenceNumber, m.GetBody<string>()));
24:                     });
25:                 }, new OnMessageOptions { MaxConcurrentCalls = 1 });
26:             }
27:
28:         const string connString = "<Service Bus connection string>";
29:         const string queueName = "<queue name>";
30:         QueueClient mSender, mReceiver;
31:
32:         private void sendMessage_Click_1(object sender,
33:             RoutedEventArgs e)
34:         {
35:             mSender.Send(new BrokeredMessage(messageText.Text));
36:             messageText.Text = "";
37:         }

```

AMQP 1.0 in 2012. We will provide a brief overview in the next section. Interested readers who want to know further details about the protocol may visit MSQP's official site at <http://www.amqp.org>.

15.3.1 AMQP Overview

AMQP is defined in several different layers, including a type system, a transportation layer, a messaging layer, a transaction layer, and a security layer.

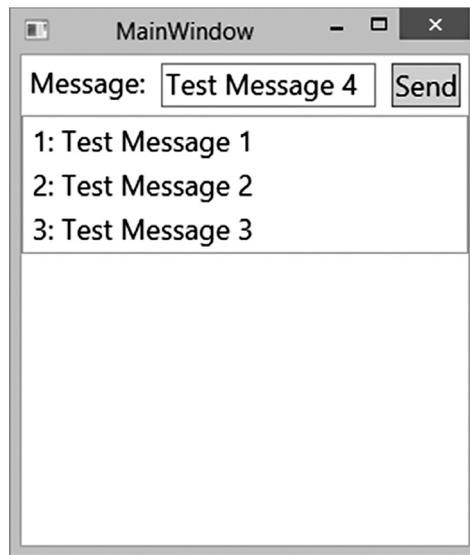


Figure 15.10 Event-driven consumer demo.

- Type system

In order to facilitate exchange of data from different technology platforms, AMQP defines a unified type system. The type system defines common basic types such as Booleans, byte, long, and strings. In addition, AMQP allows you to map custom types to basic types by attaching descriptors on types. AMQP message data are encoded as byte series, prefixed with constructors.

- Transportation layer

The AMQP link layer is made up of nodes and links. Nodes are entities that send/revive or store messages, including senders, receivers, and message queues. Links are data channels among nodes. Links are connected to nodes by source terminuses and target terminuses. Nodes are hosted in containers. A container can host multiple nodes. For example, a client program can be viewed as a container, which includes several sender nodes and receiver nodes.

AMQP transportation layer defines point-to-point connections between nodes. AMQP containers exchange data via connections, which are comprised of duplex, ordered frames. An AMQP channel can be segmented into multiple unidirectional channels. Every frame on a channel is tagged with a channel number and a sequence number. The combination of the channel number and the sequence number uniquely identifies a frame. AMQP requires that when a frame with sequence n arrives, all frames with sequence numbers less than n must have already arrived.

- Messaging layer

The messaging layer defines message formats, message transmission statuses, node statuses, as well as the description of message sources and message destinations. Messages are immutable in AMQP. However, information can be attached to messages during transmissions. Messages contain headers, which define transmission requirements such as priority,

persistence, expiration, and the number of deliveries. In addition, AMQP messages support a series of built-in properties and application-defined properties (which are limited to basic-type properties identified by string keys). Message transmission intermediaries can leverage these properties to filter or route messages. The message layer also defines message transmission statuses, such as received, rejected, released (message is not and will not be processed), modified (message is modified but unprocessed), or delivered.

- Transaction layer

The transaction layer combines multiple message transmissions in transactions. AMQP supports typical transactional operations such as commits and rollbacks. Note that AMQP 1.0 does not specify how distributed transactions are supported. In this version of the standard, all operations with a transaction are carried out by a single transactional resource.

- Security layer

The security layer is used to define security measures such as authentication, data encryption, and support for TLS and SASL.

15.3.2 AMQP Adoption

AMQP has gained wide adoption in the industry. An increasing number of systems have added support for AMQP, including Apache Qpid, Microsoft Azure Service Bus, VMWare RabbitMQ, Storm MQ, JORAM, Red Hat enterprise MRG, and Fedora AMQPI. Among these systems, Apache Qpid is a cross-platform AMQP system, which provides C++ and Java-based message brokers as well as APIs for various programming languages such as C++, Java JMS, .Net, Python, and Ruby. Microsoft Azure Service Bus has included built-in AMQP support as well.

Now let us learn how to use AMQP with an example.

Example 15.5: System integration using AMQP

Difficulty: ****

This example is constructed using both .Net and Java. On the .Net side, we will use the Microsoft Azure Service Bus Client NuGet package to create a simple console application for receiving orders sent via AMQP. On the Java side, we will use JMS and Apache Qpid to send orders using AMQP.

Note: In this example, we choose to use Google's ADT Bundle for Windows as the Java IDE, and the steps are based on this IDE. Of course, you can pick the standard Eclipse or other Java IDEs.

This example consists of three parts. First, let us prepare the Apache Qpid development environment.

Part 1: Prepare Apache Qpid environment

At the time of writing this book, Apache Qpid is still in RC state. You can go to the address <http://people.apache.org/~rgodfrey/qpid-java-amqp-1-0-client-jms.html> to get the link to download the

latest version. You will need to download the following .jar files. Save the downloaded files to a local folder. In this example, we will use `c:\java\amqp`.

- geronimo-jms_1.1_spec-1.0.jar
- qpidd-amqp-1-0-client-[version].jar
- qpidd-amqp-1-0-client-jms-[version].jar
- qpidd-ampq-1-0-common-[version].jar

In this example, we use the 0.22-RC5 version.

Part 2: Create the Java order sender

1. In Eclipse, create a new Java project named *AMQPOrderSender*. When creating the project, add references to the previous Apache Qpid .jar files (or manually add paths to these .jar files to CLASSPATH), as shown in Figure 15.11.
2. On the same dialog, select the Order and Export tab, and then select all referenced .jar files, as shown in Figure 15.12.

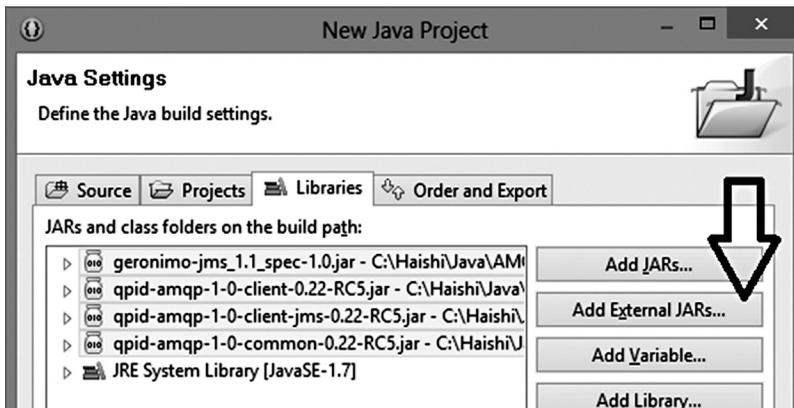


Figure 15.11 Referencing Apache Qpid .jar files.

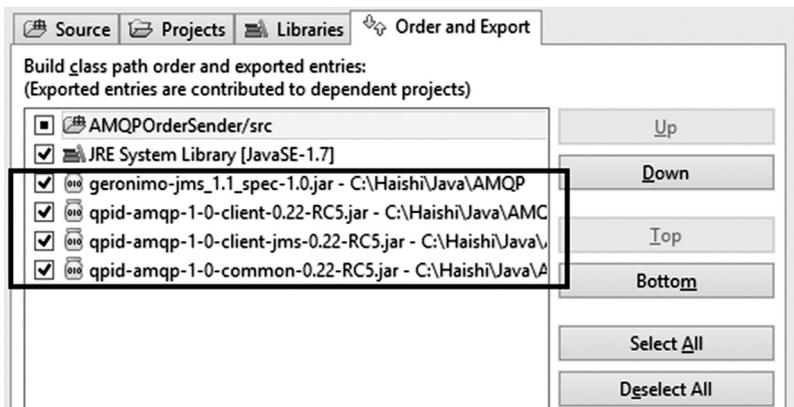


Figure 15.12 Order of class paths.

3. Add a **servicebus.properties** file under the project's root folder. You will need to replace <user> and <secret key> with your Service Bus connection information. Note that the secret key is expected to be HTTPURL encoded. For example, the equal sign (=) needs to be encoded as %3D.

```
connectionfactory.SBCF = amqps://<user>:<secret key>@<Service Bus namespace>.servicebus.windows.net
queue.QUEUE = <Service Bus queue>
```

4. Add a new **AMQPOrderSender** class under the **src** folder. Although the following code is a bit long, it is not hard to understand. You may refer to the comments in Code List 15.6 for further details.

Part 3: Create the .Net order receiver

1. Create a new Windows console application.
2. Add a reference to the **WindowsAzure.ServiceBus** NuGet package and a reference to the **System.Configuration** assembly.
3. Modify the **Microsoft.ServiceBus.ConnectionString** setting in **App.config** to enter your Service Bus namespace connection string. We need to postfix “*;TransportType = Amqp*” to the connection string to indicate that we will use AMQP in this case.
4. Modify the **Program.cs** file. The core of Code List 15.7 is the **AMQPOrderReceiver** class. The constructor of these calls creates a listener thread to listen for new order messages. As you can see, other than the slight differences during initialization, the code looks the same as “regular” Service Bus code when the proprietary protocol is used.

Part 4: System test

1. Launch the .Net receiver.
2. Launch the Java sender.
3. On Java output view, enter the name of the item to be purchased (such as Socks). The system generates a random price (the screenshot shows that the system tries to rip you off by charging \$65 for a pair of socks), and asks how many you want to buy. Enter a quantity, and press the Enter key to send the order. Once the order is sent, you can observe the \$80,000 order for socks on the .Net window, as shown in Figure 15.13.

15.4 Advantages of Message-Based Integration

Before concluding this chapter, let us summarize the advantages of message-based integration.

15.4.1 Loose Coupling

Loose coupling is one of the main characteristics of message-based integration. The middleware eliminates any direct links among integrated systems, allowing the systems using different platforms, protocols, and formats to communicate with each other without knowing any details of other systems. In the previous example, we achieved communications between a Java program and

CODE LIST 15.6 AMQP JAVA SENDER

```

1: import java.io.BufferedReader;
2: import java.io.InputStreamReader;
3: import java.util.Random;
4: import java.util.Hashtable;
5: import javax.jms.*;
6: import javax.naming.Context;
7: import javax.naming.InitialContext;
8:
9: publicclass AMQPOrderSender {
10:
11:     private Session session;
12:     private Connection connection;
13:     private MessageProducer sender;
14:
15:     public AMQPOrderSender() throws Exception{
16:         //configure JNDI environment
17:         Hashtable<String,String> env = new Hashtable<String,
18:             String>();
19:         env.put(Context.INITIAL_CONTEXT_FACTORY,
20:             "org.apache.qpid.amqp_1_0.jms.jndi.
21:             PropertiesFileInitialContextFactory");//Note:
22:             18-19 is one line
23:         env.put(Context.PROVIDER_URL, "servicebus.properties");
24:         Context context = new InitialContext(env);
25:         ConnectionFactory factory =
26:             (ConnectionFactory)context.lookup("SBCF");
27:         Destination queue = (Destination)context.
28:             lookup("QUEUE");
29:         //create connection
30:         connection = factory.createConnection();
31:         session = connection.createSession(false,
32:             Session.AUTO_ACKNOWLEDGE);
33:         sender = session.createProducer(queue);//create sender
34:             node
35:     }
36:     privatevoid close() throws JMSEException{
37:         connection.close();
38:     }
39:     privatevoid sendMessage(String item, double price, int units)
40:         throws JMSEException{
41:         double total = price * units;
42:         TextMessage message = session.createTextMessage();
43:         message.setText(item);//set message content
44:         message.setDoubleProperty("Price", price);//add message
45:             properties
46:         message.setIntProperty("Units", units);

```

```

42:         message.setDoubleProperty("Total", total);
43:         sender.send(message);
44:         System.out.println("Sent order!");
45:         System.out.println("=====");
46:         System.out.println("Order for " + item);
47:         System.out.println("-----");
48:         System.out.format("Price   : $%,10.2f%n", price);
49:         System.out.format("Quantity: %11d%n", units);
50:         System.out.format("Total   : $%,10.2f%n", total);
51:         System.out.println("=====");
52:     }
53:     publicstaticvoid main(String[] args) {
54:         try
55:         {
56:             AMQPOrderSender sender = new AMQPOrderSender();
57:
58:             BufferedReader cmdLine = new java.io.
                BufferedReader
59:                 (new InputStreamReader(System.in));
60:             while(true){
61:                 System.out.print("Enter Item Name: ");
62:                 String item = cmdLine.readLine();
63:                 if (item.equalsIgnoreCase("exit")){
64:                     sender.close();
65:                     System.exit(0);
66:                 }
67:                 //generate random price, then ask user for
                    quantity
68:                 Random rand = new Random();
69:                 double price = Math.ceil(rand.
                    nextDouble()*10000)/100;
70:                 System.out.format("Item %s is sold at
                    $%.2f in our store.
71:                     How many do you want to buy? ",
                        item, price);
72:                 String userUnits = cmdLine.readLine();
73:                 int units = 0;
74:                 try
75:                 {
76:                     units = Integer.
                        parseInt(userUnits);
77:                 } catch (NumberFormatException exp){
78:                     System.out.println("Invalid number
                        format.
79:                         Units have to be integers.");
80:                     continue;
81:                 }
82:                 //Send message
83:                 sender.sendMessage(item, price, units);

```

```
84:         }
85:
86:     } catch (Exception e){
87:         e.printStackTrace();
88:     }
89: }
90: }
```

a .Net program. If we switched any party in a Python program, it will not impact the other party, and this switch can occur while the other program is still in operation.

15.4.2 *Dynamic Extension*

Messaging middleware creates a common communication platform for participating systems. The communication pattern is similar to what is in an online chat room, where people can come and go, but the conversation goes on. With message-based integration, new systems can be joined, and old systems can be removed at any time without affecting other participating systems. For a new system to be joined, it only needs to be linked to the middleware in order to communicate with any of the systems. This is obviously much more convenient than creating connections to many systems. Dynamic extension also allows monitors and auditors to be dynamically attached and removed as needed.

15.4.3 *Asynchronous Communication*

With message-based integration, systems communicate with each other asynchronously. Message senders and receivers can work at their own pace without having to keep up with each other. The messaging middleware serves as a buffer between different systems so that when their throughputs do not match up, they can still work with each other. The extreme form of buffering is the batch mode. Message senders and message receivers do not need to be online at the same time, but they can still exchange data via the buffer.

By leveraging both dynamic extension and asynchronous communication, system administrators can dynamically adjust system capacity based on the actual system workload. For example, when an administrator observes that a job queue is becoming too long, he or she can launch additional job processors to drain the queue. On the contrary, he can turn off excessive processors to save cost while the system is not busy.

15.4.4 *Centralized Management*

While all systems communicate over a centralized middleware, system administrators can gain insight into the whole system through the central point. For example, as a message flows from system to system, an administrator can track its trail through the message system without needing to collect information of each of the participating systems.


