

THE EXPERT'S VOICE® IN .NET

SECOND EDITION

Windows Azure Platform

Tejaswi Redkar and Tony Guidici

Apress®

www.allitebooks.com

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.



Contents at a Glance

■ About the Authors	xvi
■ About the Technical Reviewer	xvii
■ Acknowledgments	xviii
■ Introduction	xx
■ Chapter 1: Windows Azure Platform Overview	1
■ Chapter 2: Windows Azure Compute	49
■ Chapter 3: Windows Azure Storage Part I – Blobs and Drives	131
■ Chapter 4: Windows Azure Storage Part II – Queues	207
■ Chapter 5: Windows Azure Storage Part III – Tables	247
■ Chapter 6: VM Role and Windows Azure Connect	307
■ Chapter 7: AppFabric: Access Control Service	327
■ Chapter 8: AppFabric Service Bus	381
■ Chapter 9: AppFabric: Caching	485
■ Chapter 10: SQL Azure	497
■ Index	561

Windows Azure Platform Overview

In the past couple of years, cloud computing has emerged as a disruptive force in the information technology (IT) industry. Its impact is of the same magnitude as the Internet and offshoring. Gartner Research has identified cloud computing as one of the “top 10 disruptive technologies 2008–2012.” According to Gartner, a *disruptive technology* is one that causes major change in the accepted way of doing things. For developers, architects, and operations, cloud computing has caused a major shift in the way of architecting, developing, deploying, and maintaining software services.

Cloud computing democratizes IT, similar to how the Internet democratized the consumer industry. The Internet opened up a vast ocean of accessible resources to consumers, ranging from free advertising-based searching to online banking. Cloud computing is bringing similar trends to businesses small and big. Businesses can now reap the benefits of agility by simply deploying their software in someone else’s datacenter for a consumption fee. Hardware costs are out of the equation because of cloud service providers. These may sound like the hosting companies you already host your web sites on, but the big difference is that this is now a utility model built on highly scalable datacenter platforms. The cloud-computing wave is powerful enough for a company like Microsoft to start disrupting its own business model to invest in the opportunity.

In this chapter, I will cover some of the basics of cloud services and then jump into an overview of the Microsoft’s Windows Azure platform. In the previous edition of this book, I introduced the development models of some of the cloud service providers in the market. I took that approach because the technology was new and I wanted readers to understand the differences in the offerings. In this edition, I do compare the cloud services providers, but not with the same detail that I did in the previous edition. The public literature about these platforms has matured enough for it to be eliminated from the book.

Introducing Cloud Services

As an introduction to our discussion, consider a typical scenario in today’s medium to large enterprises. Assume a business unit has an immediate need to deploy a highly interactive niche web application (a micro-site) for a new product that will be released in five months. The application will provide the consumers a detailed view of the product and also the ability to customize and order the product right from the web site. The business unit has the budget but not the time and resources to implement it, and this deployment needs to happen in the next three months for it to be ready for the launch.

The IT hosting team understands the requirement, but to deploy an application with IT resources requires coordination among hardware, software, operations, and support teams. Perhaps ordering hardware and preparing the operating system build itself takes two months. After that, IT has to go

through its standard testing process and operations procedures to make sure all the operations and support needs are identified. So, the earliest application delivery date would be in six months.

The business owner escalates the urgency of the issue, but cannot get past the process boundaries of the enterprise. Ultimately, the business owner outsources the project to a vendor and delivers the application in three months. Even though the application is delivered, it doesn't have the desired enterprise support and operations quality. It doesn't have to go this way—the company IT department should be the preferred and one-stop shop for all the business' needs. Even though outsourcing gives you a great return on investment, in the long run you lose significantly on the innovation. Your own IT department has the ability to innovate for you, and its members should be empowered to do so instead of forced to overcome artificial process boundaries.

I see such scenarios on a daily basis, and I don't see a clear solution to the problem unless the entire process and structure in which these organizations operate is revamped, or unless technology like cloud computing is embraced wholeheartedly.

How will cloud computing help? To understand, let's go back to the original business requirement: the business owner has an immediate need to deploy an application, and the time frame is within three months. Basically, what the business is looking for is IT agility, and if the application takes only one month to develop, then is it really worth wasting six months on coordination and acquisition of the hardware?

Cloud computing gives you an instant-on infrastructure for deploying your applications. The provisioning of the hardware, operating system, and the software is all automated and managed by the cloud service providers.

Industry Terminology

For standardizing the overall terminology around cloud computing, the industry has defined three main cloud service categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

IaaS is a utility service that provides hardware and virtualized operating systems running in massively scalable data centers of the cloud service provider. You can then rent this infrastructure to deploy your own software and manage the lifecycle of not only your software applications but also the underlying operating systems. In IaaS, you are still responsible for upgrading, patching, and maintaining the operating systems and the software applications that run on the rented hardware. Therefore, the target audiences for IaaS are system administrators and operations engineers. In short, IaaS abstracts the hardware and virtualization infrastructure from you.

PaaS is a utility service that provides the hardware, operating systems, and the runtime environment for running your applications in massively scalable data centers of the cloud service provider. The PaaS manages the operating systems and hardware maintenance for you, but you have to manage your applications and data. Therefore, naturally, the target audience for PaaS is typically developers. Even though the final deployment and maintenance will be managed by the operations teams, the platform empowers developers to make certain deployment decisions through configurations. In short, PaaS abstracts the infrastructure and the operating system from you.

SaaS is a utility service that provides you an end-to-end software application as a service. You only have to manage your business data that resides and flows through the software service. The hardware, operating systems, and the software is managed by the SaaS provider for you. Therefore, the target audience for SaaS is typically business owners who can go to the SaaS web site, sign-up for the service, and start using it.

In its natural progression, a SaaS is built on a PaaS and a PaaS is built on an IaaS. Therefore, PaaS providers have capabilities of IaaS built into PaaS. Whether to offer it as a separate service is mostly a strategic decision. Figure 1-1 illustrates the typical management boundaries between IaaS, PaaS and SaaS.

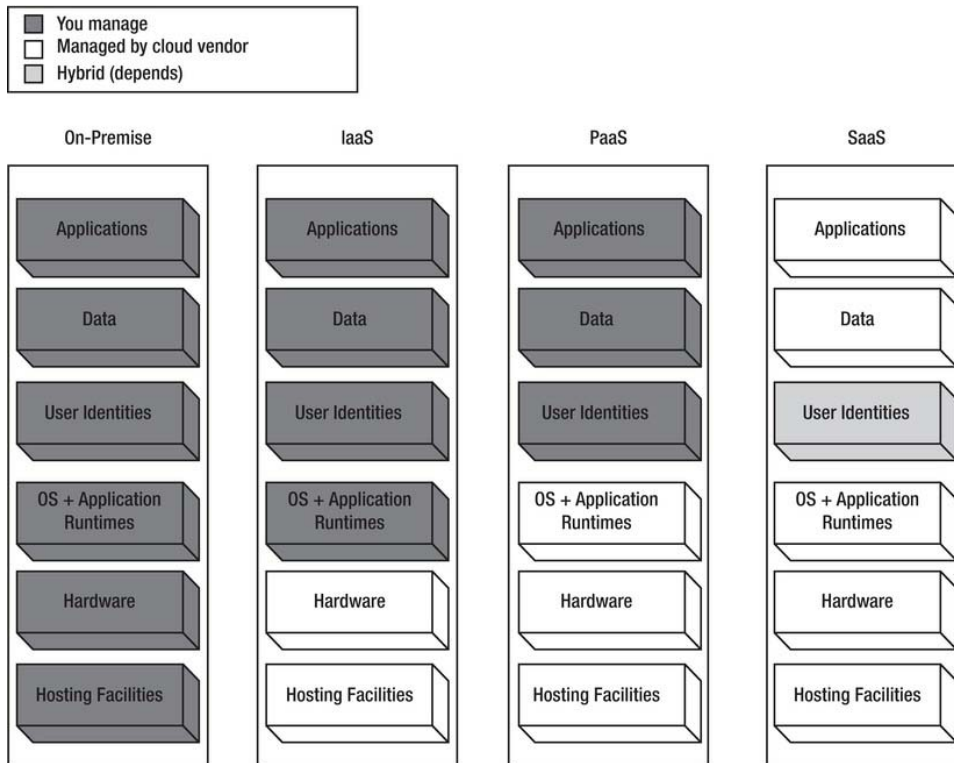


Figure 1-1. IaaS, PaaS, and SaaS Management Boundaries

The management of the user identities differs in different scenarios. Some enterprises will expose their on-premises identity provider as a federated service, while some businesses will keep SaaS and on-premises identities separate. Some additional terms that have been floating around in the past couple of years are Data as a Service (DaaS), IT as a Service, Security as a Service, and more. In the interest of simplicity, I have categorized all the services into IaaS, PaaS, and SaaS.

Types of Clouds

Along with the types of cloud services, the industry also frequently talks about the types of clouds that exist in the marketplace. A cloud is the underlying data center architecture that powers cloud services. Then what is the difference between a hosting provider and a cloud service provider? Great question....

As per my experience and knowledge, I would define a cloud only if the data center architecture provides you with the following services:

- **Pay as you go service** – A cloud must provide you with a utility service model where you are charged only for the resources you use or by the number of users accessing the service. The price should go down or up dynamically based on your usage.

- **A self-service provisioning portal** – A cloud must provide you with a self-service portal for acquiring and releasing resources manually and programmatically.
- **Server hardware abstraction** – A cloud must relieve you from acquiring and/or maintaining any hardware resources required for an application to run.
- **Network hardware abstraction** – A cloud must relieve you from acquiring and/or maintaining any networking hardware resources required by your application.
- **Dynamic scalability** – A cloud must provide you with a manual and/or programmatic option for dynamically scaling your application up and down to match the demand.
- **High Availability Service Level Agreement (SLA)** – A cloud must clearly define an SLA with guaranteed availability of the platform.

The location of this cloud determines its type: private or public. In the interest of keeping the topic simple, I will define only these two types of clouds.

A public cloud is a data center that exists in the public domain and is accessible over the Internet. The public cloud is managed by the cloud service provider. Some public cloud platforms integrate with your company's intranet services through federation and virtual private networks or similar connectivity. The core application and the data still runs in the cloud service provider's data center.

A private cloud is cloud infrastructure running in your own datacenter. Because the definition of a cloud is wide open to interpretation, every company has carved out its own definition of a private cloud. I use the capabilities defined previously as bare minimum requirements for defining a public or a private cloud. If any of these services are not offered by a private cloud, then it's merely an optimized datacenter. And it is not necessarily a bad choice; an optimized datacenter may be a better choice than a cloud in some scenarios. The primary difference between private and public clouds is the amount of capital cost involved in provisioning infrastructure. Public clouds don't require provisioning.

■ **Note** Throughout this book, depending on the context of the conversation, I have used the terms *cloud services* and *cloud applications* interchangeably to generally represent cloud services. A cloud service may be thought of as a collection of cloud applications in some instances, but in the context of this book, both mean the same thing.

Defining Our Terms

Before diving deep into cloud services, I would like to introduce you to the terminology used in this book. To be consistent, I have developed this section for defining some important terms used in this book. Table 1-1 lists the terms and their definitions as they relate to this book.

Table 1-1. Terminology in This Book

Term	Definition
Azure or Windows Azure	Microsoft's Windows Azure platform
Cloud application	An application deployed to a cloud services platform and typically part of a larger cloud service
Cloud platform	A PaaS offering by a cloud service provider for deploying cloud services (e.g., Windows Azure platform offered by Microsoft)
On-premise	Refers to applications or services deployed and managed by an enterprise in its own datacenters
Off-premise	Refers to applications or services in the cloud
Solution	When used on its own, refers to a collection of applications and/or cloud services designed for a specific business purpose (e.g., a payroll solution consisting of three cloud services and four on-premise applications)

Cloud Service Providers

In the past couple of years, several large software and Internet platform companies have started offering cloud services. It was a natural transition for companies like Amazon, Google, and Microsoft who already had a large Internet presence. VMware has been building these capabilities through acquisitions like Springsource and Zimbra. The offerings from all the cloud services are fragmented and it can sometimes be difficult to get a grasp of all the service offerings just from a single vendor. In Table 1-2, I have listed a few providers with mature cloud services offerings. You can apply the same capabilities table to any cloud service provider present and future.

Table 1-2. Cloud Service Capabilities

Capability	IaaS	PaaS	SaaS
Public	Amazon EC2 Windows Rackspace.com	Windows Azure platform Windows Azure AppFabric Force.com Google AppEngine	Office 365 Salesforce.com Google Apps

Private	VMWare vSphere Hyper-V	Windows Azure Appliance (Not yet available)	SharePoint as an IT service
---------	---------------------------	---	--------------------------------

From Table 1-2, you will be able to qualify the cloud service providers that fit your specific needs. Typically, you will not find any single cloud service provider that satisfies all your needs, which is true even with on-premises software.

Shifting to the Cloud Paradigm

As seen in the previous section, the choices provided by these offerings can put you in a dilemma, and most probably you will end up testing at least two cloud services before deciding on one. The move from a traditional on-premise model to an off-premise cloud model is a fundamental paradigm shift for businesses. Usually businesses are in their comfort zone when managing IT internally. With the cloud services model, even though the cost savings become evident, the challenge for businesses is to get out of their comfort zones and make the paradigm shift of moving to cloud services to stay competitive. The shift does not happen overnight; it takes several months of rigorous analysis, planning, and implementation. Depending on the costs, benefits, risks, and security requirements, a business can stay on-premise, embrace cloud services fully, or settle on a hybrid model yielding cost benefits while keeping core competencies on-site. Figure 1-2 illustrates the ownership of key enterprise assets in on-premise, cloud, and hybrid scenarios.

The recommended migration process is to move step by step, one application at a time. When the offshore software development model became popular in 2000, businesses faced a similar challenge in getting aboard the outsourcing wagon. Now, many businesses have significant offshore investments and clearly see the payoffs. It took time and learning for businesses to make the paradigm shift in offshore software development projects. For cloud services to succeed, businesses will be required to make a paradigm shift again.

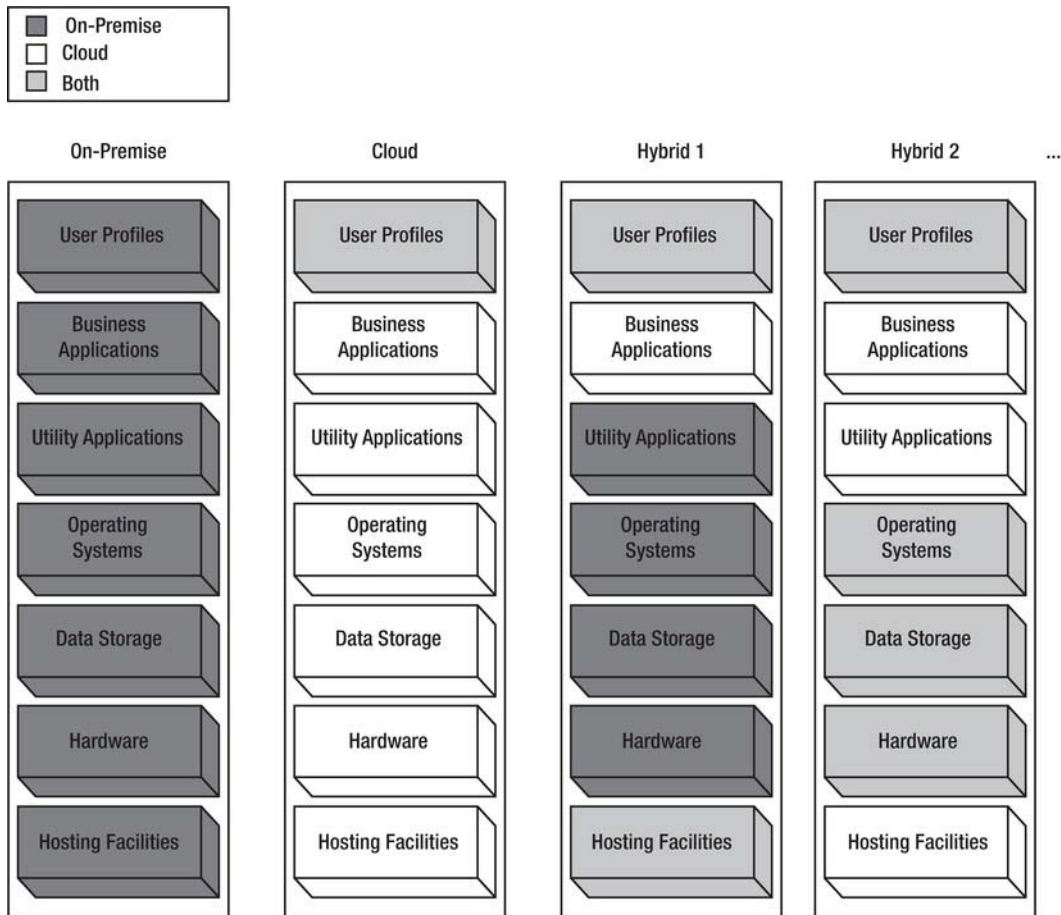


Figure 1-2. On-premise, cloud, and hybrid scenarios

In Figure 1-2, the on-premise and cloud scenarios are fairly easy to understand, because either all the assets are on-premise or in the cloud. The user profiles asset is usually required on both sides because of single sign-on requirements between on-premise and cloud services. In hybrid models, the businesses and the service provider must negotiate and decide which assets and services are better suited for locations on-premise, in cloud, or both. In the Hybrid 1 scenario, the user profiles and hosting facilities are present on both the sides; the business applications are in the cloud, whereas the utility applications, operating systems, data storage, and hardware are on-premise. In the Hybrid 2 scenario, the user profiles, operating systems, data storage, and hardware are present on both sides, whereas the business applications, utility applications, and hosting facilities are in the cloud. Most companies typically choose some hybrid model that best suits them.

Understanding the Cloud Services Ecosystem

The cloud services ecosystem consists of five major roles, as shown in Figure 1-3.

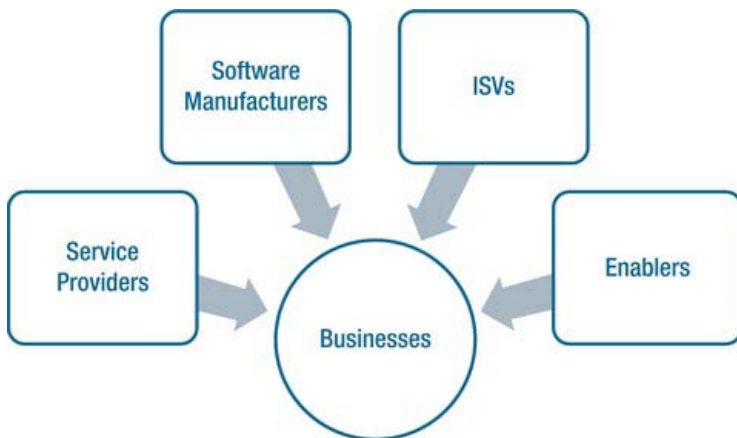


Figure 1-3. The cloud services ecosystem

Service Providers

The service providers are the companies that provide cloud services to the businesses and consumers. These companies run giant data centers hosting massively virtualized and redundant software and hardware systems. Service providers like Amazon, with its EC2 service, and Microsoft, with its Windows Azure platform, fall into this category. These companies not only have expertise in data center management, but also in scalable software management. The service providers may offer services directly to the businesses, consumers, or ISVs.