```

36:             ("Quantity: {0,11}",
37:             (int)order.
                Properties["Units"]));
38:             Console.WriteLine(string.Format
39:             ("Total    :
                ${0,10:#,###.00}",
40:             (double)order.
                Properties["Total"]));
41:             Console.
                WriteLine("=====");
42:             order.Complete();
43:         }
44:     else
45:         Console.Write(".");
46:     }
47:     }));
48:     mListen = true;
49:     mListenerThread.Start();
50: }
51: public void Close()
52: {
53:     mListen = false;
54:     mListenerThread.Join();
55: }
56: }
57: class Program
58: {
59:     static void Main(string[] args)
60:     {
61:         string connString = ConfigurationManager.AppSettings
62:             ["Microsoft.ServiceBus.ConnectionString"];
63:         string entityName = ConfigurationManager.
                AppSettings["EntityName"];
64:         AMQPOrderReceiver receiver = new AMQPOrderReceiver
65:             (connString, entityName);
66:         Console.WriteLine("Waiting for orders");
67:         Console.ReadLine();
68:     }
69: }
70: }

```

Message-based integration is the result of collective integration experiences, and it has been showing its power in various industries and applications. Microsoft has just launched BizTalk Service on Microsoft Azure. BizTalk Service brings Microsoft's BizTalk Server to cloud and provides all essential characteristics of a robust middleware, including availability, reliability, and scalability. It is foreseeable that BizTalk Service will become a powerful tool for system integration in the future.

```

AMQPOrderSender [Java Application] C:\Haishi\Java\jre7\bin\javaw.exe (Jun 9, 2013 5:08:29 PM)
Enter Item Name: Socks
Item Socks is sold at $64.98 in our store. How many do you want to buy? 1200
Sent order!
=====
Order for Socks
-----
Price   : $      64.98
Quantity:      1200
Total   : $ 77,976.00
=====
Enter Item Name:

file:///C:/Haishi/DragonLair/AMQPOrder/Net/AMQPOrderRe
Waiting for orders
.....Received order!
=====
Order for Socks
-----
Price   : $      64.98
Quantity:      1200
Total   : $ 77,976.00
=====
.....

```

Figure 15.13 AMQP-based ordering system.

15.5 Summary

In this chapter, we introduced basic concepts, patterns, and techniques in system integration. Including the Competing Consumer pattern that we introduced in Chapter 9, we have covered six typical patterns in system integration: Competing Consumer, Content-based Routing, Priority Queue, Request/Response, Dead Letter Queue, and event-driven consumer. We also introduced AMQP with an example that integrates a Java application and a .Net application. Finally, we summarized characteristics of message-based integration.

Chapter 16

Source Control and Tests with Visual Studio Online

As the software industry evolves, the complexity of software and services keeps increasing, far beyond the capability of a single developer. For most projects, the efficiency of teamwork directly decides the outcome of the projects. A cloud service development team needs not only integrated development tools such as Microsoft Visual Studio but also effective project management tools and testing tools. Furthermore, it is highly desirable that these tools are integrated with each other so that the team can seamlessly manage the whole product lifecycle, from product management to design, to development, to test and release management.

In November 2013, Microsoft announced the launch of Visual Studio 2013, .NET 4.5.1, and the new Visual Studio Online. Visual Studio Online consists of a rich collection of developer services, including the following:

- Source control and project management services. These services were part of the Team Foundation Service and are now integrated as part of Visual Studio Online.
- Visual Studio Online Application Insights service. This service provides a complete view of your applications' health, based on their availability, performance, and usage data.
- Cloud-based coding environment. This is an online editor that completes a “pure” online project development, management, and operation experience for certain Microsoft Azure development scenarios.

Visual Studio Online provides many exciting features to be explored. In this chapter, we will focus on the Microsoft Team Foundation Service equivalences and demonstrate how you can leverage these services to manage your project and source code as well as conducting tests in a team environment.

16.1 Create a Visual Studio Online Account

Visual Studio Online, formerly Team Foundation Service (TFS), provides a comprehensive solution for managing the whole software lifecycle, from project plan and management to source control and continuous integration. You can do everything online if you choose to, or you can leverage the deep integration with Visual Studio to collect your developer desktops into an organic team environment. At the time of writing this book, Visual Studio Online provides a free five-user plan. If you are already an MSDN subscriber, you can join projects at no additional charge. Or, you can visit the following address to create a new account:

<http://www.visualstudio.com>.

Look for a “Get Started for free” link, or pick the subscription levels you want to use to get started. The creation process is very easy. After you log in using a Microsoft Account, you will see an account creation page that looks similar to Figure 16.1.

Account creation only takes a few seconds, and then you are ready to create your first project on cloud (see Figure 16.2)—saying agility of cloud!

16.2 Source Control with Visual Studio Online

In this section, we learn the general process of source control using Visual Studio Online. An efficient source control system is essential for a productive development team. Although different teams may have different preferences and different ways of managing source code, some general good practices can be abstracted:

- Isolated check-ins.

Developers should make best attempts to isolate each check-in to a single specific bug fix or a single feature implementation. This practice ensures that when a check-in needs to

Create a Visual Studio Online Account

Full name *

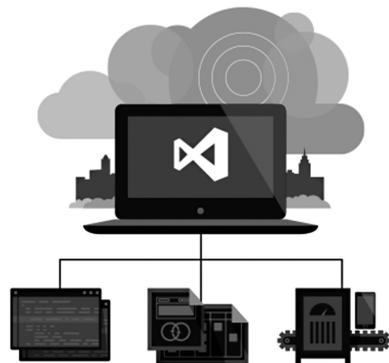
Contact e-mail *

Country/Region *

Account URL * 

Occasionally send me emails with news, training opportunities and other offers from Visual Studio. I can unsubscribe at any time.

Create Account



By clicking **Create Account**, you agree to the [Terms of Service](#) and [Privacy Statement](#).

Figure 16.1 Create a free Visual Studio Online account.

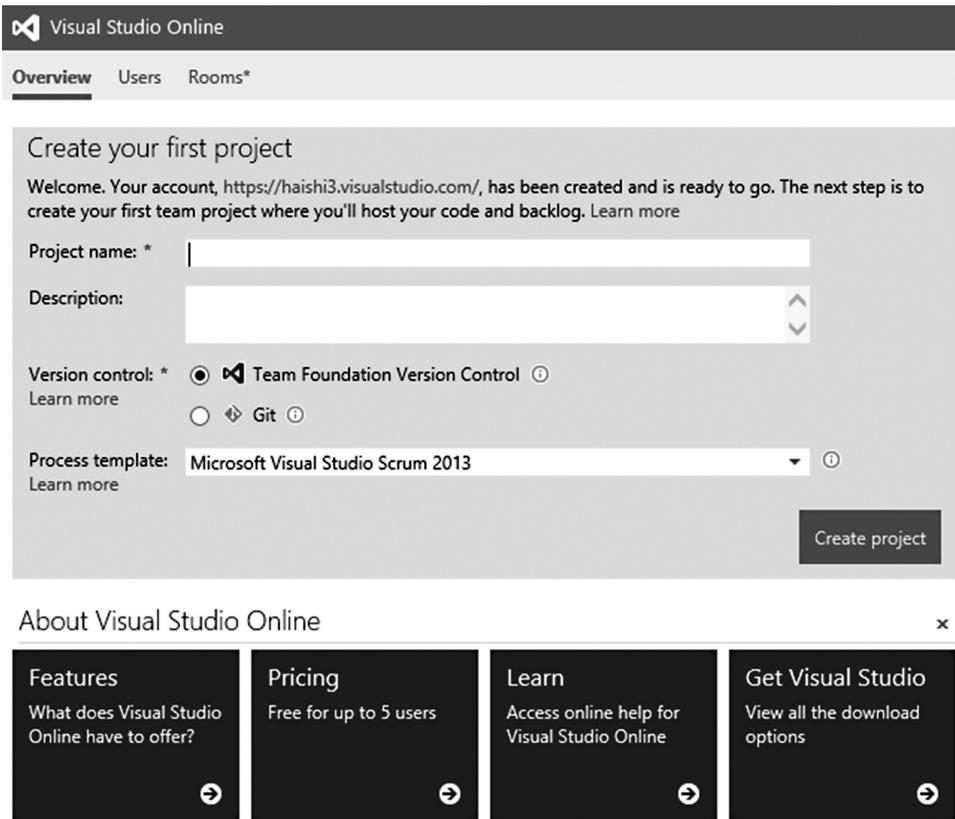


Figure 16.2 Created Visual Studio Online account.

be rejected or rolled back, it will not affect other bug fixes or features. In addition, when a big fix is associated with a work item, the corresponding work item should be marked accordingly.

- Retain a separate branch/tag for every release.
When a version is released, the corresponding source should be explicitly branched or tagged. This will be the baseline of hotfixes and patches for the specific version. The fixes should be selectively merged into the main branch after review.
- Use the source control system to keep modification history.
Before source control systems were widely adopted, programmers used to keep modification histories by commenting out the old code. Indeed, this was once considered a good practice. Such manual recording of history is no longer necessary with a modern source control system, which gives you not only the history, but also the capabilities to compare and merge changes. Many developers, especially those who have been coding for a long time, need to break away from the old custom and leverage the source control system to retain history records.

Next, let us go through the general steps of source control with an example.

Example 16.1: Source Control with Visual Studio Online

Difficulty: **

1. Sign in to the Visual Studio Online site. If you have not created a project yet, you will see a page as shown in Figure 16.2, where you can directly create a new project. To create more projects after the first, look for the **Recent projects and teams** section, and click the **New** link, as shown in Figure 16.3.
2. On the **CREATE NEW TEAM PROJECT** dialog, enter project **Name** and **Description**. Pick a **Process template** you want to use and a **Version control** engine (Team Foundation Version Control or Git). Then, click the **Create Project** button, as shown in Figure 16.4.
3. After the project has been created, click the **Navigate to Project** button.
4. Then, you can click the **Open new instance of Visual Studio** link to launch Visual Studio (as shown in Figure 16.5). However, if you plan to use the Azure Compute Emulator, you will need to launch Visual Studio as an administrator. So we will not use the link here.

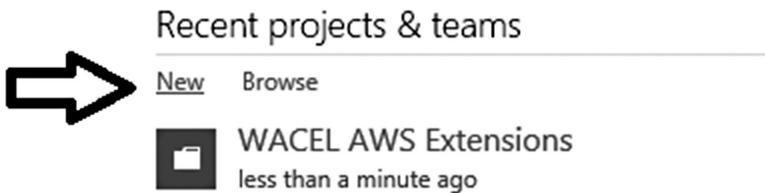


Figure 16.3 Create a new TFS project.

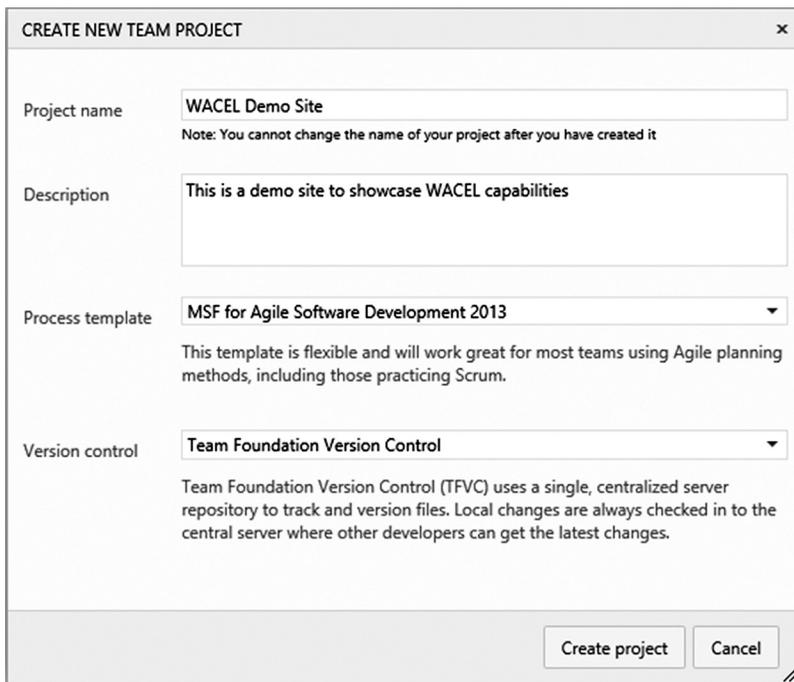


Figure 16.4 New team project dialog.

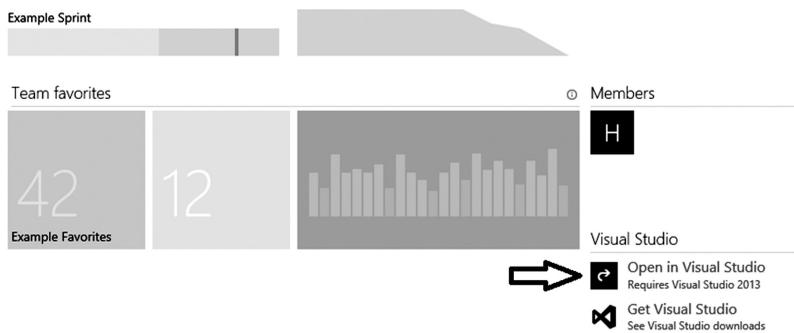


Figure 16.5 Link to launch Visual Studio.

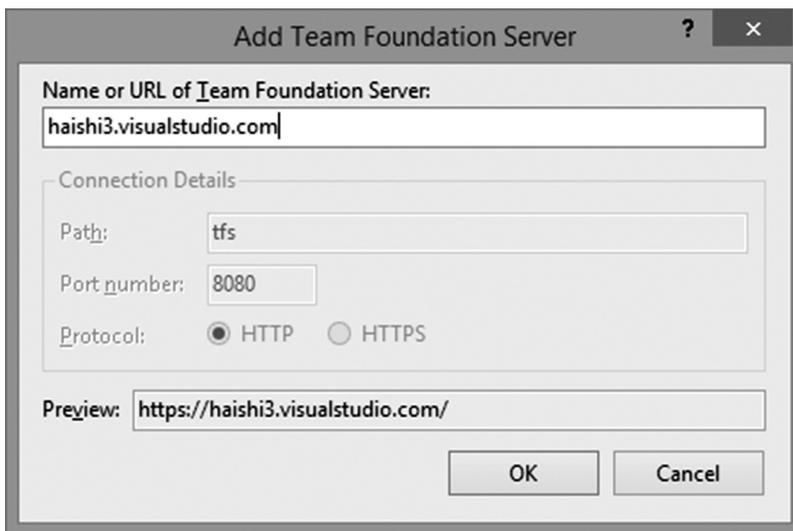


Figure 16.6 Add a TFS server.

5. Launch Visual Studio 2013 (Ultimate version) as an administrator. Then, select the **TEAM**→**Connect to Team Foundation Server** menu.
6. On the **Connect to Team Foundation Server** dialog, click on the **Servers** button (if you do not see the dialog after selecting the menu, look for a **Select Team Projects** link on **Team Explorer**).
7. On the **Add/Remove Team Foundation Server** dialog, click the **Add** button.
8. On the dialog, enter the URL of your Visual Studio Online subscription, and then click the **OK** button, as shown in Figure 16.6.
9. The system will prompt you to sign in using your Microsoft account. After logging in, go back to the **Add/Remove Team Foundation Server** dialog, and click the **Close** button.
10. Back on the **Connect to Team Foundation Server** dialog, select your project, and then click the **Connect** button, as shown in Figure 16.7.
11. After the connection has been established, the **Team Explorer** will open, allowing access to various Visual Studio Online functionalities, as shown in Figure 16.8.

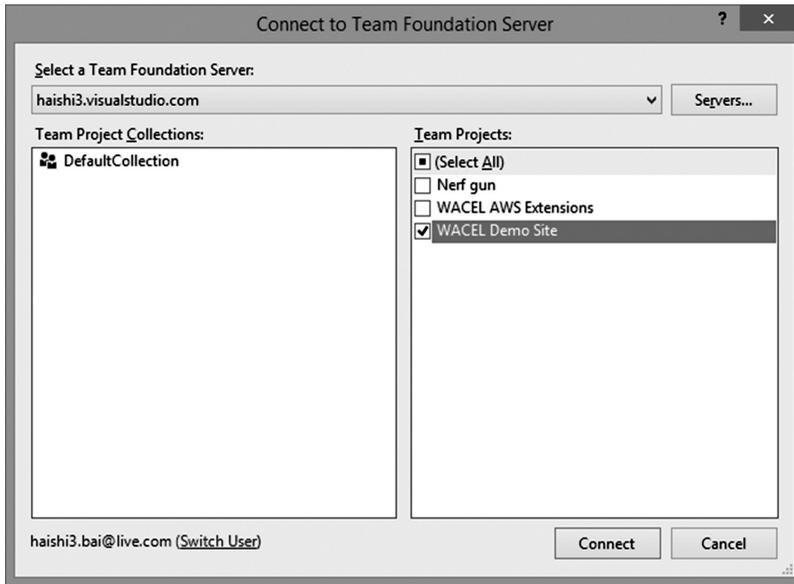


Figure 16.7 Connect to a TFS project.

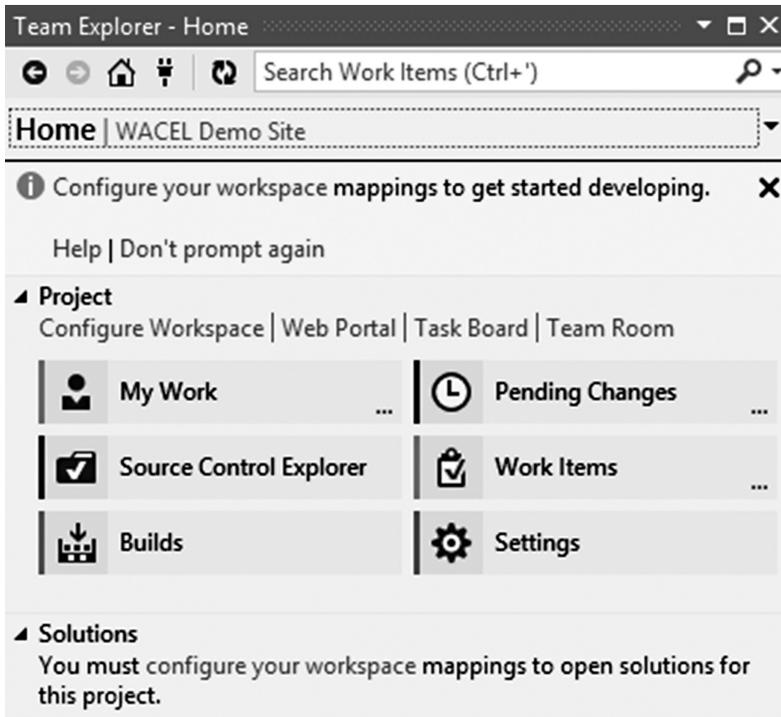


Figure 16.8 Team Explorer.

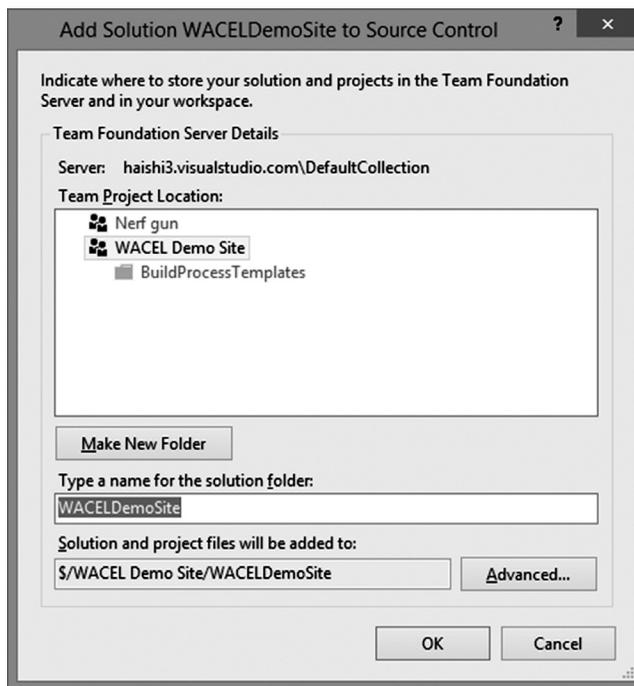


Figure 16.9 Add project to source control.

12. In Visual Studio, create a new cloud service with a single ASP.NET Web Role (using the MVC template).
13. In **Solution Explorer**, right-click the solution and select the **Add Solution to Source Control** menu.
14. On the **Choose Source Control** dialog, select **Team Foundation Version Control** or **Git**. Here, we will choose **Team Foundation Version Control**.
15. On the **Add Solution [your solution] to Source Control** dialog, click the **OK** button to continue (see Figure 16.9). You can optionally enter a different solution folder on this dialog.
16. When you are ready to effect changes, right-click the solution and select the **Check In** menu.
17. In **Team Explorer**, enter a comment for your check-in, and then click the **Check In** button, as shown in Figure 16.10.
18. On **Check-in Confirmation** dialog, click **Yes** to continue.
19. Once the check-in is completed, go to the **Team Explorer**. Click on the **Home** button, and then click on the **Source Control Explorer** link to open the **Source Control Explorer**, as shown in Figure 16.11.
20. As your code has been preserved on TFS, you can use the Source Control Explorer to check-out/check-in the code, to create and merge branches, or to view histories. For example, in Figure 16.12, my edit of *HomeController* is automatically picked up by the Source Control Explorer.
21. Right-click the modified file, and select the **Compare** menu.
22. On the **Compare** dialog, click the **OK** button to continue. You can see that the differences between the versions are highlighted, and you can navigate through all the changes by using the scroll bar to the right of the window, as shown in Figure 16.13.

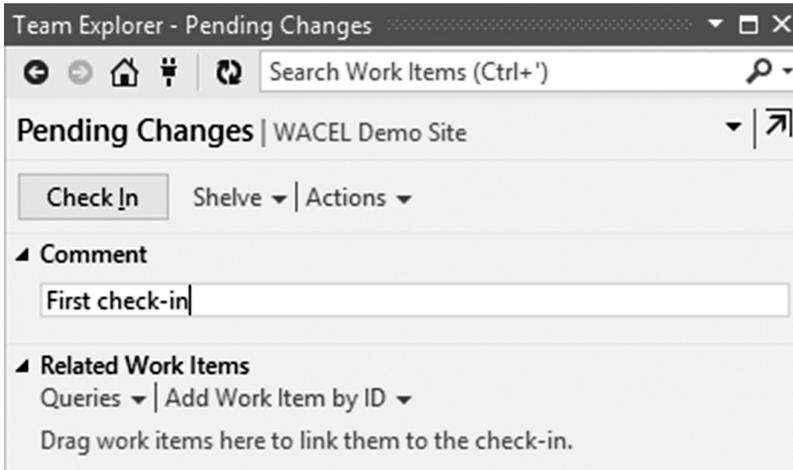


Figure 16.10 Check in source code.

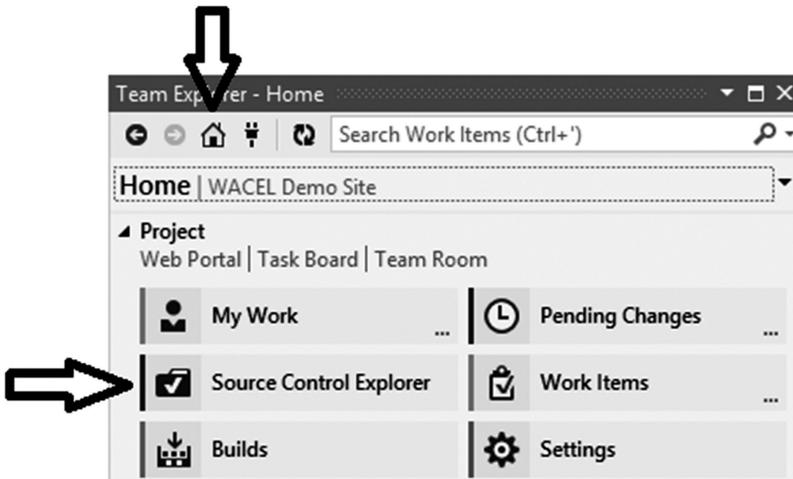


Figure 16.11 Open Source Control Explorer.

16.3 Create and Use Unit Tests

What is the purpose of writing unit tests? Many developers believe that writing unit tests is ensuring code quality. However, to me this idea is a paradox—if a developer cannot write high-quality code, how can we expect him or her to write high-quality unit tests to improve upon the code? I believe the real value of unit tests resides in two areas: for use as a design tool, and for use as an effective tool to capture and reinforce design decisions.

- Unit tests as a design tool

Unit tests are a very good tool for designing new functions and APIs. By writing the unit test code first, a developer can examine the API design from a consumer’s perspective. This helps

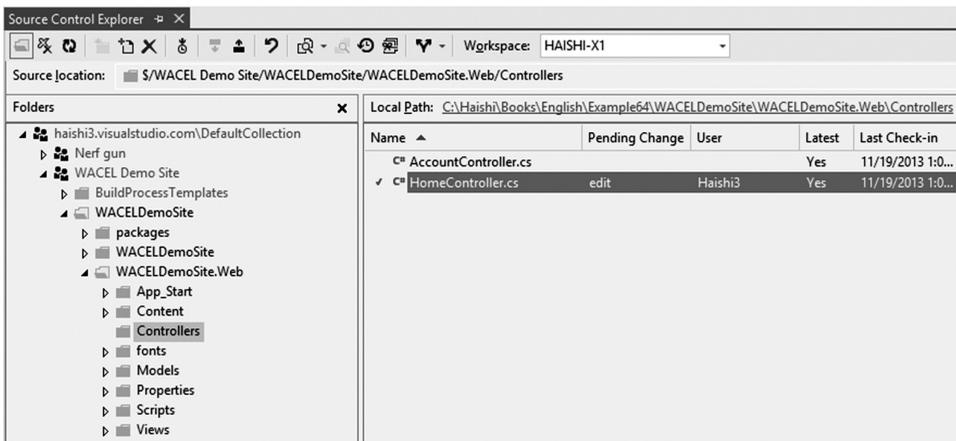


Figure 16.12 Source Control Explorer.

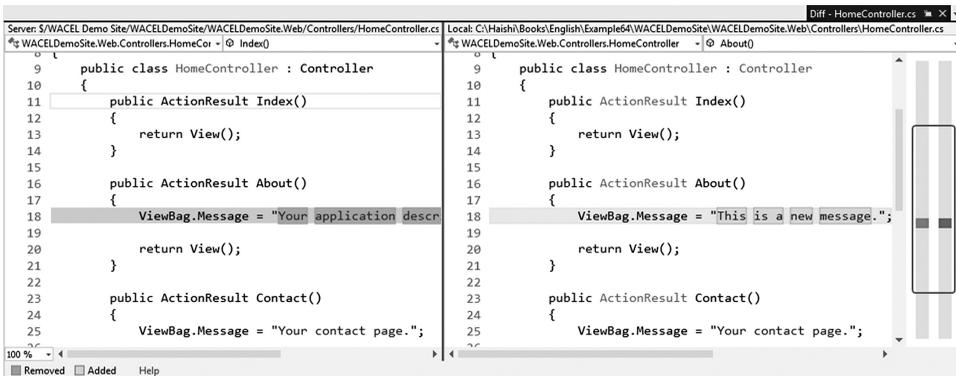


Figure 16.13 Compare source code versions.

the developer design clearer, more concise, and easy-to-use interfaces. This is the essence of the Test-Driven Development (TDD) process, which has proven to be very effective, especially in framework developments.

- Unit tests as a design document

Unit tests capture the original intentions of system architects. Design documents in text are lifeless and are easily forgotten. However, unit test projects are alive and can proactively inform developers if they were deviating from the original design. Such checks can be automated by integrating unit tests as part of the continuous integration process. The ability to capture and reinforce system design in this way is extremely important. Nowadays, members of development teams constantly change, and often there is not enough time to ramp up new members as they join. Without these unit tests, the original design is eventually lost, and the system architecture becomes murky over time. In other words, unit tests are the best architecture documents you can rely on.

Next, we will learn how to create and use unit tests through two examples. We will first create a unit test case, and then set it to automatic run during each build. Next, we will set up gated check-in, which allows new code to be checked in only if all unit test cases have passed.

Example 16.2: Create and Use Unit Tests

Difficulty: ***

1. Continue with Example 16.1. Right-click the solution, and select the **Add→New Project** menu.
2. Choose the **Test→Unit Test** template, and click the **OK** button.
3. In the new unit test project, add a reference to the Web Role project. In addition, add a reference to **System.Web.Mvc 4.0.0.0** (from the Extensions group) and a reference to **Microsoft.CSharp 4.0.0.0** (from the Frameworks group). If you are using MVC 5, you will need to add a reference to **System.Web.Mvc 5.0.0.0** instead.
4. Modify the test method. The modified code is shown in Code List 16.1.
5. Ensure the HomeController in the Web Role has been modified:

```
public ActionResult About ()
{
    ViewBag.Message = "This is a new message.";
    return View();
}
```

6. Select the **TEST→Windows→Test Explorer** menu.
7. In the **Test Explorer**, click the **Run All** link. Because we only have one test case, the test will finish quickly, as shown in Figure 16.14.
8. Now let us set up the unit tests to automatically run when the solution is built. In Visual Studio, select the **VIEW→Team Explorer** menu. On the project's home page, click on the **Builds** link to switch to the **Builds** view.
9. Click in the **New Build Definition** link.
10. On the build definition window, switch to the **Process** tab. Then, in the table to the right, expand the row that says "Automated Tests." Change **Fail Build on Test Failure** to **True**. This option will make the build fail unless all unit tests have passed. In addition, change **Target platform for text execution** to **X64**, as shown in Figure 16.15.
11. Press Ctrl + S to save the changes.

CODE LIST 16.1 A UNIT TEST METHOD