Software Vendors

Software designed to run on-premise is very different from software designed for cloud. Even though they both may provide the same business functionality to the end users, architecturally they may differ. The cloud services must account for multi-tenancy, scalability, reliability and performance at a much broader scale than on-premise architecture. Cloud services run in data centers offered by cloud service providers. In some cases, there is a significant overlap between the service providers and the software vendors. For example, Microsoft Windows Azure platform, Microsoft's Office 365, and Google Apps are cloud software running in their own datacenters. The software vendors have found it economically feasible to package hardware and software together in the datacenters to optimize software delivery via cloud.

Independent Software Vendors

Independent software vendors (ISVs) are going to play a key role in the success of cloud services because of their expertise in vertical business applications. ISVs typically build vertical applications on an already existing platform. ISVs identify the business demand for a particular solution in vertical markets and

thrive by offering the solution on existing platforms. The cloud offers a great platform for the ISVs to build vertical solutions. For example, an ISV could build a medical billing solution in the cloud and offer the service to multiple doctors and hospitals. The infrastructure required for building multitenant scalable software is already provided by the service providers, so the ISVs have to focus only on building the business solution and can enable them to penetrate new markets with lightning speed.

Enablers

Enablers (which are also called *implementers or system integrators*) are vendors offering services to build end-to-end solutions by integrating software from multiple vendors. Many enterprises purchase software licenses from vendors but never deploy the software because of lack of strategic initiative or availability of product expertise. Enablers fill in the gap by offering consulting services for the purchased software. Organizations like Microsoft Consulting Services and IBM Global Services offer customer-specific services regardless of the underlying platform. Enablers play a key role by integrating on-premise and cloud services or building end-to-end cloud services customized for a business. Cloud platform offers enablers an opportunity to expand their service offerings beyond on-premise solutions.

Businesses

Finally, businesses drive the demand for software products and services. If businesses see value or cost savings in a particular solution, they do not hesitate to implement it. To stay competitive in today's market, businesses have to keep their IT and applications portfolios up to date and take advantage of economies of scale wherever possible. Cloud service offerings are architected to achieve economies of scale by supporting multiple businesses on a scalable and automated platform. For cloud service offerings to be successful, service providers, software vendors, ISVs, and enablers must work together in creating cloud applications and services not only providing cost savings but also a competitive edge to businesses and in-turn to consumers through these businesses.

Microsoft's Cloud Strategy

For building a successful cloud services business, a company needs to first invest in building globally distributed datacenters that are highly automated, efficient, and well connected. Building such datacenters requires significant investment and support from software and operating system business partners to help monetize them. Therefore, typically, you will only see very large companies like Microsoft and Amazon offer such services at a global scale.

Microsoft is the largest software manufacturer in the world and its Global Foundation Services (GFS) group has done a phenomenal job in building a global network of datacenters that can be leveraged by software partners within the company for delivering software services. This network of Microsoft datacenters is termed as Microsoft's cloud. I have toured one of these datacenters and I think they are one of the most advanced in the world. What follows is a list of the 10 largest datacenters in the world. Four in the top 10 belong to Microsoft (I have highlighted them in bold).¹

1. 350 East Cermak / Lakeside Technology Center (Digital Realty)
2. Metro Technology Center, Atlanta (Quality Technology)

¹ www.datacenterknowledge.com/special-report-the-worlds-largest-data-centers/

3. The NAP of the Americas, Miami (Terremark)
4. NGD Europe, Newport Wales (Next Generation Data)
5. Container Data Center, Chicago (Microsoft)
6. Microsoft Dublin (Microsoft)
7. Phoenix ONE, Phoenix (i/o Datacenters)
8. CH1, Elk Grove Village, Ill. (DuPont Fabros)
9. 9A and 9B. Microsoft Datacenters in Quincy Washington and San Antonio
10. The SuperNAP, Las Vegas (Switch Communications)

As the adoption of cloud services increase in the IT industry, this list will change over time.

■ **Note** I highly recommend you visit www.globalfoundationservices.com for more information on Microsoft's datacenters and GFS.

Microsoft's cloud strategy consists of the following four main initiatives:

1. Build a network of highly available datacenters around the world as a software platform of the future.
2. Leverage these datacenters for delivering its PaaS offerings.
3. Leverage these datacenters for delivering its SaaS offerings.
4. Leverage the partner network for delivering IaaS offerings.

PaaS and SaaS offerings will be the primary mode of delivering most of its software assets in the future. In the past two years, Microsoft has successfully positioned Windows Azure platform as a leading public cloud platform. Microsoft went an extra mile innovating PaaS beyond traditional IaaS. Today, there are several Fortune 500 and Small & Medium businesses actively using the Windows Azure platform. Ultimately Microsoft will need to provide an IaaS offering just to ease the on-boarding process for enterprises. The current on-boarding is not as attractive for Enterprise customers because of investment required in migrating legacy applications to Windows Azure platform.

Windows Azure Platform Overview

In 2008 during the Professional Developer's Conference (PDC), Microsoft announced its official entry into the PaaS arena with the Windows Azure platform. Even though the SaaS offering called Business Productivity Online Suite (BPOS) or Office 365 has been around for a few years, the Windows Azure platform is an attempt to create a complete PaaS offering. See Figure 1-4.

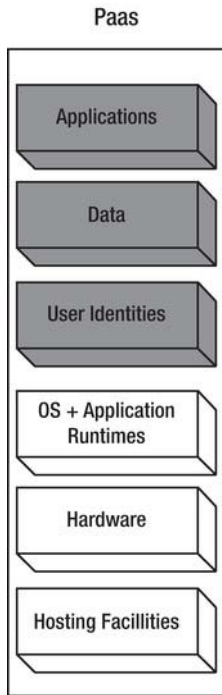


Figure 1-4. Platform as a service

Windows Azure platform is a key component of Microsoft's cloud strategy. The Windows Azure platform is a paradigm shift where unlimited resources are available at your fingertips for developing and deploying any .NET application in a matter of minutes. It disrupts your current ways of process-oriented sequential thinking. Microsoft has designed Windows Azure as an operating system for the data center. You don't have to wait for provisioning of any server or networking hardware, and then the operating system for deploying your application. Windows Azure platform drastically reduces the time from idea to production by completely eliminating the hardware and operating systems provisioning steps.

Windows Azure platform is a collection of building blocks for cloud services. Microsoft has been in the cloud business for quite some time with its consumer cloud services like MSN, Xbox Live, and Hotmail. Microsoft has also rebranded its business productivity and collaboration suite as Office 365 that includes services like SharePoint Online, Exchange Online, and Conferencing Services (Microsoft Office Lync). Windows Azure platform consists of three core components: Windows Azure, SQLAzure, and Windows Azure AppFabric, as shown in Figure 1-5.

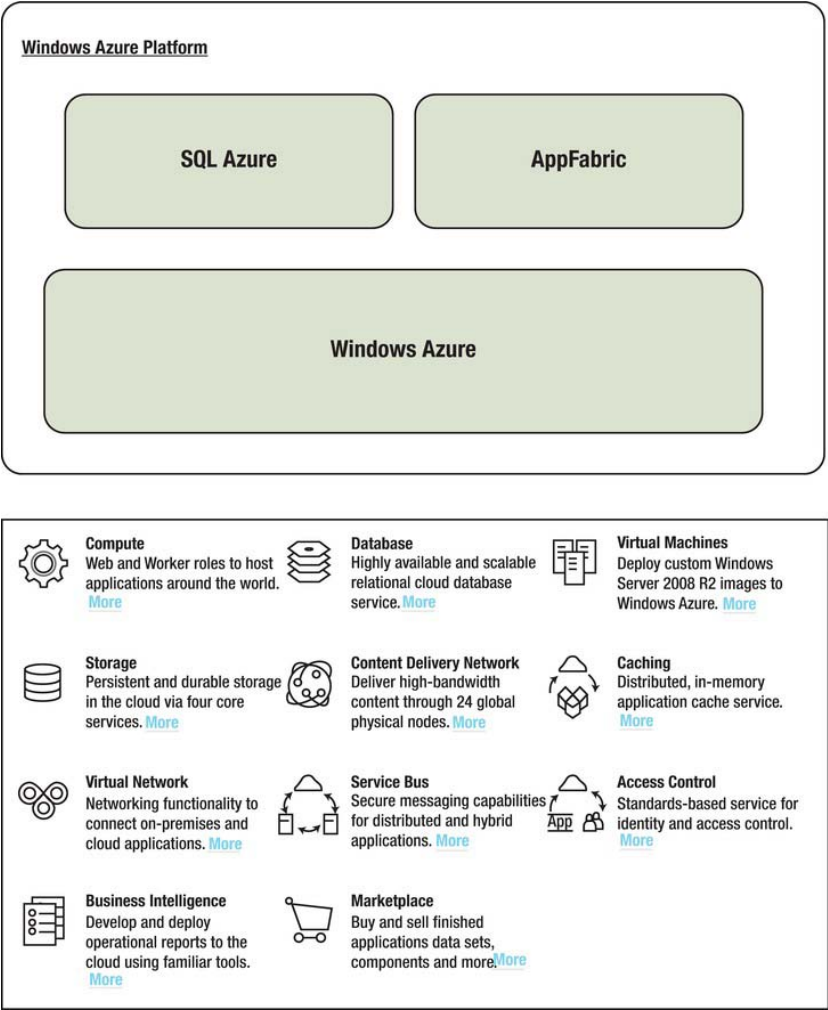


Figure 1-5. Microsoft Windows Azure Platform (Source: www.microsoft.com/windowsazure/features/)

The core components are then split into several sub-components. But, the overall idea is to provide an a-la-carte menu so you can use any individual component in your solution. The pricing is typically based on each sub-component usage. The feature set is updated every six months, and there will be more features released between writing and publishing of this book.

■ **Note** Figure 1-5 only addresses the core components of the Windows Azure platform, as the platform matures; Microsoft is slowly promoting some sub-categories into its own offerings like the Virtual Network category that

includes Windows Azure Connect and Traffic Manager. The Marketplace feature currently consists of the Windows Azure Marketplace DataMarket where you can publish and monetize your data over standard interfaces. Covering DataMarket goes beyond the scope of this book.

Windows Azure is the operating system for the datacenter that provides compute, storage, and management services. SQL Azure is a relational database engine in the Windows Azure Platform. Windows Azure AppFabric is the middleware component that consists of services like Service Bus, Access Control, and Caching Services. Developers can either build services that span across all these components or pick and choose the components as needed by the service architecture. The overall concept of Windows Azure platform is to offer developers the flexibility to plug in to the cloud environment as per the architectural requirements of the service.

In this book, I have covered all the three main components and their sub-components. Software development today typically consists one or more of the following types of applications:

- **Rich client and Internet applications** – Examples are Windows Client, Windows Presentation Foundation, HTML5, and Silverlight.
- **Web services and web applications** – Examples are ASP.NET, ASP.NET Web Services, and Windows Communications Foundation.
- **Server applications** – Examples are Windows Services, WCF, middleware, message queuing, and database development.
- **Mobile application** – Examples are .NET Compact Framework and mobile device applications.

The Windows Azure platform provides you with development tools and the deployment platform for developing and deploying all of these types of applications.

■ **Note** It took me more time to write this chapter than it did to develop and deploy a high-scale compute application on Windows Azure platform. Traditionally, the application would have taken 6–12 months to develop and deploy in production.

Understanding Windows Azure Compute Architecture

Fundamentally, Windows Azure platform compute architecture is based on a software fabric controller running in the data center and defining clear abstraction between server hardware and operating systems. The fabric controller automates the deployment of virtualized operating systems images on server hardware. In its simplest form, a typical cloud data center consists of a bank of server hardware and massive storage for storing fully functional operating system images. The fabric controller manages the life cycle of the deployment by allocating and decommissioning hardware and operating system images as needed. As a user, when you deploy your service to the cloud, the fabric controller provisions the hardware servers, deploys operating system image on those servers, provisions appropriate

networking software like routers and load-balancers, and deploys your service to those servers. Once the service is deployed on servers, it is ready to be consumed. The numbers of service instances are configured by the service owner and would typically depend on the demand and high availability requirements of the service. Over the life cycle of the instance, the fabric controller is responsible for automating the maintenance, security, operations and high availability of the instances. Figure 1-6 illustrates the Windows Azure platform compute architecture.

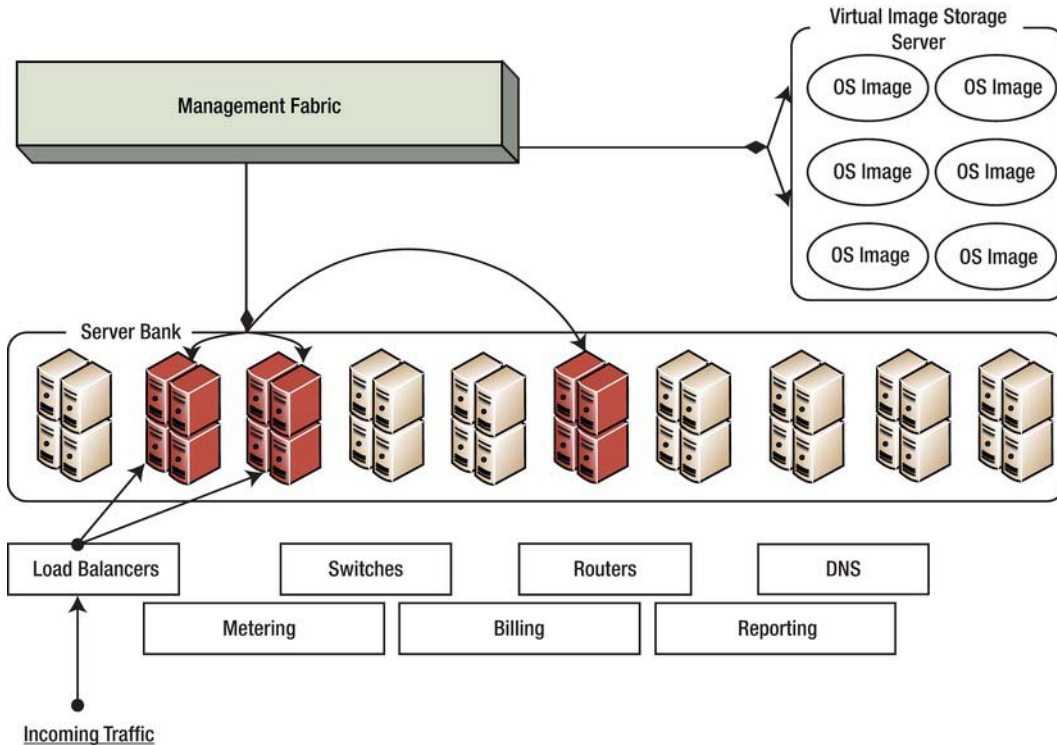


Figure 1-6. Windows Azure platform compute architecture

The architecture also consists of some fixed assets like switches, routers, and DNS servers that manage the workload distribution across multiple service instances. This architecture is componentized and deployed into several geographically dispersed datacenters for providing geo-located services. The metering, billing and reporting components complement the infrastructure with the ability to measure and report the usage of the service per customer.

■ **Note** Windows Azure platform is hosted in six datacenters around the world. North Central US (1), South Central US (1), Europe (2), Asia (2). From the administration portal, you have the ability to choose the datacenter location for your application. Microsoft gives regular tours of these datacenters to enterprises. You will have to

poke your management to get a tour of one of these world-class datacenters. You can find more information of these datacenters at www.globalfoundationservices.com.

Windows Azure

Windows Azure is the core operating system of the platform that provides all the necessary features for hosting your services in the cloud. It provides a runtime environment that includes the IIS web server, background services, storage, queues, management services, and load-balancers. Windows Azure also provides developers with a local development fabric for building and testing services before they are deployed to Windows Azure in the cloud. Windows Azure also integrates seamlessly with the Visual Studio 2010 development environment by providing you with service publishing and IntelliTrace features. Figure 1-7 illustrates the three core services of Windows Azure.

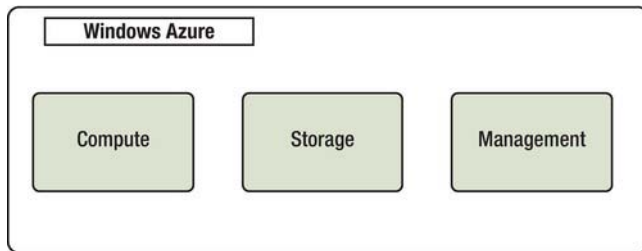


Figure 1-7: Windows Azure core services

Following is a brief description of the three core services of Windows:

Compute – The compute service offers scalable hosting of services on 64-bit Windows Server 2008 R2. The platform is virtualized and designed to scale dynamically based on demand. The platform runs Internet Information Server (IIS) version 7 enabled for ASP.NET Web applications. From version 1.3 of the SDK, you have access to Full IIS and administration features. You can also script start-up tasks that require administration privileges like writing to a registry or installing a COM dll library or installing third-party components like Java Virtual machines. Developers can write managed and unmanaged services for hosting in the Windows Azure Compute without worrying about the underlying operating systems infrastructure.

Storage – There are three types of storage supported in Windows Azure: tables, blobs, and queues. These storage types support direct access through REST APIs. Windows Azure tables are not relational database like SQL Server tables. Instead, they provide structured data storage capabilities. They have an independent data model popularly known as the entity model. Tables are designed for storing terabytes of small-sized highly available data objects. For example, user profiles in a high-volume ecommerce site would be a good candidate for tables. Windows Azure blobs are designed to store large sets of binary data like videos, images, and music in the cloud. Windows Azure queues are the asynchronous communication channels for connecting between services and applications not only in Windows Azure but also from on-premises

applications. Queues are also the recommended method of communication between multiple Windows Azure role instances. The queue infrastructure is designed to support unlimited number of messages, but the maximum size of each message cannot exceed 8KB. Any account with access to storage can access tables, blobs, and queues. The total storage capacity of one storage account is 100TB and you can have multiple storage accounts. Windows Azure Drives provides NTFS drive volumes for Windows Azure applications in the cloud. So, you can create a drive, upload it to the blob storage and then attach it as an external drive to the windows azure instances. Drives provide you with durable storage access natively within your role but at the same time you lose the broad distributed nature of storing blobs that are scaled-out across multiple storage servers. A drive is still a single blob from Windows Azure Blob storage perspective.

Management – The management service supports automated infrastructure and service management capabilities to Windows Azure cloud services. These capabilities include automatic commissioning of virtual machines and deploying services in them, as well as configuring switches, access routers, and load balancers for maintaining the user defined state of the service. The management service consists of a fabric controller responsible for maintaining the health of the service. The fabric controller supports dynamic upgrade of services without incurring any downtime or degradation. Windows Azure management service also supports custom logging and tracing and service usage monitoring. You can interact with the management service using a secure REST-API. Most of the functionality available on the Windows Azure platform portal is available through the service management API for programmatically executing tasks. The API is widely used for automating the provisioning and dynamic scaling tasks. For example, the publish feature in the Visual Studio and the Windows Azure PowerShell cmdlets use the service management API behind the scenes to deploy cloud services to Windows Azure.

Figure 1-8 illustrates the Windows Azure architecture from available service perspective.