```
using Example16.1.Controllers;
using System.Web.Mvc;
...
[TestMethod]
public void TestMethod1 ()
{
    HomeController controller = new HomeController();
    ViewResult result = controller.About() as ViewResult;
    Assert.AreEqual("This is a new message.", result.ViewBag.
        Message);
}
```

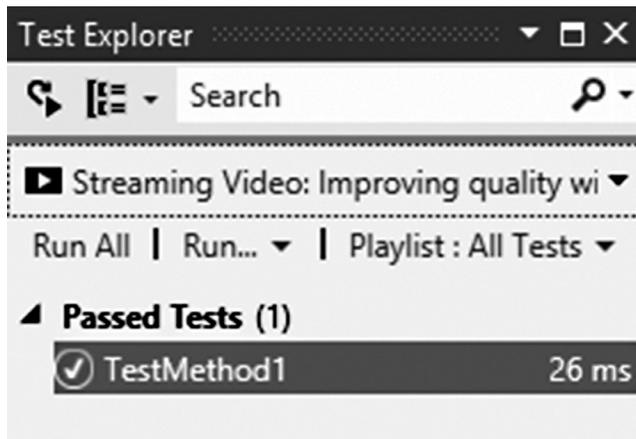


Figure 16.14 Unit test result.

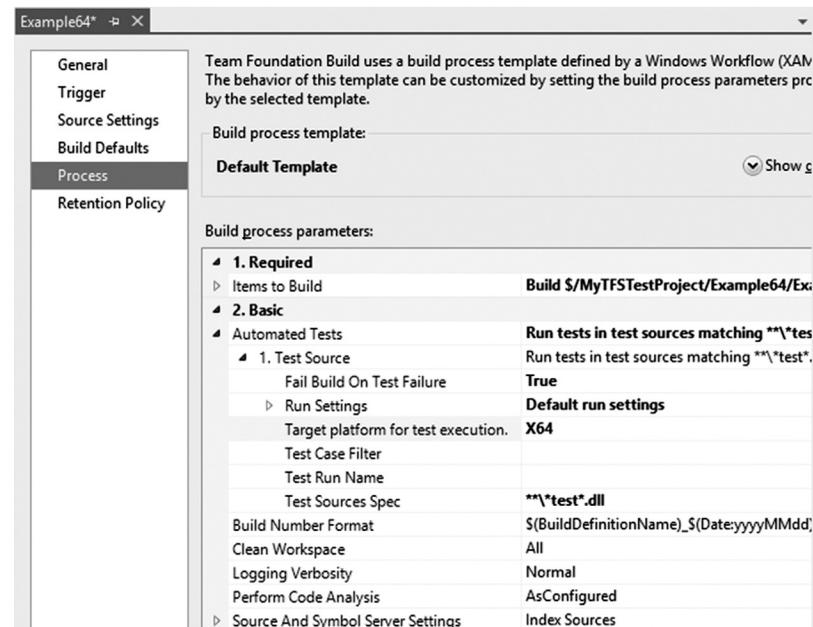


Figure 16.15 Creating a build definition.

12. Back in **Team Explorer**, right-click the newly created build definition, and select the **Queue New Build** menu. On the pop-up dialog, click the **Queue** button to queue a new build. Back in **Team Explorer**, right-click the build definition again and select the **View Builds** menu. Depending on how fast the build process is, your queued build will be either on the **Queued** view or on the **Completed** view. Double-click on the build to view its details, as shown in Figure 16.16. You will observe that our test case has been executed during the build process. Because the test passes, the build is successful.
13. [Optional] If you wish to run the test cases during local builds as well, check the **TEST**→**Test Settings**→**Run Tests After Build** menu.

✓ **Example64_20131013.6 - Build succeeded**
 View Summary | [View Log](#) - Open Drop Folder | Diagnostics ▾ | <No Quality Assigned> ▾ | Actions ▾

Haishi2 Bai triggered Example64 (MyTFSTestProject) for changeset 36
 Ran for 88 seconds (Hosted Build Controller), completed 84 seconds ago

Latest Activity

Build last modified by Elastic Build (haishi2) 84 seconds ago.

Request Summary

Request 12, requested by Haishi2 Bai 3.7 minutes ago, Completed

Summary

Debug | Any CPU

- 0 error(s), 0 warning(s)
- \$/MyTFSTestProject/Example64/Example64.sln compiled
- ▲ 1 test run completed - 100% pass rate
 - buildguest@BUILD-0002 2013-10-13 05:35:20_Any CPU_Debug, 1 of 1 test(s) passed
- No Code Coverage Results

Impacted Tests

No tests were impacted

Figure 16.16 Build result.

Example 16.3: Gated Check-In

Difficulty: **

1. Continue with the previous example. Open the build definition you created in Example 16.2. Switch to the **Trigger** tab, and select the **Gated Check-in** option. Once this option is enabled, only check-ins that are successfully built can be checked in. Because in Example 16.2, we have set up to fail the build if unit tests fail, we are essentially saying that a new code may be allowed to be checked in only when all unit test cases pass (Figure 16.17).
2. Press Ctrl + S to save the changes.
3. Modify the message in the *HomeController* again to make our test case fail:

```

public ActionResult About ()
{
    ViewBag.Message = "This is a WRONG message.";
    return View();
}

```

4. Save and rebuild the solution.
5. In **Solution Explorer**, right-click the solution and select the **Check-in** menu.
6. In **Team Explorer**, enter a comment for the check-in, and click the **Check-in** button.

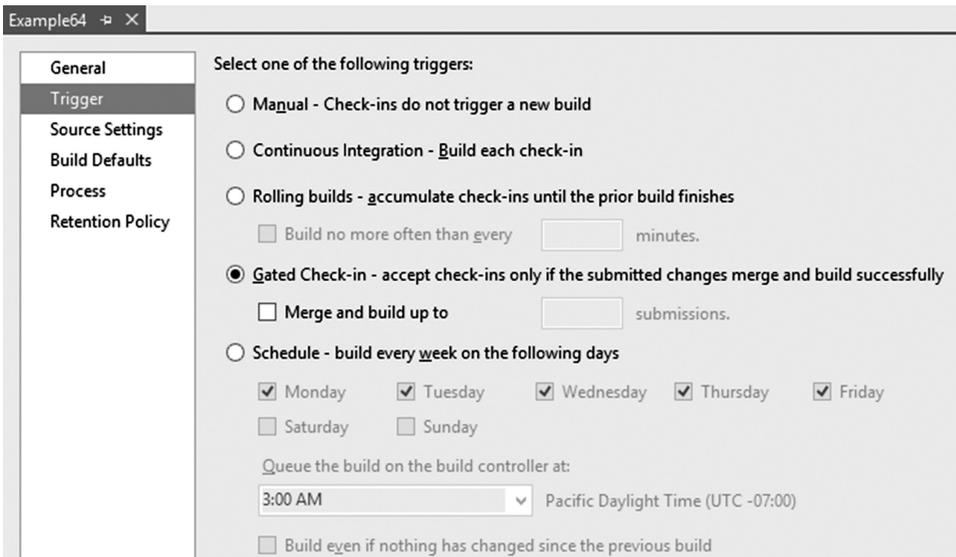


Figure 16.17 Set up gated check-in.

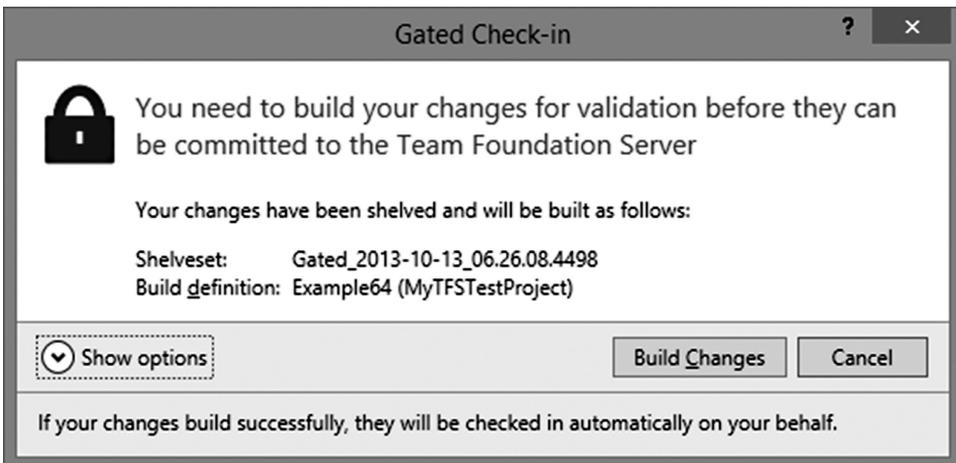


Figure 16.18 Gated check-in.

7. Because we have set up gated check-in, our code has to be verified (by building and running unit tests) before being checked into the source repository. On the Gated Check-in dialog, click **Build Changes** to build the solution, as shown in Figure 16.18.
8. In **Team Explorer**, you can observe the pending check-in. Click on the “here” link (as shown in Figure 16.19) to view the check-in details.
9. Because the test case will fail, the check-in request will be rejected. You can switch to the details view to see the exact reason why the check-in is rejected (see Figure 16.20). In order to check in any new code, you will have to make sure the solution builds and all test cases pass.

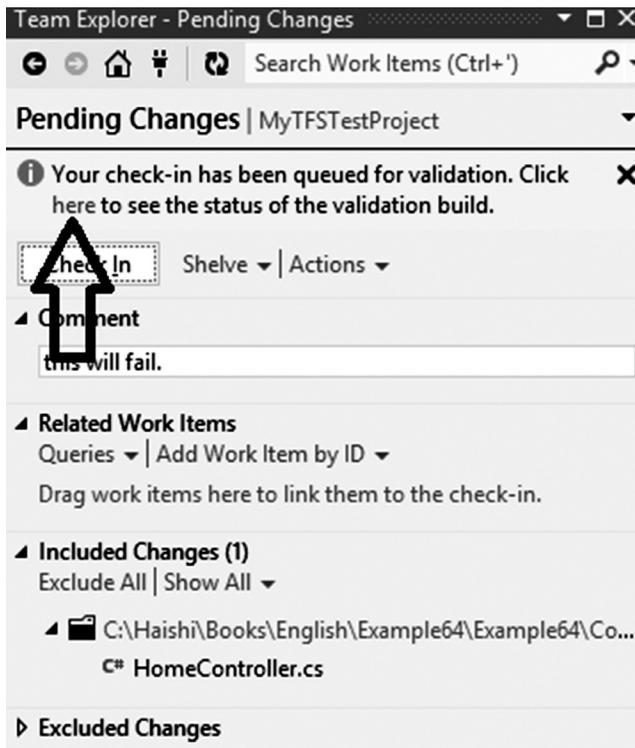


Figure 16.19 Pending changes.

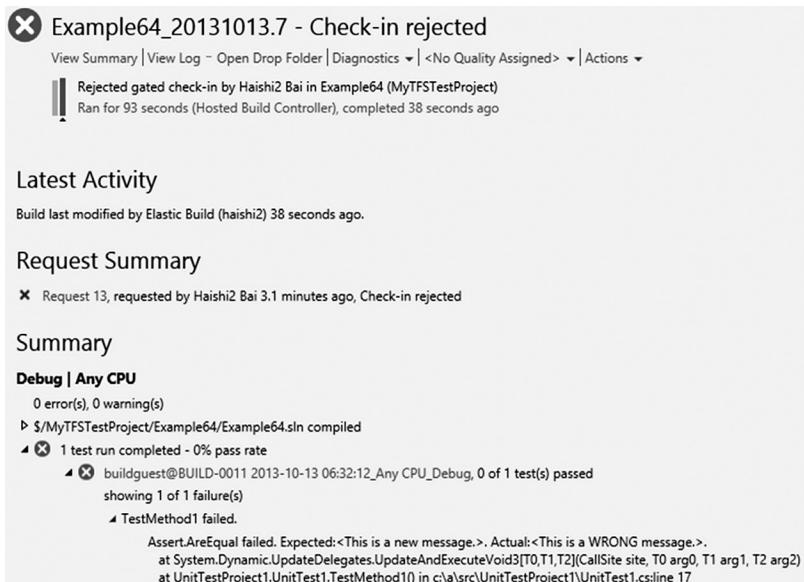


Figure 16.20 Rejected check-in.

16.4 Create and Use Load Tests

Visual Studio Online also allows you to create and run load tests. In the past, in order to run serious load tests, you would need to set up, configure, and manage a large number of test resources such as test controller machines and test agent machines. Visual Studio Online streamlines load testing by freeing you from managing the underlying test infrastructure.

With a Visual Studio Online subscription, you are granted a certain number of virtual user minutes for load tests. During the Commercial Preview period, you get 15,000 virtual user minutes free per month. For example, if your test consists of 250 concurrent virtual users and lasts 60 min, it will consume $250 \times 60 = 15,000$ virtual user minutes.

Example 16.4: Load Tests with Visual Studio Online

Difficulty: **

1. Launch Visual Studio 2013 Ultimate edition.
2. Create a new ASP.NET Web Application (using MVC template). This will be your test target site.
3. Add an empty **Web API 2 Controller** named *FileController*.
4. Add the following line to the **Application_Start** method of **Global.asax.cs**:

```
GlobalConfiguration.Configure(WebApiConfig.Register);
```

5. Add the following method to your **FileController**. This method returns a text file with a given *name* and *size*, after the specified *delay*. This method allows us to simulate variable page sizes and job complexities in our tests.