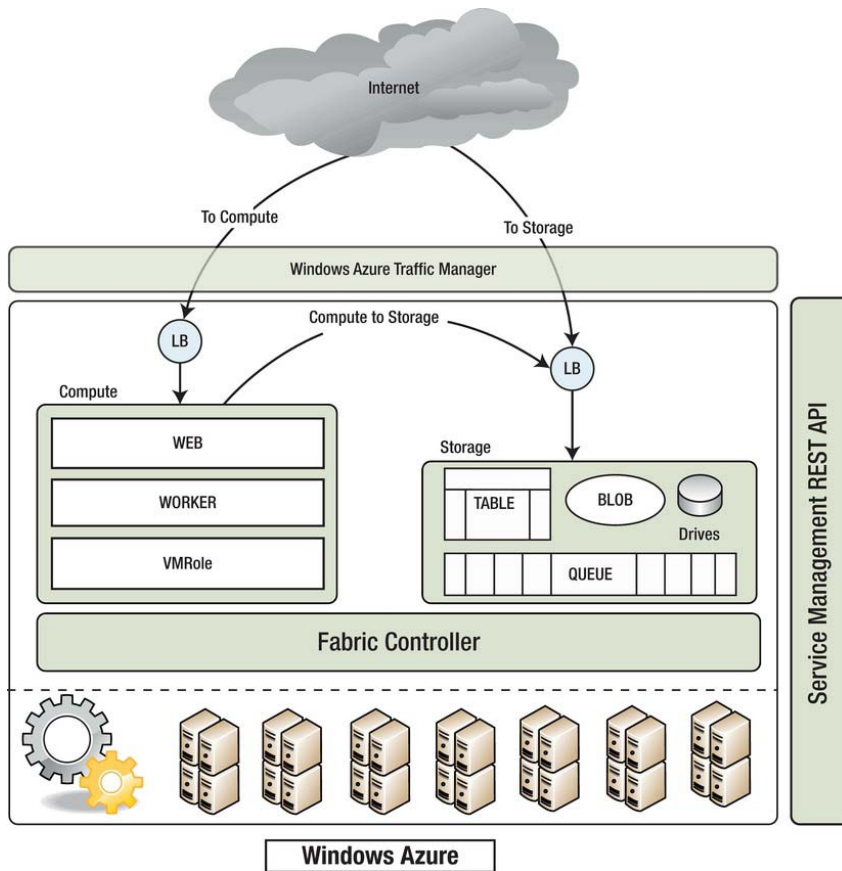


Figure 1-8. Windows Azure

When a request from the Internet comes in for your application, it passes through the load balancer and then to the specific Web/Worker role instance running your application. If a request for a Storage service comes in, it passes through the load balancer to the appropriate Storage service component.

Windows Azure Traffic Manager is a cross-region (datacenter) high-availability service that allows you to configure rules-based traffic diversions to your applications running in multiple Windows Azure datacenters. Traffic Manager should be a key component in your disaster/recovery and business continuity strategy. Even though Traffic Manager manages traffic diversions, it does not replicate data across multiple datacenters; you have to replicate data across multiple datacenters for maintaining data consistency. At the time of writing, Traffic Manager was not available in production release. In the current version, there were three traffic diversion rules available: Fault-tolerance, Performance-based, and Round-Robin. In the CTP version, you could try it out for free.

Compute

The Windows Azure Compute service is based on a role-based design. To implement a service in Windows Azure, you have to implement one or more roles supported by the service. The current version of Windows Azure supports three roles: Web Role, Worker Role, and VM Role. A role defines specific behavior for virtual machine instances running in the cloud. A web role is used for deploying web sites, a worker role is used for deploying background services or middle tier applications, and a VM Role is typically used for running applications that do not fit a web or a worker role. Applications with intrusive installation process are well suited for VM Role. Architecture of the system should dictate the types of roles you will need in your application.

The abstraction between the roles and the hardware is managed by the Fabric Controller. Fabric Controller manages end-to-end automation of the role instances, from hardware provisioning to maintaining service availability. Fabric Controller reads the configuration information you provide for your services and adjusts the deployment profile accordingly, as shown in Figure 1-9.

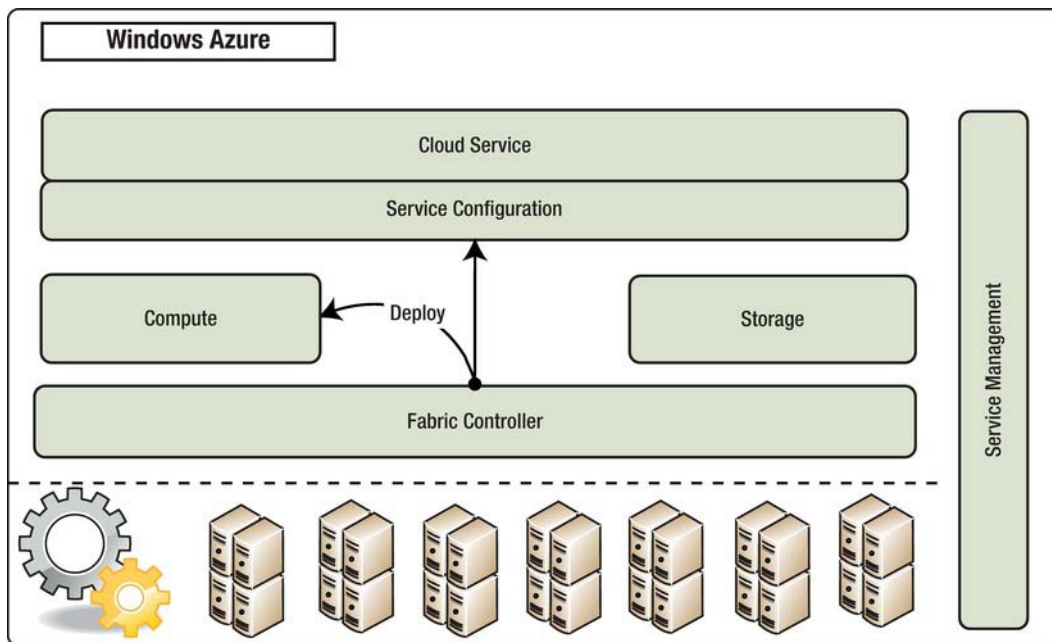


Figure 1-9. Fabric Controller deploys application

In the service configuration, you have to specify how many instances of a particular role you want to start with. The provisioning portal and the service management API can give you the status of the deployment. Once the cloud service is deployed, it is managed entirely by Windows Azure. You only manage your application and data.

Web Role

A Web role gives you the ability to deploy a web site or web service that can run in an IIS 7 environment. Most commonly, it will be an ASP.NET web application or external facing Windows Communications Foundation (WCF) service endpoints. Even though a Web Role can host multiple sites, it is assigned only one external endpoint or entry point. But, you can configure different ports on the same entry point for http, https and custom TCP connections.

■ **Note** The UDP protocol is not supported at this time in Windows Azure services.

The Web role also supports FastCGI extension module to IIS 7.0. This allows developers to develop web applications in interpreted languages like PHP and native languages like C++. Windows Azure supports Full Trust execution that enables you to run FastCGI web applications in Windows Azure Web role. To run FastCGI applications, you have to set the `enableNativeCodeExecution` attribute of the Web role to true in the `ServiceDefinition.csdef` file. In support of FastCGI in the Web role, Windows Azure also introduces a new configuration file called `Web.roleconfig`. This file should exist in the root of the web project and should contain a reference to the FastCGI hosting application, like `php.exe`.

In the interest of keeping this book conceptual, I will not be covering FastCGI applications. For more information on enabling FastCGI applications in Windows Azure, please visit the Windows Azure SDK site at <http://msdn.microsoft.com/en-us/library/dd573345.aspx>.

■ **Caution** Even though Windows Azure supports native code execution, the code still runs in the user context, not administrator, so some WIN32 APIs that require system administrator privileges will not be accessible by default but can be configured using the startup tasks and elevated privileges. I will cover start up tasks in detail in the next chapter.

Can I Run Existing Java Server Applications in Windows Azure?

There are a lot of enterprises that run Java and migrating these applications to Windows Azure is a big opportunity for Microsoft in winning and sustaining the underlying platform. Behind the scenes, Windows Azure runs Windows Server operating systems that can run Java virtual machines. Therefore, you can write custom scripts to install Java virtual machines on the compute instances and then run your Java application on those instances. I have covered writing custom start-up tasks in Azure in a bit more detail in the next chapter, but the short answer is yes, you can run Java server applications on Windows Azure, but it's still not the first class citizen due to tools and endpoint limitations.

Worker Role

The Worker role gives you the ability to run a continuous background process in the cloud. It is analogous to Windows Services in the Windows platform. Technically, the only major difference between a Web Role and a Worker Role is the presence of IIS on the Web Role. The Worker role can

expose internal and external endpoints and also call external interfaces. A Worker role can also communicate with the queue, blob, and table storage services. A Worker role instance runs in a separate virtual machine from a Web role instance, even though both of them may be part of the same cloud service application. In some Windows Azure services, you may require communication between a Web role and a Worker role. Even though the Web and Worker role expose endpoints for communication among roles, the recommended mode of reliable communication is Windows Azure queues. Web and Worker roles both can access Windows Azure queues for communicating runtime messages. I have covered Windows Azure queues later in the book.

A Worker role class must inherit from the `Microsoft.WindowsAzure.ServiceRuntime.RoleEntryPoint` class. `RoleEntryPoint` is an abstract class that defines functions for initializing, starting and stopping the Worker role service. A Worker role can stop either when it is redeployed to another server, or you have executed the Stop action from the Windows Azure developer portal. Figure 1-10 illustrates the sequence diagram for the life cycle of a Worker role.

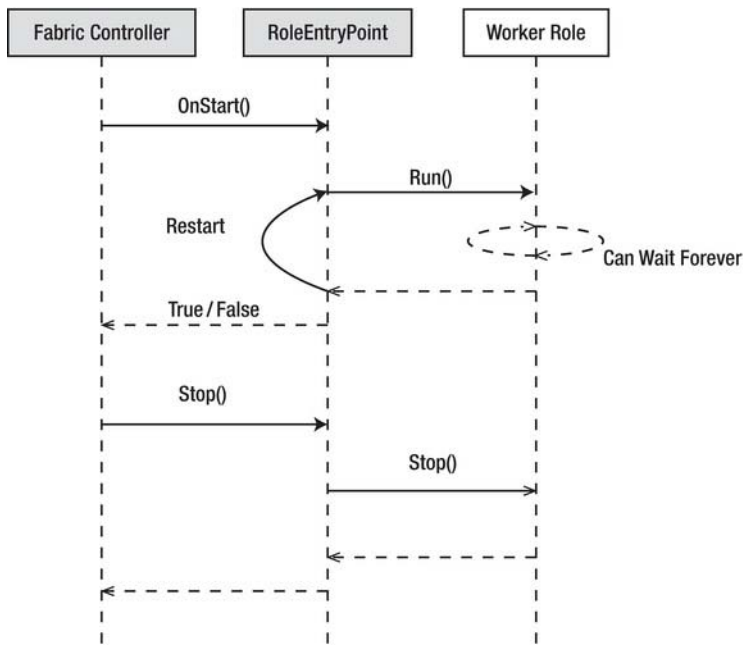


Figure 1-10. Sequence diagram for a Worker role service

In Figure 1-10, there are three objects: Fabric Controller, `RoleEntryPoint`, and a Worker role implementation of your code. Fabric Controller is a conceptual object; it represents the calls that the Windows Azure Fabric Controller makes to a Worker role application. The Fabric Controller calls the `Initialize()` method on the `RoleEntryPoint` object. `RoleEntryPoint` is an abstract class so it does not have its own instance; it is inherited by the Worker role instance to receive calls. The `OnStart()` method is a virtual method, so it does not need to be implemented in the Worker role class. Typically, you would write initialization code like starting diagnostics service or subscribing to role events in this method. The Worker role starts its application logic in the `Run()` method. The `Run()` method should have a continuous loop for continuous operation. If the `Run()` method returns, the role is restarted by the `OnStart()` method. If the role is able to start successfully, the `OnStart()` method returns `True` to the Fabric Controller; otherwise, it returns `False`. The Fabric Controller calls the `Stop()` method to shut down the

role when the role is redeployed to another server or you have executed a Stop action from the Windows Azure developer portal.

VM Role

The VM role is specifically designed by Microsoft to reduce the barriers to entry into the Windows Azure Platform. VM Role lets you customize a Windows Server 2008 R2 Enterprise virtual machine, based on a virtual hard drive (VHD), and then deploy it as a base image in Windows Azure. Typical scenarios for using VM role are as follows:

- If you want to install software in your farm that does not support silent installation
- If the installation of a specific software component required manual intervention
- If you want to install a third-party application that needs significant modifications for deploying to the web or worker roles
- Any application that does not fit in web role or a worker role model
- Quick functional testing for specific software components that may later need to run on-premises

VM role gives you more control over the software you can install on the virtual machine before uploading and running in Windows Azure. Therefore, when creating a VM role, you have for first create a VHD on your local machine, install the appropriate software on it, and then upload the operating system image to Windows Azure. Once the operating system image is uploaded, you can then create a service definition for the service and then deploy multiple instances of the operating system image adhering to your service definition. In a Web role and a Worker role, the underlying operating system image is provided to you by Windows Azure, but in the case of VM role, the service runs in the operating system image you created. Figure 1-11 illustrates the high-level process for creating a VM role image in Windows Azure. I have covered the entire process of creating and running VM Roles in detail later in the book.

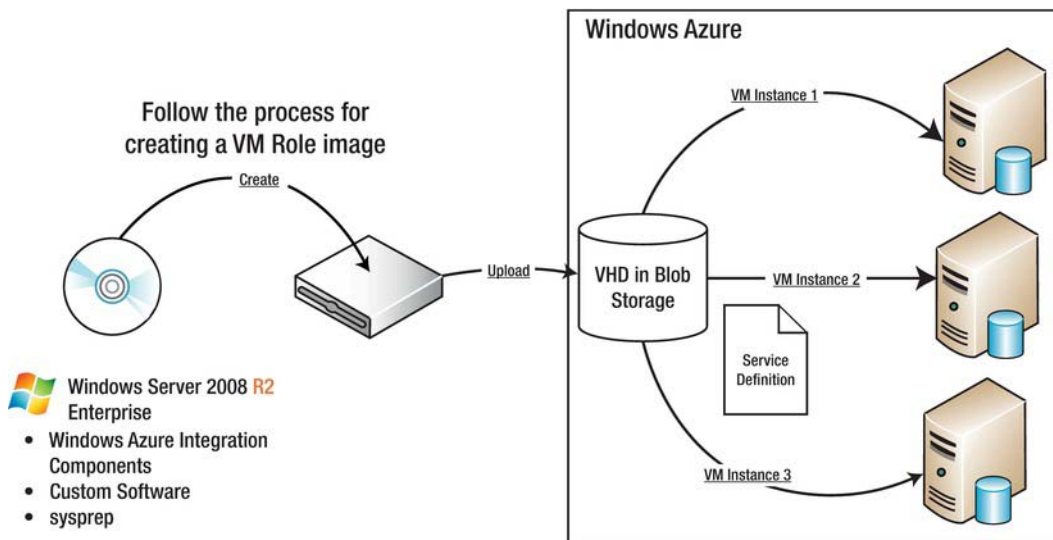


Figure 1-11. Windows Azure VM Role

■ **Tip** Don't use VM role unless absolutely needed, because by acquiring more control over the underlying operating system image, you also inherit the risks and the burden associated with maintaining it. Windows Azure does not understand the health of your applications running on a VM role, and therefore it becomes your responsibility to track application health. For any Windows Azure application architecture, the Web role and the Worker role models must be preferred over VM role.

Windows Azure Connect

Windows Azure Connect (aka Project Sydney) provides you with secure network connectivity between your on-premises machines and Windows Azure role instances. Windows Azure Connect is a new feature launched with Windows Azure SDK 1.3. With Windows Azure Connect, you can host your applications in Windows Azure and connect back to your data or applications that run on-premises. The motivation behind this feature was reducing the barriers to entry into Windows Azure. There is some data in enterprises that cannot be moved to the cloud due to various reasons like regulatory compliance, legal holds, company policies or simply IT politics. With Windows Azure Connect, the argument is reduced to tradeoff between latency and benefits of Windows Azure.

In the current version, Windows Azure Connect allows you to create point-to-point connectivity between your Windows Azure role instances and on-premises machines. It also gives you the ability to domain join the Windows Azure instances to your domain. Figure 1-12 illustrates the high-level architecture of the Windows Azure Connect capabilities.

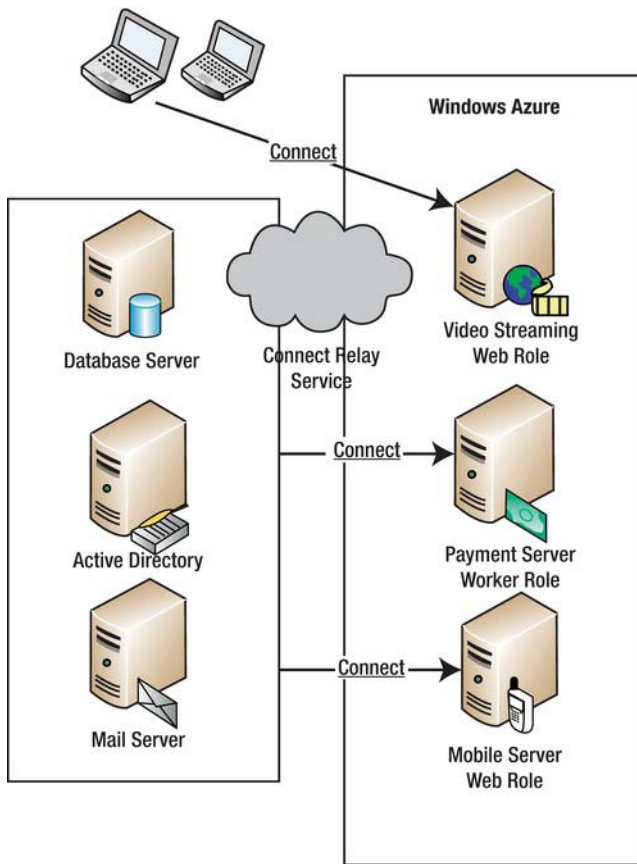


Figure 1-12. Windows Azure Connect

In Figure 1-12, the database server, the Active Directory server, and the mail server are grouped together for connecting with the Payment Server and Mobile Server Windows Azure role instances, whereas the development machines are grouped together for connecting with the Video Streaming role instance. The Windows Azure administration portal allows you to setup group-level as well as machine level connectivity. The connectivity between the on-premises and Windows Azure instances is secured end to end via IPsec. Windows Azure Connect also uses a cloud-based relay service for firewall and NAT traversal of the network traffic between your on-premises machines and Windows Azure role instances.

■ **Note** In the current version (1.3), you cannot use Windows Azure Connect to connect between your Windows Azure role instances because they are assumed to connect with each other. For such communication, you can use Windows Azure Queues, Input or Internal endpoints.

■ **Tip** Don't use Windows Azure Connect unless absolutely needed because as you move data away from the application, you will see performance degradation due to latency. If you have an interactive web application, it is better to move the data closer to the application in Windows Azure storage or SQLAzure. In some cases, you can separate the sensitive data and keep it on-premises and move rest of the data into the cloud and use Windows Azure Connect for calling on-premises applications

Windows Azure Storage

Storage service offers the compute nodes access to a scalable storage system. The storage service has built-in highly availability within a datacenter boundary. It maintains three copies of your data at any point in time behind the scenes. You can access the storage service from anywhere through a REST API. The open architecture of the Storage service lets you design your applications to store data using REST APIs. Figure 1-13 illustrates the Windows Azure storage service architecture.