```
public async Task<HttpResponseMessage>
    Get(string name, int size, int delay)
{
    Thread.Sleep(delay);
    HttpResponseMessage result = new
        HttpResponseMessage(HttpStatusCode.OK);
    result.Content = new StringContent(new string('X', size));
    result.Content.Headers.ContentDisposition =
        new ContentDispositionHeaderValue("attachment");
    result.Content.Headers.ContentDisposition.FileName = name +
        ".txt";
    result.Content.Headers.ContentType =
        new MediaTypeHeaderValue("text/plain");
    return result;
}
```

6. Add a new **Web Performance and Load Test Project** (under *Test* category) to the solution.
7. A **WebTest1.webtest** should be opened in the editor. If the test does not open, double-click on **WebTest1.webtest** in Solution Explorer to open it.
8. The easiest way to create a web test is to record a user session in a browser. You can record different sessions for typical scenarios and then mix them together in your load test. Because the recorded sessions are supposed to be realistic representations of typical user behavior, you should pretend to be a regular user trying to carry out actual business instead of randomly

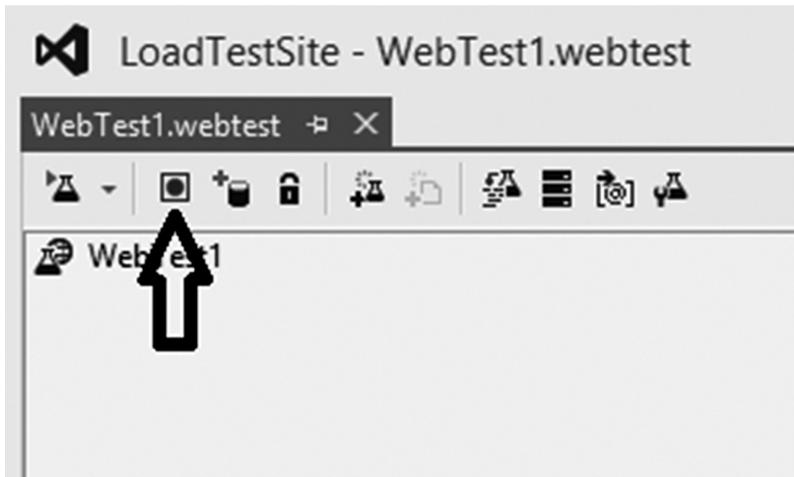


Figure 16.21 Record session button.

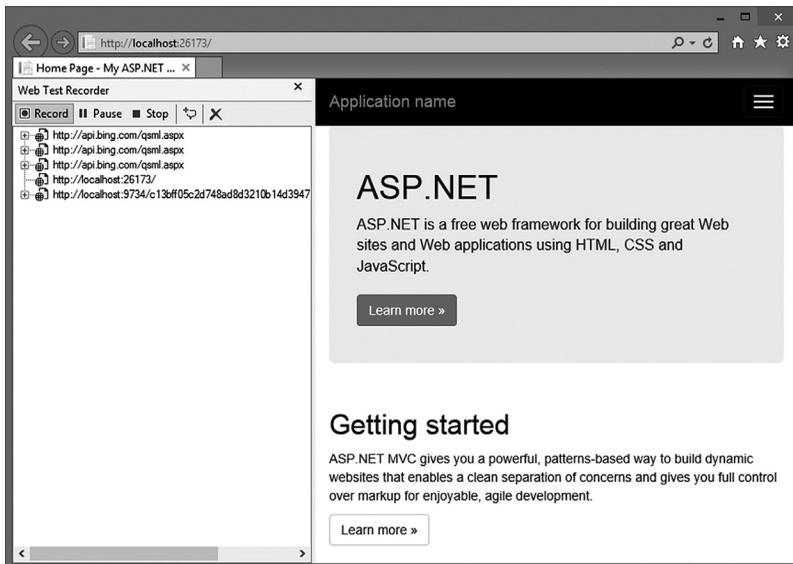


Figure 16.22 Recording a session.

- clicking around. End users think that times (delays between operations) are also recorded, so ensure that you do not introduce unreasonable delays, or transit too fast between operations.
9. Press **Ctrl + F5** to launch the website without a debugger.
 10. Click the record button to launch a browser to record your session, as shown in Figure 16.21.
 11. This operation pops up a browser. Paste in the website URL and start recording your sessions, as shown in Figure 16.22.

In our session, we recorded the following actions. The four queries represent a landing page, a small page, a big page, and a long task, respectively:

```
http://localhost:[port]/
http://localhost:[port]/api/File?name=abc.txt&size=1024&delay=0
```

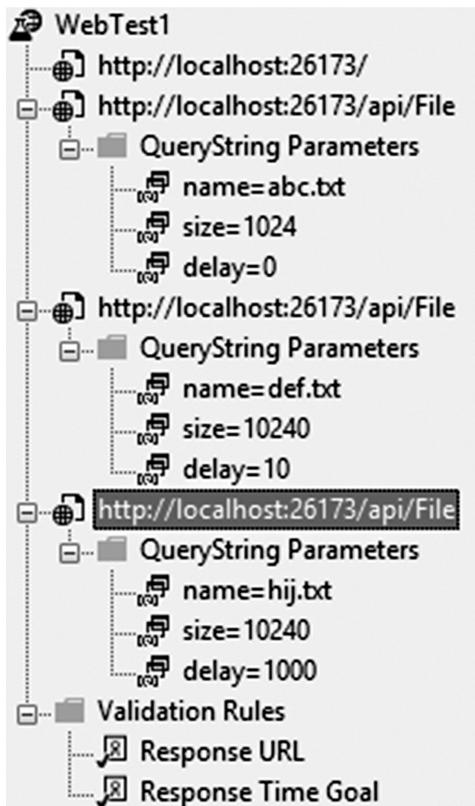


Figure 16.23 Cleaned-up queries.

http://localhost:[port]/api/File?name=def.txt&size=10240&delay=10
http://localhost:[port]/api/File?name=hij.txt&size=10240&delay=1000

12. The recording is very likely to capture some other requests, depending on your browser addons. After stopping the recording, you should remove unwanted entries—simply select them in the editor and press the **Delete** key to delete them. Figure 16.23 shows that all but the entries we want to keep are deleted.
13. Now you can run the test to verify if everything is recorded correctly. Click the **Run Test** button (to the left of the record button) to run the test. If everything is done correctly, you should see the test pass, as shown in Figure 16.24.
14. Next, we will create our load test using the previous web test as workload. As mentioned earlier, you can record multiple web tests to simulate a more realistic mixture of workloads. But for simplicity, we will just use one web test here. Right-click the test project and select **Add→Load Test**. On the **New Load Test Wizard** dialog, click the **Next** button to continue.
15. On the next screen, accept all default settings and click the **Next** button to continue. By default, the load test will use think times under a normal distribution centered on recorded think times. This creates realistic variations in virtual user operation speeds.
16. The next screen allows you to pick a load pattern. You can use a constant load (default), or configure the load to ramp up gradually over the given period of time. A step load is very useful when you perform stress tests. Basically, the test will keep adding more users till the website starts to return errors (such as 500), or the site's performance drops below the expected threshold (such as response time exceeding 3 s). Here, we use the default setting and click the **Next** button to continue.

Request	Status	Total Time	Request Ti...	Request Bytes	Response Bytes
http://localhost:26173/	200 OK	0.045 sec	0.017 sec	0	560,988
http://localhost:26173/api/File	200 OK	0.003 sec	0.003 sec	0	1,024
http://localhost:26173/api/File	200 OK	0.026 sec	0.026 sec	0	10,240
http://localhost:26173/api/File	200 OK	1.028 sec	1.028 sec	0	10,240

Figure 16.24 Passed test.

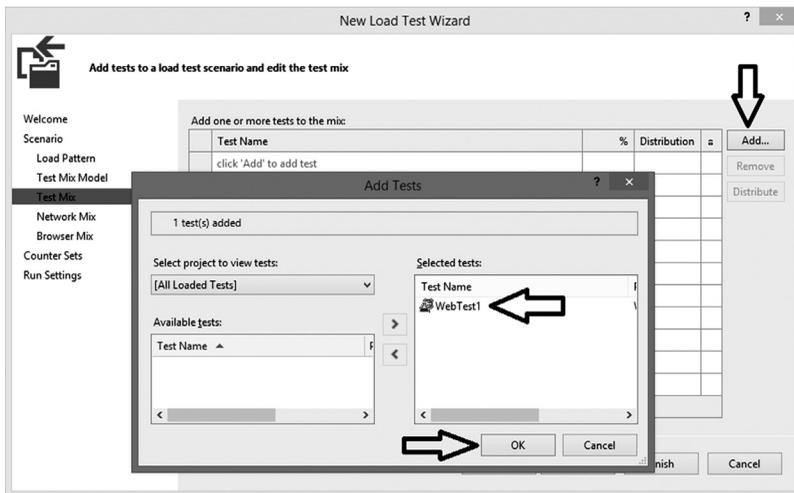


Figure 16.25 Adding web test to load test.

17. Click the **Next** button again and you will reach the **Test Mix** page, on which you can mix recorded tests together. Click on the **Add** button. Then, add the web test into the test mix, as shown in Figure 16.25.
18. Because we only have one test in the mix, by default it takes over 100% of the workload distribution. Click the **Next** button to continue.
19. On the next couple of screens, you can pick different network speed mixtures and browser mixtures for your load test. These mixtures are ideal for simulating the actual traffic to your site more realistically. Here, we accept all default settings and move forward.
20. On the screen where you can add computers to act as test controllers and test agents, simply click the **Next** button to continue without adding machines. This is because we leverage Visual Studio Online to manage test resources for us.
21. On the last screen, you can decide how long the load test should last. In addition, you can specify a warm-up period for your website to warm up. The warm up period allows JIT of web pages, as well as other initializations you may have, such as preloading cache, to execute so that when the actual load test starts, you have a stable environment that is fully prepped. Here, we give the site 1 min to warm up and we run the load test for 5 min, as shown in Figure 16.26. Of course, an actual load test would last much longer. Click the **Finish** button to complete the wizard.
22. Before we can run the load test on Visual Studio Online, we need to make the website accessible from cloud. So, we publish the website to Microsoft Azure.

Specify the length of the load test by:

Load test duration

Warm-up duration (hh mm ss):

Run duration (hh mm ss):

Figure 16.26 Specify test duration.

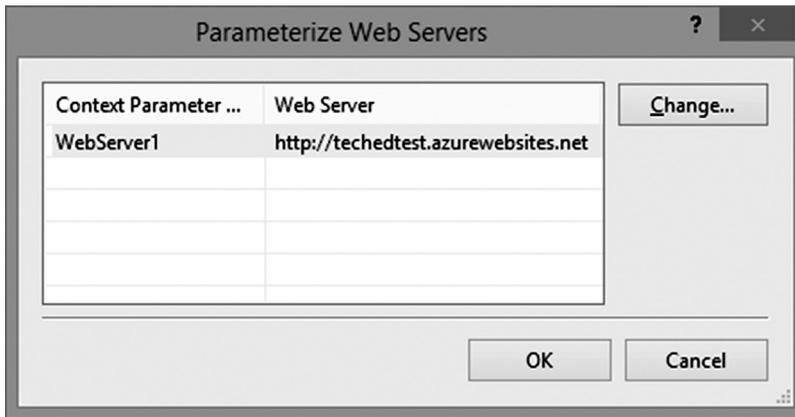


Figure 16.27 Parameterize web servers.

23. Next, we need to modify the web test to modify the recorded *loadhost* address to the published website address. Right-click on the web test and select the **Parameterize Web Servers** menu. Then, on the **Parameterize Web Servers** dialog, click on the **Change** button and enter the published website address. Click the **OK** button, and then press Ctrl + S to save the changes (Figure 16.27).
24. Back in **Solution Explorer**, double-click the **Local.testsettings** file under the **Solution Items** folder. Then, on the **Test Settings** dialog, select the option to **Run tests using Visual Studio Team Foundation Service**, as shown in Figure 16.28. Click **Next** to continue.
25. Keep clicking the **Next** button till you reach the last screen, where you can change the **Run tests in 32 bit or 64 bit process** to **Run tests in 64 bit process on 64 bit machine**. Click the **Apply** button and then the **Close** button.
26. Back in **Solution Explorer**, double-click the load test to open it.
27. Click the **Run Load Test** button to start the test. Visual Studio Online will provision the necessary machines and execute the test. After the test is completed, you can download the complete report by clicking the **Download report** link, as shown in Figure 16.29.
28. Click on the **Download report** link to download the report. Once the report is downloaded, click the **View report** link to open the report, as shown in Figure 16.30.

Due to the constraint of size, we cannot elaborate more on load tests here but we can leave several tips:

- Before you analyze the data in detail, examine the errors from the test agents. Usually, you would want 0 errors on the test agents so that you know the test result is not skewed because agent machines are stressed out. By default, Visual Studio Online automatically decides how

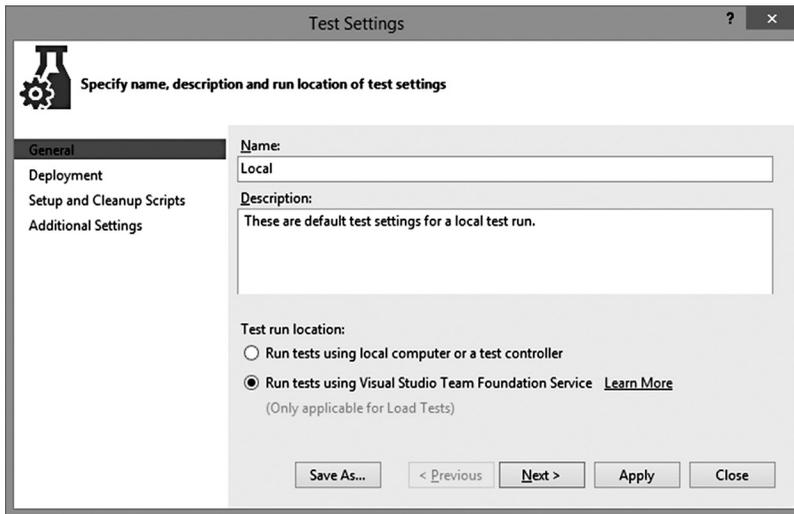


Figure 16.28 Configure load test to run on Visual Studio Online.

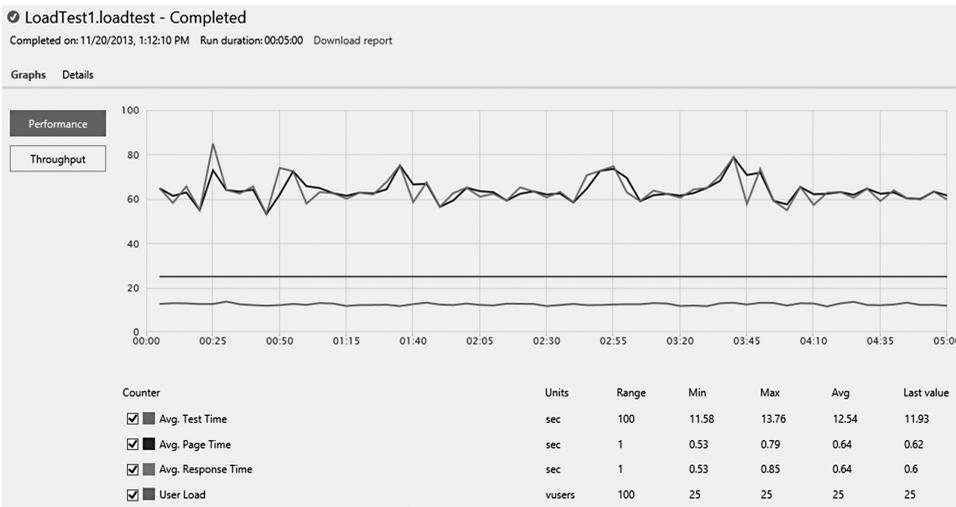


Figure 16.29 Load test result.

many agents need to be allocated for the test. You can change this behavior by changing the run settings of a load test. To do this, double-click the load test, and then right-click its run setting node (there should be a *Run Settings1* by default). Select the Properties menu, and then enter the number of agents you want to use to the **Agent count** field.

- Load test results should be evaluated against clear goals. For example, “Support 10,000 users” is not a clear goal. On the other hand, “Allow 10,000 users to place orders at the same time under an average of 3 s response time, without any server-side errors” is a very reasonable goal. In the second case, you can measure the response time as well as the error count under the specified load and compare the metrics against your goal.

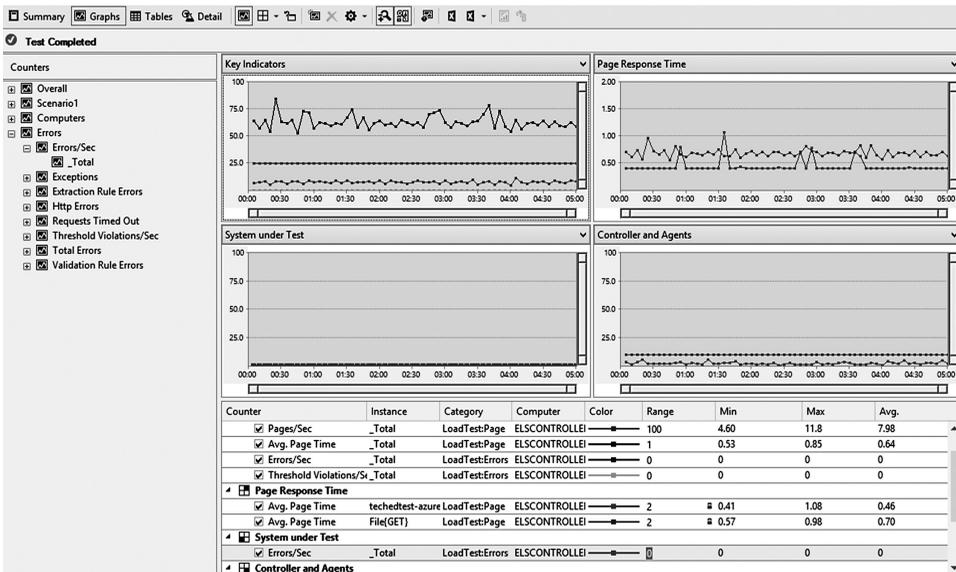


Figure 16.30 Sample test report.

- You should separate your load tests and your stress tests. The purpose of load tests is to determine whether the system can perform under expected loads. Stress tests, on the other hand, are to find out the upper limits of the system. It is usually advisable to run stress tests on a different deployment than your production environment as stress tests can be destructive. On the other hand, load tests can optionally be executed on the production environment at a lower scale—basically, you are sampling part of the production traffic to measure system performance under loads.
- You should also separate load tests from system profiling. Load tests are used to measure system performance under realistic traffic, while system profiling is often used to pinpoint specific performance issues. Maintaining the two separately allows you to better focus on the task at hand. Running and analyzing load tests is itself a lengthy and complex task, so you should try to avoid complicating it further and focus on getting the test result out.

Of course, here we have barely scratched the surface of Visual Studio Online. You may consult www.visualstudio.com and MSDN for further information.

16.5 Summary

Visual Studio Online provides a comprehensive collection of tools and services for your development needs. It helps to orchestrate individuals in your engineering team into an agile and productive team. In this chapter, we focused on source control and test features. Visual Studio Online offers much more, especially in project management. Project management and various methodologies are out of the scope of this book. The www.visualstudio.com **Getting Started** session has great information to help you to start using Visual Studio Online project management features, including managing backlogs, using Kanban board, managing sprints, and collaborating teams. Be sure to check it out and start leveraging its useful features in your projects.

Chapter 17

Scripting and Automation

Automated scripting has been a crucial tool for system administrators to manage various hardware and software systems. As a leading cloud platform, Microsoft Azure provides comprehensive scripting and automation supports for developers and system administrators. First of all, Microsoft Azure provides a complete set of REST-style management API for developers. Second, Microsoft Azure provides PowerShell cmdlets for managing various Microsoft Azure resources. In addition, Microsoft Azure also provides a command line tool for developers that use Mac or Linux systems. Cloud service providers can pick and choose among these offers based on their needs and automate common administrative tasks, hence to improve efficiency as well as reducing human errors.

17.1 Microsoft Azure PowerShell Cmdlets

Windows PowerShell is an automation framework based on .NET. It includes a command line environment and a scripting language. System administrators can manage Windows systems locally or remotely using PowerShell. The features of PowerShell are delivered as a series of cmdlets. PowerShell has a rich set of built-in cmdlets such as file I/O and WMI. You can also import additional cmdlets to extend PowerShell. Microsoft Azure is such a cmdlet developed for Microsoft Azure management tasks.

17.1.1 *Preparing a Microsoft Azure PowerShell Cmdlets Environment*

To use Microsoft Azure PowerShell cmdlets, you need Windows 8, Windows 7, Windows Server 2012, or Windows Server 2008 R2. Then, you can download and install Microsoft Azure PowerShell from <http://www.windowsazure.com/en-us/downloads/#cmd-line-tools>, or install it using Web Platform Installer (Web PI).