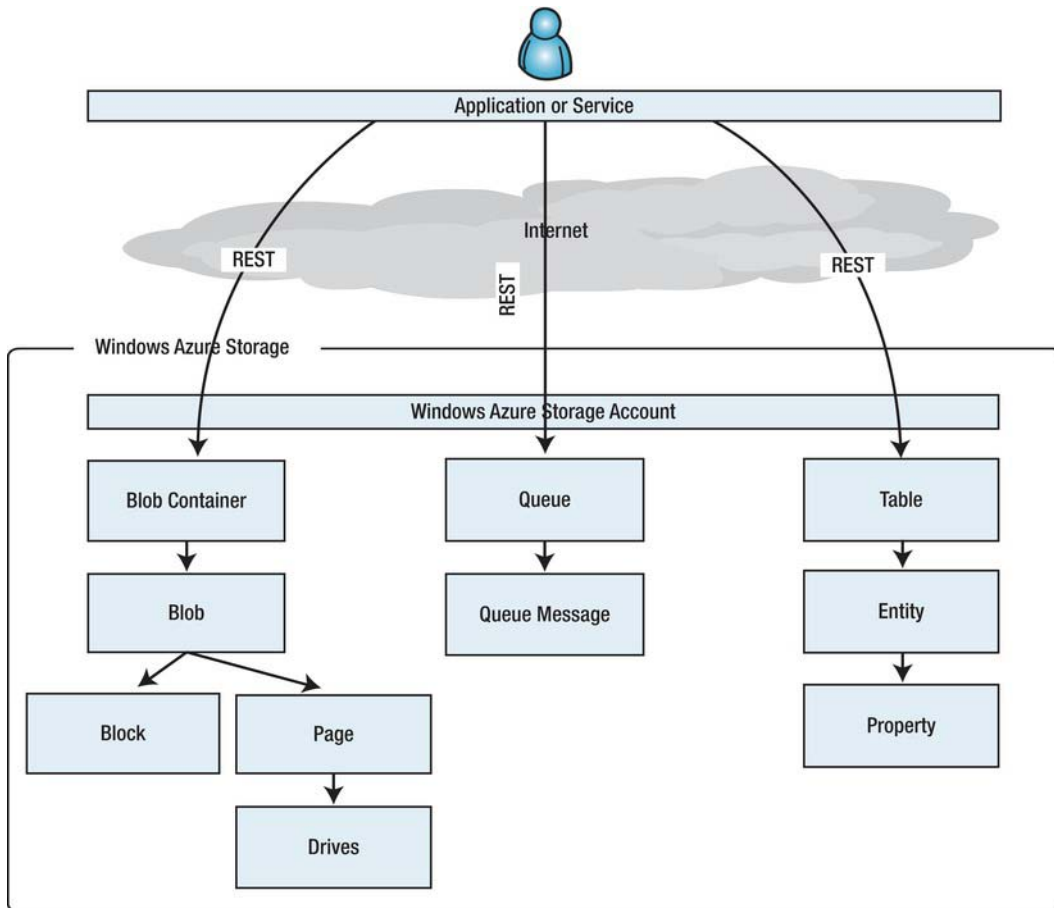


Figure 1-13. Storage service architecture

Windows Azure storage supports four types of services: blobs, drives, queues, and tables. Blobs, queues, and tables have independent REST APIs. Drives are a special type of storage that is different from the other storage services. Drives are a type of page blob and are uploaded to the blob storage for mounting them to the compute nodes. Drives don't have a direct REST API because they behave like other page blobs when uploaded to the blob storage. Drives do have a managed API in SDK that you can use in your compute nodes.

Windows Azure Storage types are scoped at the account level. This means that when you open a storage account, you get access to all the Windows Azure storage services: blobs, queues, and tables.

A blob account is a collection of containers. You can create any number of containers in an account. A container consists of number of blobs. A blob can be further composed of a series of blocks or pages. I have covered blocks and pages in the Blob storage chapter.

A queue account is a collection of queues. An account can have any number of queues. A queue is composed of queue messages sent by the message sending applications.

Table 1-3 lists the commonalities and differences among the three storage types in Windows Azure.

Table 1-3. Windows Azure Storage

Feature	Blob	Queue	Table
Url Schema	http://[Storage Account].blob.core.windows.net/	[Container Name]/[Blob Name]	http://[Storage Account].queue.core.windows.net/[Queue Name] http://[Storage Account].table.core.windows.net/[Table Name]?\$filter=[Query]
MAX Size	200GB(block blob)/1TB (page blob)	8K (string)	Designed for terabytes of data
Recommended Usage	Designed for large binary data types	Designed for cross-service message communication	Designed for storing smaller structured objects like the user state across sessions
API Reference	http://msdn.microsoft.com/en-us/library/dd135733.aspx	http://msdn.microsoft.com/en-us/library/dd179363.aspx	http://msdn.microsoft.com/en-us/library/dd179423.aspx

Even though the Storage service makes it easy for Windows Azure compute services to store data within the cloud, you can also access it directly from on-premises applications using the REST API. For example, you could write a music storage application that uploads all your MP3 files from you client machine to the blob storage, completely bypassing the Windows Azure Compute service. Compute and Storage services can be used independently of each other in Windows Azure. There are several customers using the storage service purely for backing up their on-premises data to the cloud. You can create a scheduled automated utility that uploads back-up files to Windows Azure blob storage. This gives you clear separation between your production data and disaster recovery data.

■ **Note** The Windows Azure SDK also includes .NET-managed classes for calling storage service REST API. If you are programming in .NET, I recommend you to use the classes in Microsoft.WindowsAzure.StorageClient assembly.

Windows Azure, because it is a platform, does not provide you with any direct user interface for uploading files to the storage service. You have to build your own application client for using the storage service. There are several third-party tools like the Cloud Storage Studio from Cerebrata (www.cerebrata.com/Products/CloudStorageStudio/Default.aspx) you can use to upload and download files.

■ **Note** The Windows Azure Storage service is independent of the SQL Azure database service offered by the Windows Azure Platform. Windows Azure storage services are very specific to Windows Azure and unlike the compute and SQL Azure services, there is no parity with any on-premises product offered by Microsoft.

Management

Unlike on-premise applications, the deployment of cloud services in PaaS involves only software provisioning from the developer's perspective. In a scalable environment for provisioning multiple services across thousands of instances, you need more programmatic control over the provisioning process. Manually uploading service packages and then starting and stopping services from the portal interface works well for smaller services, but are time-consuming and error-prone for large-scale services. The Windows Azure Service Management API allows you to programmatically perform most of the provisioning functions via a REST-based interface to your Windows Azure cloud account. The Service Management API is the hidden jewel of the platform. It makes Windows Azure a truly dynamically scalable platform allowing you to scale-up and scale-down your application on-demand. Using the Service Management API, you can automate provisioning, de-provisioning, scaling, and administration of your cloud services. Some of the common scenarios for leveraging the Service Management API are as follows:

- Automating the provisioning and de-provisioning of your cloud services through a well-defined release management process. Figure 1-14 illustrates the typical provisioning process of loading the application package from blob store and deploying it through service management API.

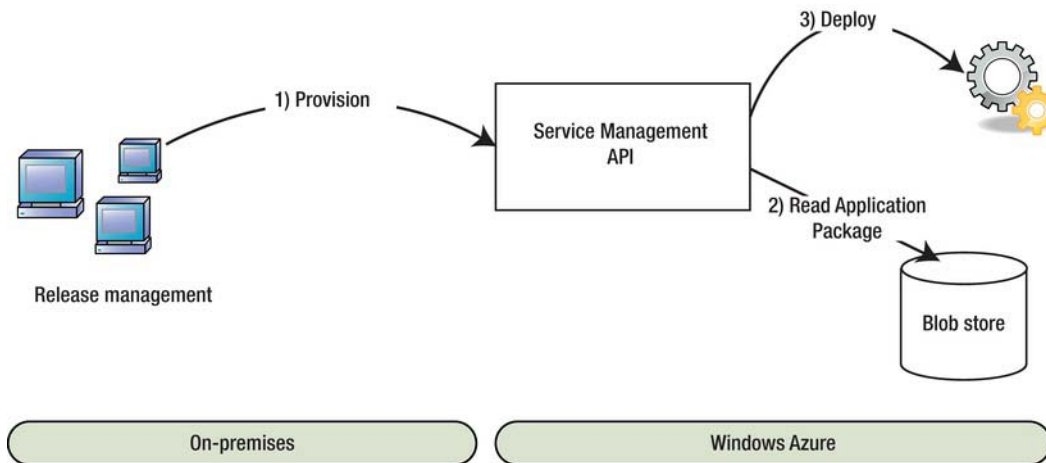


Figure 1-14. Provisioning using the Service Management API

- Dynamically scaling-up and down your applications based on the demand and application performance. Figure 1-15 illustrates a typical scaling architecture based on performance metrics of the roles instances of your application.

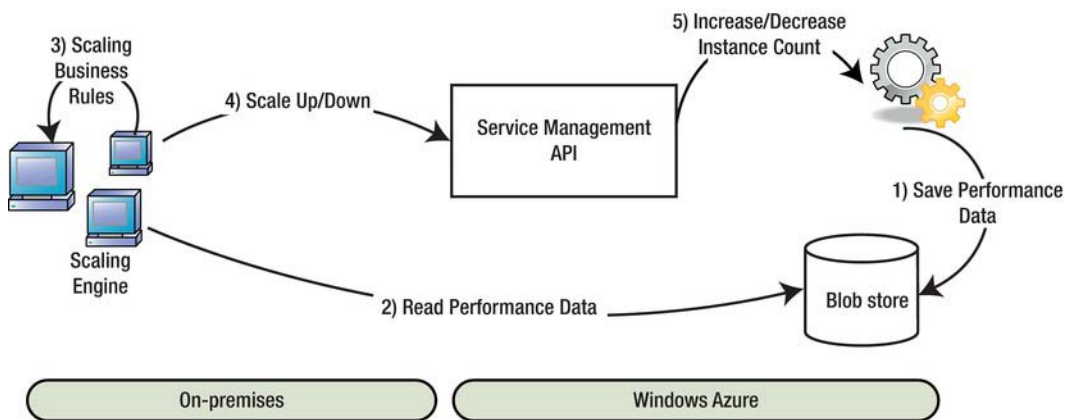


Figure 1-15. Scaling using the Service Management API

- Service Management API also enables you to build an enterprise applications store in Windows Azure. Combining it with other Windows Azure platform services like Access Control and Windows Identity Foundation, you can federate on-premises identities with applications running in Windows Azure.

■ **Note** There are some third-party tools that offer dynamic scaling as a service. AzureWatch from Paraleap (www.paraleap.com/AzureWatch) is one such tool and even Microsoft Consulting Services has an Auto-scale toolkit that is offered in the context of a consulting engagement. You can also build your own using the Service Management API.