```

PS C:\> Set-ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): y
PS C:\>

```

Figure 17.1 Set up PowerShell execution policy.

As a security measure, the PowerShell execution policy defines conditions a script has to meet before it can be executed. For example, the execution policy can specify that only signed scripts can be executed, so as to avoid accidentally executing scripts from untrusted parties. Microsoft Azure PowerShell cmdlets requires the execution policy to be **RemoteSigned**, **Unrestricted**, or **Bypass**. To set up the execution policy, launch Microsoft Azure PowerShell as an administrator, and use the command **Set-ExecutionPolicy** to set up the execution policy, as shown in Figure 17.1.

Note: You need to launch Microsoft Azure PowerShell as administrator only when you set up the execution policy.

Before you can use Microsoft Azure cmdlets, you need to configure your workstation to establish the connection to Microsoft Azure. You can use the **Set-AzureSubscription** cmdlet and **Select-AzureSubscription** cmdlet to provide details of the subscription such as the management certificate. Or, you can configure the environment by downloading and importing a publish settings profile, just as you have done when you publish Cloud Services. Here we will use the second approach:

1. Launch a browser. Access the address <https://windows.azure.com/download/publishprofile.aspx> to download your publish settings file. After signing in to Microsoft Azure, save the publish settings file to a local folder.
2. In Microsoft Azure PowerShell, enter the command **Import-AzurePublishSettingsFile** <*publish settings file*> to complete the configuration step, as shown in Figure 17.2.

If your publish settings file contains multiple Microsoft Azure subscriptions, you can use this command to list your subscriptions:

```
Get-AzureSubscription | SELECT SubscriptionName
```

```

PS C:\> Import-AzurePublishSettingsFile C:\Haishi\subscriptions\haishi.publishsettings
VERBOSE: Setting: Pay-As-You-Go as the default and current subscription. To view other subscriptions use
Get-AzureSubscription

```

Figure 17.2 Import Microsoft Azure publish settings file.

You can use this command to select current subscription and to use current storage account:

```
Set-AzureSubscription -SubscriptionName <subscription name>
  -CurrentStorageAccount <storage account>
```

17.1.2 Managing Virtual Machines

We can use Microsoft Azure PowerShell to manage Microsoft Azure Virtual Machines. We will learn how to create and delete a Virtual Machine using PowerShell through an example. In the example, we use the following cmdlets:

- **Get-AzureVMImage**

This cmdlet returns a list of Virtual Machine images that are available to you, for example:

```
PS C:\>Get-AzureVMImage | SELECT ImageName
ImageName
-----
0b11de9248dd4d87b18621318e037d37__RightImage-CentOS-6.2-x64-v5.8.8.1
0b11de9248dd4d87b18621318e037d37__RightImageCentOS-6.3-x64-v5.8.8
0b11de9248dd4d87b18621318e037d37__RightImage-CentOS-6.3-x64-v5.8.8.5
0b11de9248dd4d87b18621318e037d37__RightImage-CentOS-6.3-x64-v5.8.8.6
0b11de9248dd4d87b18621318e037d37__RightImage-CentOS-6.3-x64-v5.8.8.7
0b11de9248dd4d87b18621318e037d37__RightImage-CentOS-6.3-x64-v5.8.8.8
0b11de9248dd4d87b18621318e037d37__RightImage-CentOS-6.3-x64-v5.8.8.9
0b11de9248dd4d87b18621318e037d37__RightImage-CentOS-6.4-x64-v13.4
...
```

- **Get-AzureLocation**

This command returns the list of regions you can deploy your services to, for example,

```
PS C:\>Get-AzureLocation | SELECT DisplayName
DisplayName
-----
West Europe
Southeast Asia
East Asia
North Central US
North Europe
South Central US
West US
East US
```

- **New-AzureQuickVM**

This cmdlet creates a Virtual Machine. See the following example for details.

- **Remove-AzureVM**

Delete a Virtual Machine. See the following example for details.

Example 17.1: Managing Virtual Machines using Microsoft Azure PowerShell

Difficulty: **

1. First, set the image you want to use. You can use **Get-AzureVMImage** to get a list of images.

```
$image = "0b11de9248dd4d87b18621318e037d37__RightImage-CentOS-6.4-x64-v13.4"
```

2. Then, set the region you want to deploy your Virtual Machine. You can use **Get-AzureLocation** to get a list of regions.

```
$location = "West US"
```

3. Set the name of the Cloud Service corresponding to the Virtual Machine and the password of the administrator:

```
$svcName = "hbaivmservice02"
$admPass = "P@ssword$123"
```

Note: The service name has to be globally unique. You can use **Test-AzureName-Service <name>** to test if a name is valid. Note that the cmdlet returns **False** to indicate that a name has not been used, hence usable.

4. At last, use the **New-AzureQuickVM** cmdlet to create a Virtual Machine:

```
New-AzureQuickVM -Linux -name "hbailinux" -ImageName $image -
  ServiceName $svcName -Location $location -Password $admPass -
  LinuxUser "haishi"
```

5. To delete a Virtual Machine:

```
Remove-AzureVM -ServiceName $svcName -Name "hbailinux"
```

Common Virtual Machine-related cmdlets are summarized in Table 17.1.

Note: Configure a Health Probe for a Virtual Machine

In Example 7.7 of this book, we introduced how to add multiple virtual machines to a load balance to achieve high availability. You can also use the **Set-AzureEndpoint** cmdlet and

Table 17.1 Virtual Machine Cmdlets

<i>Cmdlet</i>	<i>Description</i>
Add-AzureDataDisk	Attach a data disk to a virtual machine
Add-AzureDisk	Add a virtual disk to Microsoft Azure virtual disk gallery
Add-AzureVhd	Upload a .vhd file to Microsoft Azure Storage account
Add-AzureVMImage	Add a new OS image to the image gallery
Get-AzureDataDisk	Read data disk objects
Get-AzureDisk	Read virtual disk objects from Microsoft Azure virtual disk gallery
Get-AzureOSDisk	Read the OS virtual disk object on a virtual machine
Get-AzureVM	Read one or more virtual machine objects
Get-AzureVMImage	Read available virtual machine images
Import-AzureVM	Import Microsoft Azure virtual machine state from a file
New-AzureQuickVM	Create a virtual machine using simple settings
New-AzureVM	Create a virtual machine
New-AzureVMConfig	Create a virtual machine configuration object
Remove-AzureDataDisk	Remove a data disk from a virtual machine
Remove-AzureDisk	Remove a virtual disk from Microsoft Azure virtual disk gallery
Remove-AzureVM	Remove a virtual machine
Remove-AzureVMImage	Remove an OS image from the image gallery
Restart-AzureVM	Restart a virtual machine
Save-AzureImage	Capture and save a mirror image of a stopped virtual machine
Set-AzureVMSize	Set virtual machine size
Start-AzureVM	Start a virtual machine
Stop-AzureVM	Shut down a virtual machine
Update-AzureDisk	Update the label of a disk in Microsoft Azure virtual disk gallery
Update-AzureVM	Update a virtual machine
Update-AzureVMImage	Update the label of an OS image in the image gallery

the Update-AzureVM cmdlet to specify a custom health probe for virtual machines. For example, the following command

```
Get-AzureVM -ServiceName "MyService" -Name "MyVM" | Set-AzureEndpoint
-LBSetName "MyLBSet" -Name MyEndpoint2 -Protocol tcp -LocalPort 80
-ProbePort 80 -ProbeProtocol http -ProbePath "/" | Update-AzureVM
```

specifies a health probe based on HTTP, port 80, and at path “/”.

17.1.3 *Managing Cloud Services*

You can also use Microsoft Azure PowerShell to manage Cloud Services. For example, to create a new Cloud Service using PowerShell is very easy:

```
NewAzureService -ServiceName <service name> -AffinityGroup <affinity
group>
```

You can use the following command to get the list of affinity groups:

```
Get-AzureAffinityGroup | Select Name
```

Common Cloud Service–related cmdlets are summarized in Table 17.2.

17.1.4 *Managing Microsoft Azure Websites*

Similarly, you can manage Microsoft Azure Websites using Microsoft Azure PowerShell cmdlets. Common website-related cmdlets are summarized in Table 17.3.

17.1.5 *Other Cmdlets*

In addition to the preceding cmdlets, you can use Microsoft Azure PowerShell cmdlets to manage various Microsoft Azure resources such as subscriptions, storage accounts, service bus entities, and virtual networks. Interested readers may consult related MSDN documents.

17.2 **Microsoft Azure Cross-Platform Command Line Tools**

Microsoft Azure Cross-Platform Command Line Tool provides a complete set of tools for Linux and Mac users to manage Microsoft Azure resources. You can use the Command Line Tools to perform various operations such as creating Virtual Machines and managing websites and Mobile Services.

Table 17.2 Cloud Service Cmdlets

<i>Cmdlet</i>	<i>Description</i>
Add-AzureCacheWorkerRole	Add a dedicated caching role
Add-AzureCertificate	Upload a certificate
Add-AzureDjangoWebRole	Add a new Python Django web role
Add-AzureNodeWebRole	Add a new Node.js web role
Add-AzurePHPWebRole	Add a PHP web role
Add-AzureWebRole	Add a web role
Add-AzureNodeWorkerRole	Add a Node.js worker role
Add-AzurePHPWorkerRole	Add a PHP worker role
Add-AzureWorkerRole	Add a worker role
Disable-AzureServiceProjectRemoteDesktop	Disable remote desktop access
Enable-AzureMemcacheRole	Enable Memcache protocol on a caching role
Enable-AzureServiceProjectRemoteDesktop	Enable remote desktop access
Get-AzureAffinityGroup	List affinity groups
Get-AzureCertificate	Read certificate
Get-AzureDeployment	Read deployment details
Get-AzureOSVersion	List currently supported OS versions
Get-AzureRemoteDesktopFile	Read remote desktop file (.RDP)
Get-AzureRole	List roles in a Cloud Service
Get-AzureService	List all Cloud Services under current subscription
Get-AzureServiceProjectRoleRuntime	List all supported role environments, such as IIS and PHP
Move-AzureDeployment	Swap staging and production deployment
New-AzureAffinityGroup	Create a new affinity group
New-AzureDeployment	Deploy a Cloud Service
New-AzureRoleTemplate	Create a web/worker role template
New-AzureService	Create a new Cloud Service
Publish-AzureServiceProject	Deploy current service to Microsoft Azure
Remove-AzureAffinityGroup	Delete an affinity group

(Continued)

Table 17.2 (Continued) Cloud Service Cmdlets

<i>Cmdlet</i>	<i>Description</i>
Remove-AzureCertificate	Delete a certificate
Remove-AzureDeployment	Delete a Cloud Service deployment
Reset-AzureRoleInstance	Reset or remirror role instances
Save-AzureServiceProjectPackage	Package a Cloud Service as a .cspkg file
Set-AzureAffinityGroup	Update affinity group properties
Set-AzureDeployment	Update Microsoft Azure deployment states, update method or configuration
Set-AzureRole	Update number of instances of a role
Set-AzureService	Set or update Cloud Service's label and description
Set-AzureServiceProject	Set default region, subscription, deployment environment, and storage account
Set-AzureWalkUpgradeDomain	Perform an update domain walk
Start-AzureEmulator	Launch computer emulator and storage emulator
Start-AzureService	Start a Cloud Service
Stop-AzureEmulator	Shutdown emulators
Stop-AzureService	Shutdown a Cloud Service
Test-AzureName	Test if a service name or storage account name has been taken

17.2.1 Installing the Command Line Tools

Under Mac, you can download and install the Command Line Tool package from this address: <http://go.microsoft.com/fwlink/?linkid=252249&clid=0x409>.

Under Linux, you can use the following command to install the package:

```
npm install azure-cli -g
```

Moreover, you can install the Windows-based version from the following address, or install it using Web PI:

<http://www.windowsazure.com/en-us/downloads/>

By default, the Windows-based version is installed under the folder `c:\Program Files (x86)\Microsoft SDKs\Microsoft Azure\CLI\wbin`. You can launch the tools using `azure.cmd`, as shown in Figure 17.3.

17.2.2 Getting Started with the Command Line Tools

First, you need to import the public settings file:

```
azure account import <public settings files>
```

The result of this command is shown in Figure 17.4.

Then, you can use the following command to set the current subscription:

```
azure account set <subscription>
```

Now, your environment is ready for various Microsoft Azure operations. For example, Figure 17.5 demonstrates how to use the **azure service list** command to enumerate Cloud Services under the current subscription.

Microsoft Azure Cross-Platform Command Line Tools comes with complete online documents. When you get started following the previous steps, you can explore the usage of other commands with the help of online documents.

```
C:\Program Files (x86)\Microsoft SDKs\Windows Azure\CLI\wbin>azure account import
t c:\Haishi\Subscriptions\haishi.publishsettings
info: Executing command account import
info: Found subscription: Pay-As-You-Go
info: Found subscription: Azdem169H43081X
info: Found subscription: Introductory Package
info: Found subscription: ASP.NET Online Demos
info: Setting service endpoint to: https://management.core.windows.net/
info: Setting default subscription to: Pay-As-You-Go
info: Use "azure account set" to change to a different one.
warn: The 'c:\Haishi\Subscriptions\haishi.publishsettings' file contains sensitive information.
warn: Remember to delete it now that it has been imported.
info: Account publish settings imported successfully
info: account import command OK
```

Figure 17.4 Import Microsoft Azure publish settings file.

```
C:\Program Files (x86)\Microsoft SDKs\Windows Azure\CLI\wbin>azure service list
info: Executing command service list
+ Fetching cloud services
data: Name Status
data: -----
data: firework Created
data: haishicdn Created
data: hbaivmservice02 Created
data: horsevm Created
data: lamp-apache-te Created
data: partinspector Created
data: sayapicture Created
data: toberemoved Created
info: service list command OK
```

Figure 17.5 Enumerate Cloud Services under a subscription.

CODE LIST 17.1 MICROSOFT AZURE MANAGEMENT API SAMPLE

```

1: using System;
2: using System.IO;
3: using System.Net;
4: using System.Security;
5: using System.Security.Cryptography;
6: using System.Security.Cryptography.X509Certificates;
7: using System.Xml;
8:
9: namespace ManagementAPIListServices
10: {
11:     class Program
12:     {
13:         static void Main(string[] args)
14:         {
15:             string subscriptionId = "<subscription id>";
16:             string operationName = "hostedservices";
17:             Uri requestUri = new Uri("https://management.core.
                windows.net/"
18:                 + subscriptionId
19:                 + "/services/"
20:                 + operationName);
21:             HttpWebRequest request =
22:                 (HttpWebRequest)HttpWebRequest.
                    Create(requestUri);
23:             request.Headers.Add("x-ms-version", "2010-10-28");
24:             request.Method = "GET";
25:             request.ContentType = "application/xml";
26:             //locate certificate
27:             string certThumbprint = "<certificate thumbnail>";
28:             X509Store certStore = new X509Store(StoreName.My,
29:                 StoreLocation.CurrentUser);
30:             certStore.Open(OpenFlags.ReadOnly);
31:             var certs = certStore.Certificates.Find
32:                 (X509FindType.FindByThumbprint, certThumbprint,
                    false);
33:             certStore.Close();
34:             //attach the certificate to the request
35:             request.ClientCertificates.Add(certs[0]);
36:             HttpWebResponse response = (HttpWebResponse)request.
                GetResponse();
37:             using (StreamReader reader = new
38:                 StreamReader(response.GetResponseStream()))
39:             {
40:                 //parse returned XML file
41:                 XmlDocument doc = new XmlDocument();
42:                 doc.LoadXml(reader.ReadToEnd());
43:                 XmlNamespaceManager nm =
44:                     new XmlNamespaceManager(doc.
                        NameTable);

```

```

45:             nm.AddNamespace("i",
46:             "http://www.w3.org/2001/XMLSchema-
                instance");
47:             nm.AddNamespace("M",
48:             "http://schemas.microsoft.com/
                windowsazure");
49:             var services = doc.SelectNodes("//
                M:HostedService",nm);
50:             Console.WriteLine(string.Format("{0,-20}{1,-20}
                {2,-35}",
51:             "Name", "Location", "Url"));
52:             Console.WriteLine(string.Format("{0,-20}{1,-20}
                {2,-35}",
53:             new string(' ', 19), new string(' ', 19),
54:             new string(' ', 35)));
55:             for (int i = 0; i < services.Count; i++)
56:             {
57:                 var locationNode = services[i].
                    SelectSingleNode
58:                 ("M:HostedServiceProperties/M:Location",
                    nm);
59:                 Console.WriteLine(string.Format("{0,-20}
                {1,-20}{2,35}",
60:                 services[i].SelectSingleNode("M:ServiceName",
                    nm)
61:                 .InnerText,
62:                 locationNode == null? "-----":
63:                 locationNode.InnerText,
64:                 services[i].SelectSingleNode("M:Url", nm)
65:                 .InnerText.Substring(0,32) + "...")
66:                 ));
67:             }
68:         }
69:     }
70: }
71: }

```

Name	Location	Url
firework	West US	https://management.core.windows...
haishicdn	North Central US	https://management.core.windows...
partinspector	West US	https://management.core.windows...
sayapicture	East Asia	https://management.core.windows...
toberemoved	-----	https://management.core.windows...

Press any key to continue . . .

Figure 17.7 Enumerate Cloud Services using Microsoft Azure Management API.

17.4 Summary

In this chapter, we provided a brief introduction of Microsoft Azure PowerShell cmdlets, Microsoft Azure Cross-Platform Command Line Tools, and Microsoft Azure Management API. Of course, the focus of this chapter is merely to get you started so that you can start exploring the rich functionalities of these tools by yourself.

Chapter 18

Azure and DevOps

Microsoft Azure is under constant development. At the time of writing, tons of new features are being added to it. In this chapter, we will discuss some of Azure's exciting new features that reflect its future directions for evolution. Since it is impossible to enumerate all of the new features of Azure in a single chapter, we will focus only on those features that revolve around DevOps. As the name suggests, the essence of DevOps is to ensure smooth interaction between development and operations. Many of Azure's new features resonate with this idea. And Microsoft is providing additional tools and infrastructural support to enable DevOps practices.

18.1 DevOps Overview

Before looking at specific features, let us briefly review what DevOps really means. Because DevOps is still relatively new, there has not been a universally accepted definition. Generically speaking, DevOps is a collection of ideas, methodologies, practices, and tools that encourage close team collaboration, improve business agility, as well as IT alignment. If you know about Agile software development, many of these ideas should sound familiar to you. DevOps is a continuation of the Agile movement, and IT alignment across teams is a key aspect of DevOps.

It is rather difficult to cover DevOps in its entirety in just a few paragraphs because DevOps is still an evolving concept that impacts software business at different levels. Instead, we will list a few *DevOps phenomena* that are likely to occur in your working environment. Of course, what we describe here are exaggerated, ideal scenarios. What you will encounter in the real world will probably be less extreme and sometimes subtle.

18.1.1 Everything Is Code

Developers are a creative bunch and they always want change, while one of the top-priority tasks of IT professionals is to keep systems stable. This has indeed been an unresolvable conflict that has caused much friction between the two rival tribes, developers and IT professionals.

From the viewpoint of DevOps, a software (or a service) is a holistic unit that consists of application code, configuration, infrastructure, and data. When a software is revised or updated, not only should the application code be versioned and archived, the infrastructural changes should also be captured and versioned at the same time. This requires IT professionals and developers to communicate with each other with clarity, reliability, and traceability. Capturing everything as abstracted code and scripts serve the purpose perfectly.

With the responsibility of maintaining physical servers on cloud, IT professionals find themselves working more and more at an abstraction level where resources are described and configured by scripts. This abstraction allows infrastructures to be defined and maintained as *code*. On the other hand, because the infrastructural code can be executed by any authorized users, developers can deploy their own development and testing environments by themselves while keeping a high confidence that the production environments will be constructed and maintained in a consistent manner. As the developers gain more knowledge on infrastructural code, they can start manipulating the infrastructure at the abstracted level, and IT professionals in turn can fine-tune the code for large-scale, optimized deployment.

18.1.2 Everyone Is a Developer

DevOps is not a new job title that sits between developments and operations. It reflects the fusion of the two departments. Such convergence brings new challenges to all staff. Developers are often more capable of coping with such challenges—after all, code is what they live, eat, and breathe! So it is not surprising to see developers becoming superheroes, who take care of everything. This phenomenon is often observed in start-ups, who have been major advocators and beneficiaries of the movement.

For larger enterprises and interdependent software vendors (ISVs), it is unrealistic to expect such complete fusion. Instead, operations personnel behave more and more like developers as they start to work with abstract artifacts, codes, and scripts. Although, in many larger enterprises, the work style and human resource allocations have been fixed by existing architectures and legacy systems. However, this does not mean these companies cannot adopt DevOps for their new projects at a smaller scale.

18.1.3 Every Day Is Release Day

The landscape of marketing has drastically changed over the past decade. Consumers are more proactive than ever. Nowadays, people use technology at their fingertips to search for products and services instead of waiting to hear something they might like in marketing campaigns. For a service provider, getting a higher rank in search engines and a better rating in popular sites are becoming far more important than marketing campaigns. And once they get a customer, they also need to take measures to retain that customer, as the customer is in a position to find *the next big thing* at any time.

Continuous engagement with customers has become key to sustainable business. And with development and operations folding into one unit, products can be released more frequently, changes can be applied quickly, and new features can be introduced more promptly. Such unprecedented agility provides ISVs opportunities to respond to market changes faster and to maintain relationships with their customers.

18.2 VM Agent and VM Extensions

Configuration automation is a fundamental requirement of DevOps. Microsoft Azure provides both first-party and third-party options to capture and automate machine configurations at scale. And all of this is possible because of VM Agent, which we discuss next.

18.2.1 VM Agent

VM Agent is a lightweight process that can pull down additional VM Extensions, which are software packages that can be dynamically installed in your virtual machines to customize them to meet your specific project needs. When you create a new virtual machine using Azure Management Portal, the VM Agent is enabled by default, as shown in Figure 18.1. If you want to install VM Agent to an older virtual machine, which does not come with a built-in VM Agent, you can download and run the Agent installer from this link: <http://go.microsoft.com/fwlink/?LinkID=394789&clcid=0x409>.

18.2.2 VM Extensions

Once VM Agent is installed on a virtual machine, it can acquire and install additional VM Extensions from the Azure VM Extension Gallery, as shown in Figure 18.2.

At the time of writing, there are only a limited number of VM Extensions available from the VM Extension Gallery. You can use PowerShell cmdlet *Get-AzureVMAvailableExtension*

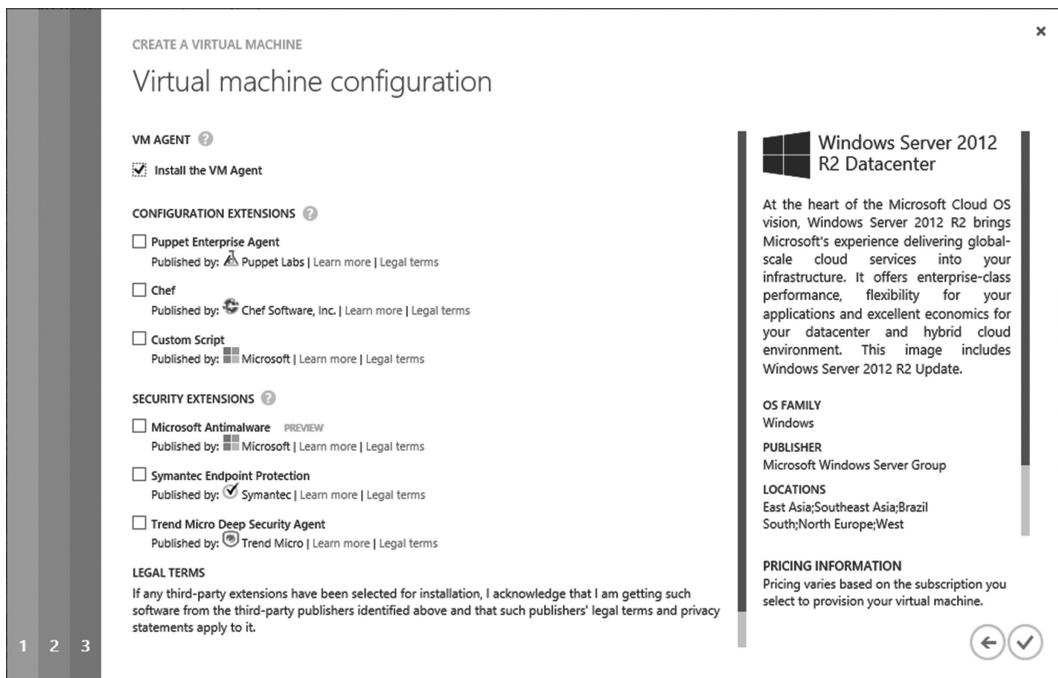


Figure 18.1 VM Agent is enabled by default.

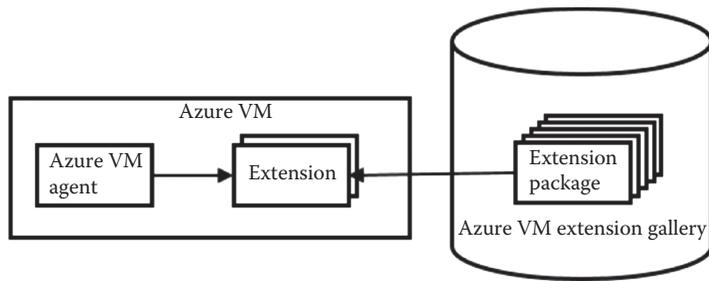


Figure 18.2 Install VM Extensions from Gallery.

```
PS C:\windows\system32> Get-AzureVMAvailableExtension | Select ExtensionName, Publisher, Version
```

ExtensionName	Publisher	Version
MSEnterpriseApplication	Microsoft.SystemCenter	1.0
ChefClient	Chef.Bootstrap.WindowsAzure	11.12
HpcVmDrivers	Microsoft.HpcComputeTest	1.1
PPHandlerTesting2	TrendMicro.PortalProtect.Tes...	2.3
PPHandlerTesting	TrendMicro.PortalProtect.Tes...	2.2
PPHandler	TrendMicro.PortalProtect	2.1
BGInfo	Microsoft.Compute	1.1
CustomScriptExtension	Microsoft.Compute	1.0
VMAccessAgent	Microsoft.Compute	1.0
PuppetEnterpriseAgent	PuppetLabs	3.2

Figure 18.3 List of available VM Extensions.

```
PS C:\windows\system32> $vm = Get-AzureVM -ServiceName haishiwin2012 -Name haishiwin2012
PS C:\windows\system32> Get-AzureVMExtension -VM $vm | Select ExtensionName, Publisher, Version
```

ExtensionName	Publisher	Version
BGInfo	Microsoft.Compute	1.*

Figure 18.4 List of Extensions installed on a VM.

(requires Microsoft Azure PowerShell; see Chapter 17) to list currently available extensions, as shown in Figure 18.3.

If you want to list extensions that are already installed on a VM, you can use the *Get-AzureVMExtension* command, as shown in the example in Figure 18.4.

18.2.3 Custom Script Extension

Custom Script Extension is a VM Extension that can automatically download and execute PowerShell scripts and files from Azure Storage. Although the functionality of this agent is very simple, it can help you accomplish lots of customization scenarios because of the capability of PowerShell.

To install the Custom Script Extension, you can use cmdlet *Set-AzureVMExtension*, as shown in Figure 18.5.

```
PS C:\windows\system32> Set-AzureVMExtension -ExtensionName CustomScriptExtension -VM $vm -Publisher Microsoft.Compute | Update-AzureVM
```

OperationDescription	OperationId	OperationStatus
Update-AzureVM	6be82852-7fc3-934e-b75d-6c0abf95c63b	Succeeded

Figure 18.5 Install Custom Script Extension.

Once the Custom Script Extension has been installed, you can start to pull down additional PowerShell scripts from your Azure Storage account and execute them. In this case, I have uploaded a simple PowerShell script that simply prints “Hello World” to the output stream:

```
Host-Write "Hello World!"
```

Assume the script has been uploaded to my BLOB container *powershell* under my *haishigateway* account, then I can use *Set-AzureVMCustomScriptExtension* command to launch the script and update my virtual machine:

```
Set-AzureVMCustomScriptExtension -ContainerName powershell
-StorageAccountName haishigateway -FileName HelloWorld.ps1 -VM $vm
-ReferenceName 'CustomScriptExtension' | Update-AzureVM -Verbose
```

The output of this command is shown in Figure 18.6.

To get the execution result, which can be written to either Stdout stream and/or StdErr stream, you need to refresh your virtual machine reference and then read the script execution result from its **ResourceExtensionStatusList.ExtensionSettingStatus.SubStatusList** property, as shown in Figure 18.7.

18.2.4 DSC, Puppet, and Chef

Desired state configuration (DSC), Puppet, and Chef are all configuration automation systems. They allow you to choose desired states for your resources, and they make sure your requirements are met. With these systems, configurations are centralized, and configuration updates are either pushed from a server or periodically pulled by agents installed on target machines.

```
WARNING: No Run File has been assigned, and the Custom Script extension will try to use the first specified File Name as the Run File.
VERBOSE: 3:50:06 PM - Completed Operation: Get Deployment
VERBOSE: 3:50:06 PM - Begin Operation: Update-AzureVM
VERBOSE: 3:50:39 PM - Completed Operation: Update-AzureVM
```

OperationDescription	OperationId	OperationStatus
Update-AzureVM	d2c1f45f-ce0e-9ec7-92c1-566d6df7fa33	Succeeded

Figure 18.6 Output of custom script run.

```
PS C:\windows\system32> $vm.ResourceExtensionStatusList.ExtensionSettingStatus.SubStatusList | Select Name, @{Label=""Message"; Expression = {$_.FormattedMessage.Message}}
```

Name	Message
StdOut	Hello World!
StdErr	

Figure 18.7 Retrieve script execution results.

You may have noticed that both Puppet Agent and Chef Client are provided as VM Extensions (see Figure 18.1). DSC is part of PowerShell 4, and your configurations can be applied via mechanisms such as the Custom Script Extension we introduced in the previous section.

As we have discussed at the beginning of this chapter, being able to capture and automatically reinforce infrastructural requirements across different environments is one of the key enabling technologies for DevOps. Unfortunately, we cannot cover these systems here as each deserves an entire book. Interested readers can visit the author's blog (<http://blog.haishibai.com>) to find several tutorials on these technologies.

18.3 New Portal

Early in 2014, a new Microsoft Portal was put into preview. Existing Azure subscribers can visit <https://portal.azure.com/microsoft.onmicrosoft.com> to access the new portal (Figure 18.8).

The new portal looks fairly different from the current portal. The current portal design is resource oriented, where you can provision, manage, and monitor your Azure resources with ease. The new portal focuses on your applications instead of scattered resources. When you deploy an application to cloud, you probably need to provision a bunch of related resources, such as a web server, an application server, and a database. In the current portal, you cannot really treat them as a single unit. However, the new portal supports the concept of Resource Groups, which allow you to manage a group of related cloud resources as a single unit.

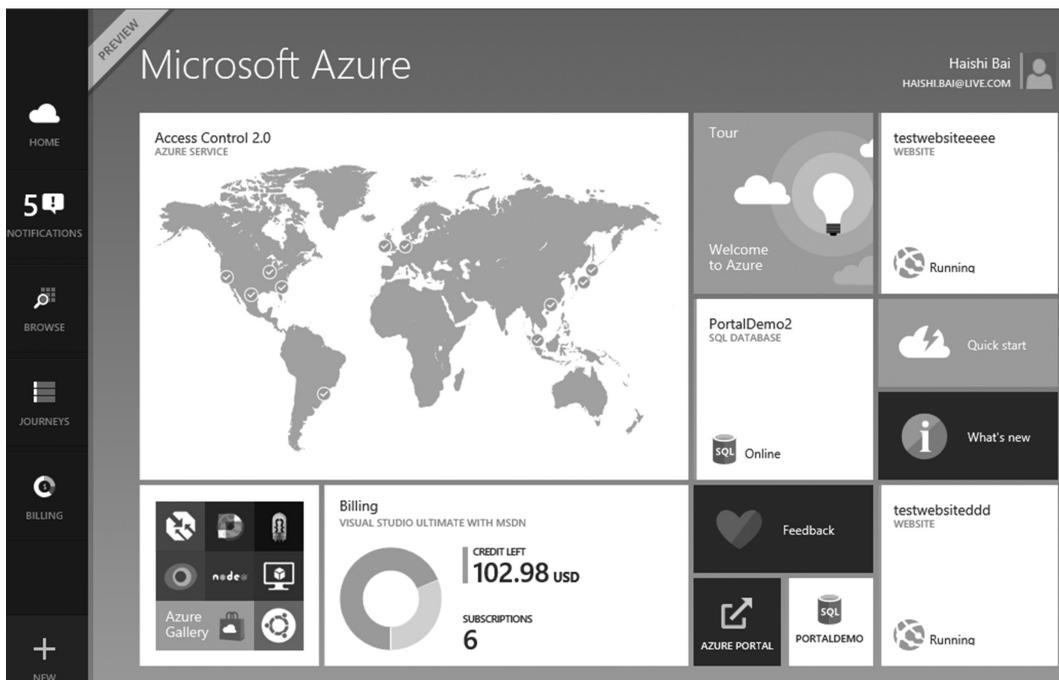


Figure 18.8 Microsoft Azure Preview Portal.

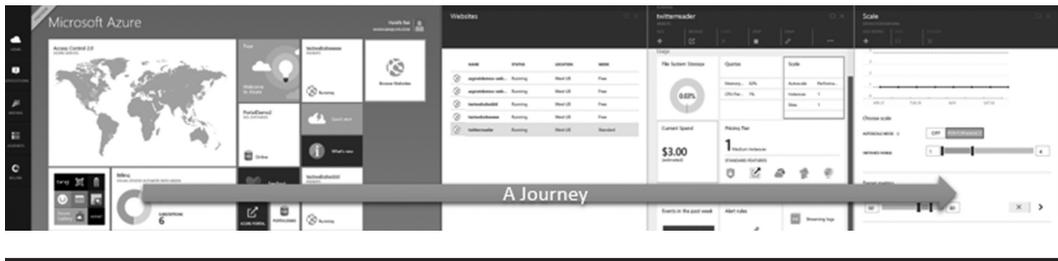


Figure 18.9 A Journey on Azure portal.

The new portal is also highly customizable. You can manage your home page tiles just as you can do on a Windows 8 PC or Windows phone. You can pin everything related to your application on the home page so that you can have a consolidated view of your application.

Another concept of the new portal is Journey. When you perform operations, the screen keeps expanding to the right, guiding you through multistep operations and revealing more and more relevant information (see Figure 18.9). And as you complete operations, you are brought back toward the left till you return to the home page. A Journey is saved across browser sessions. So the next time you log in, or open the portal from a different machine, you can pick up from where you left off.

The new portal provides strong support for DevOps. It is fully integrated with Visual Studio Online, Azure monitoring, billing, and various other services so that you can gain incredible insights into every aspect of your application. The dynamic UI may seem a little intimidating at the beginning, but once you have tried out a couple of operations, you will find out that the new portal is really easy to use and can become a vital tool for your DevOps scenarios.

18.4 Zen of Cloud

The core concept of cloud is to separate business applications from underlying infrastructures. The value of cloud resides in reduced operation cost, improved quality of service, and unprecedented agility. The opportunity brought by cloud is the innovations that can happen once applications are freed from the constraints of infrastructures. Finally, the future of cloud is Ubiquitous Cloud promoting Ubiquitous Computing to the next level, where the real world and the cyber world merge into one.

No matter how technology evolves, at the end, people want the required functionalities to be delivered and to remain accessible. The separation of applications and infrastructure allows application and service developers to better focus on developing core business values, without having to worry about how to tailor the application for particular platforms. It is foreseeable that in the near future, local data centers, or even single server boxes, will adopt the same infrastructure designs as those for cloud platforms. The parity between cloud, local systems, and even mobile systems unifies the application ecosystem so that an application can be written once and be made available everywhere. An application or a service can be migrated freely among on-premises datacenters, private clouds, community clouds, and public clouds. Choosing a hosting environment will no longer be an architectural decision, but just an operational decision.

For many software developers, the key for adopting cloud is the shift in mindset from scaling out to scaling up, from controlled environment to dynamically composed environment, and from

single-instance reliability to collective reliability. None of these concepts are new. Actually, experienced large-scale system designers and developers have been doing these things for years. It is just that they are getting the public's long-overdue attention now.

The real revolutionary changes brought by cloud reside in the area of Big Data and Internet of Things (IoT). Both Big Data and IoT need the capability of large scale, highly available data acquisition, transmission, storage, and manipulation. Cloud provides all of these. It infuses new, potent, infinite energy into Big Data and IoT development, making them both thrive on cloud.

Cloud is still young, and the developments in the field have been phenomenal. We hope this book provides some practical help along your cloud journey, and we wish you great success with your cloud projects. Thank you!

Bibliography

- Bertocci, V. *Programming Windows Identity Foundation*. Redmond, WA: Microsoft Press, 2011.
- Brown, J. W. *Anti Patterns*. New York: Wiley, 1998.
- Dobson, S. M. *Streetwise Project Management*. Avon, MA: Adams Media Corporation, 2003.
- Erl, T. *SOA Design Patterns*. Upper Saddle River, NJ: Prentice Hall, 2008.
- Ferraiolo, D. F., Kuhn, D. R., Chandramouli, R. *Role-Based Access Control*, 2nd edition. Norwood, MA: Artech House Publishers, 2007.
- Frithey, G., Sajal, D. *SQL Server 2008 Query Performance Tuning Distilled*. New York: Apress, 2009.
- Galloway, J. et al. *Professional ASP.NET MVC 4*. Indianapolis, IN: John Wiley & Sons, Inc., 2012.
- Hohpe, G., B. Woolf. *Enterprise Integration Patterns*. Boston, MA: Addison Wesley, 2004.
- Microsoft. *A Guide to Claims-Based Identity and Access Control*. Redmond, WA: Microsoft, 2010.
- Microsoft. MSDN online documentations. [Online] Microsoft, 2013a. <http://msdn.microsoft.com>, accessed May 12, 2013.
- Microsoft. Windows azure. [Online] Microsoft, 2013b. <http://www.windowsazure.com>, accessed May 12, 2013.
- Sommerville, I. *Software Engineering*. Upper Saddle River, NJ: Pearson Education Ltd., 2001.

Information Technology

This book explains the various concepts of Azure in a logical and clear manner. ... The book consists of 69 complete, end-to-end examples that provide step-by-step guidance on implementing typical cloud-based scenarios. The examples cover a wide range of application types and technologies with different levels of difficulties.

—Pierre Masai, CIO of Toyota Motor Europe

Zen of Cloud: Learning Cloud Computing by Examples on Microsoft Azure provides comprehensive coverage of the essential theories behind cloud computing and the Windows Azure cloud platform. Sharing the author's insights gained while working at Microsoft's headquarters, it presents nearly 70 end-to-end examples with step-by-step guidance on implementing typical cloud-based scenarios.

The book is organized into four sections: cloud service fundamentals, cloud solutions, devices and cloud, and system integration and project management. Each chapter contains detailed exercises that provide readers with the opportunity to develop valuable hands-on skills in cloud service development.

- Explains how to prepare for Microsoft Azure development and how to use Microsoft Azure Management Portal
- Provides best practices for designing cloud-based applications
- Includes online access to updated examples and answers to the exercises

Supplying comprehensive coverage of the Windows Azure cloud platform, the book provides a practical understanding and powerful tips that readers can immediately apply to their own work—making it ideal for cloud system developers, architects, and IT professionals. Organized into easily digestible sessions, it is also ideal for use in instructional settings.



CRC Press
Taylor & Francis Group
an **informa** business
www.crcpress.com

6000 Broken Sound Parkway, NW
Suite 300, Boca Raton, FL 33487
711 Third Avenue
New York, NY 10017
2 Park Square, Milton Park
Abingdon, Oxon OX14 4RN, UK

K22024

ISBN: 978-1-4822-1580-9



www.crcpress.com