SQL Azure

SQL Azure is a relational database service in the Windows Azure platform. It provides core relational database management system (RDBMS) capabilities as a service, and it is built on the SQL Server product code base. In the current version, developers can access SQL Azure using tabular data stream (TDS), which is the standard mechanism for accessing on-premise SQL Server instances through SQL client today. The SQL client can be any TDS-compliant client, like ADO.NET, LINQ, ODBC, JDBC, or ADO.NET Entity Framework.

■ **Note** At the time of writing, the maximum database size allowed per database was 50GB. There are well known partitioning patterns (e.g., sharding) for distributing data across multiple instances of 50GB databases. Built-in support for federation is on the roadmap for SQL Azure.

Figure 1-16 illustrates the core components of SQL Azure.

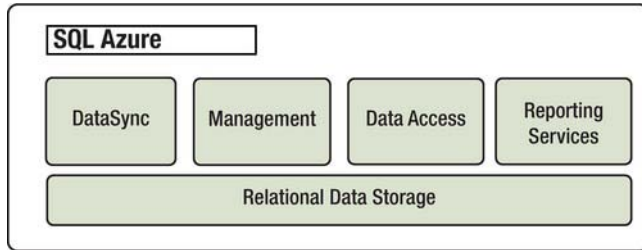


Figure 1-16. SQL Azure core components

The core services offered by SQL Azure are as follows:

Relational Data Storage – The relational data storage engine is the backbone of SQL Azure and is based on the core SQL Server code base. This component exposes the traditional SQL Server capabilities like the tables, indexes, views, stored procedures, and triggers. From a developer’s perspective, SQL Azure is a relational subset of SQL Server. Therefore, if you have developed for SQL Server, you are already a SQL Azure developer. The majority of the differences are along the physical management of the data and some middleware features like ServiceBroker that are not offered in SQL Azure.

Data Sync – The Data Sync capabilities provide the synchronization and aggregation of data to and from SQL Azure to enterprise, workstations, partners and consumer devices using the Microsoft Sync Framework. The Data Sync component has two flavors: Data Sync service between SQL Azure instances and Data Sync services between on-premises database and SQL Azure. Both the services are based on Microsoft Sync Framework. With the combination of the two flavors, you can trickle down data all the way to field offices that are in remote locations.

■ **Note** The Microsoft Sync Framework is included in the SQL Server 2008 product (<http://msdn.microsoft.com/en-us/sync/default.aspx>).

Management – The management component provides automatic provisioning, metering, billing, load-balancing, failover, and security capabilities to SQL Azure. Each database is replicated to one primary and two secondary servers. In case of a failover, the switching between the primary and the secondary server is automatic without interruptions. You can manage your SQL Azure databases from the Windows Azure portal as well as other existing toolsets like SQL Server Management Studio, OSQL, and BCP.

Data Access – The Data Access component defines different methods for accessing SQL Azure programmatically. Currently, SQL Azure will support Tabular Data Stream (TDS), which includes ADO.NET, Entity Framework,

ODBC, JDBC, and LINQ clients. Developers can access SQL Azure either directly from on-premise applications or through cloud services deployed in Windows Azure. You can also locate a Windows Azure compute cluster and a SQL Azure instance in the same datacenter for faster data access.

Reporting Services – The Windows Azure platform is new and you should expect it getting rich in features every year. Microsoft is committed to reducing the parity gap between on-premises and Windows Azure platform services. SQL Azure is no exception; during PDC 2010, Microsoft announced the availability of SQL Reporting Services in Windows Azure platform. This brings reporting capabilities to your applications and an essential step towards business intelligence in the cloud.

Some of the common scenarios for leveraging the SQL Azure are as follows:

Database consolidation – If you have home-brewed databases that are sitting under desktops or isolated islands, then having SQL Azure as one of the options in your database consolidation strategy will greatly help you strategize your long term goal of reducing the operations footprint. You can migrate to SQL Azure from Access, MySQL, SQL Server, DB2, Oracle, Sybase, and pretty much any relational database to SQL Azure with proper tools like SQL Server Migration Assistant (SSMA) and SQL Azure Migration Wizard from Codeplex. If there is no direct migration path to SQL Azure for a particular database, then the most common path is from third-party database to SQL Server and then from SQL Server to SQL Azure. SQL Azure will provide you with ability to quickly provision databases and reduce the time to market for your solution. You don't have to wait for provisioning clustered hardware for running your databases.

Back-end for Windows Azure Compute – SQL Azure is the recommended relational database for applications running in Windows Azure compute. If you co-locate your Windows Azure compute instances and SQL Azure in the same datacenter, you can get great performance advantage and you will also not incur any data-transfer costs.

Geo-replication of Data – SQL Azure and Data Sync services for SQL Azure give you the ability to synchronize two-way synchronizations across multiple geographies and also trickle down data to remote hubs. This kind of data movement allows you to move data closer to the application for better user experience. Figure 1-17 illustrates a common geo-replication pattern in SQL Azure.

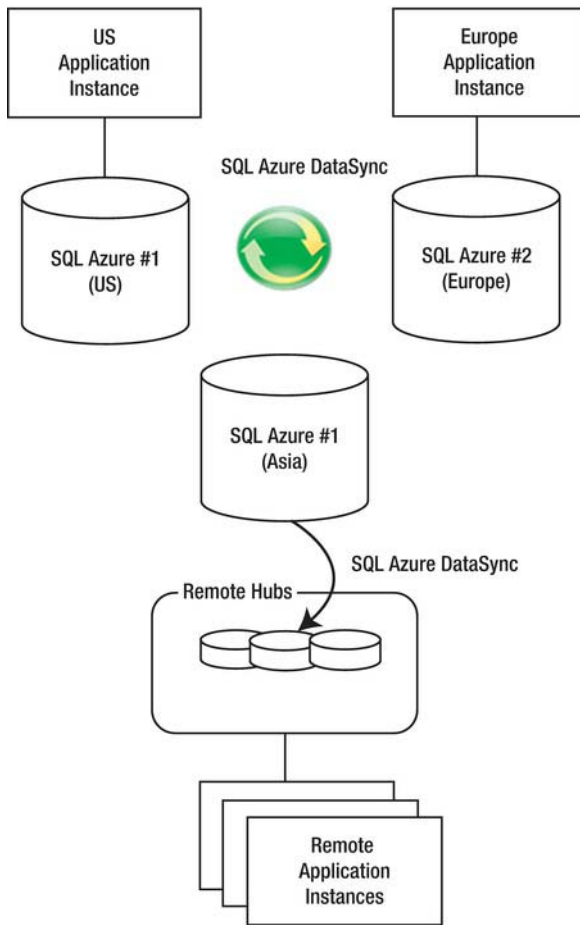


Figure 1-17. Geo-replication in SQL Azure

Quick Application Migrations – SQL Azure allows you to quickly migrate the relational database component of your on-premises application to Windows Azure. This is a big differentiator for Microsoft in comparison to its competitors. With minimal efforts, you can quickly move your existing relational data to Windows Azure and then focus on migrating the application. Once the data is moved to SQL Azure, you can just change the connection string of your application to point to the SQL Azure database.

■ **Tip** SQLAzure runs a labs program where you can try out upcoming features for free and provide feedback to the Microsoft product teams. You can find more information about the labs program here www.sqlazurelabs.com.

Windows Azure AppFabric

Windows Azure AppFabric is the cloud-based middleware platform hosted in Windows Azure. Windows Azure AppFabric is the glue for connecting critical pieces of a solution in the cloud and on-premises. You must have heard several times from industry speakers and even cloud enthusiasts that “everything is moving to the cloud.” I firmly believe that everything is *not* moving to the cloud. The disruptive effect of cloud computing is similar to the disruption created by cell phones. Cell phones gave users quick provisioning and anywhere access to communications. But, some critical functions like building security still run over landlines and have no plans for switching to a cell phone network. Windows Azure AppFabric provides you with components for integrating Windows Azure applications with on-premises applications. The core components of the Windows Azure AppFabric are as follows:

1. An Internet service bus, called Service Bus, for connecting applications running in Windows Azure (and other domains) and on-premises
2. A claims-mapping service, called Access Control Service, for supporting claims-based authorization in your applications running in Windows Azure and on-premises
3. A distributed caching service within the Windows Azure platform, called Windows Azure AppFabric Cache

■ **Note** Windows Azure AppFabric and Windows Server AppFabric are two different products. In the long run, Microsoft’s goal is to bring services available in Windows Server AppFabric to Windows Azure AppFabric. Some of the features like ACS and Service Bus will always remain exclusive to Windows Azure AppFabric.

I think of Azure AppFabric as the integration middleware of the Windows Azure platform, because it provides connectivity, caching, identity claims federation, and messaging capabilities among distributed applications. You can leverage these capabilities not only for cloud services but also for on-premises applications. Microsoft’s strategy is to provide Windows Azure AppFabric as a middleware and building blocks for building and deploying distributed applications on Windows Azure. Figure 1-18 illustrates the three core services of Windows Azure AppFabric.

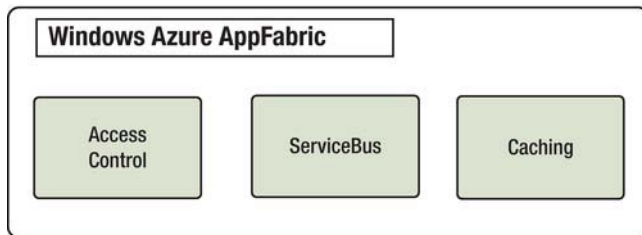


Figure 1-18. Windows Azure AppFabric core services

Access control – The access control service provides rules-driven, claims-based access control for distributed applications. The access control service is

designed to abstract identity providers from your application. It maps input claims from different types of identity providers to a standard set of output claims known to your application. This architecture provides a model for dynamically adding and removing identity providers without changing a single line of code in your application. The most common uses of Access Control service are:

- Providing identity federation between multiple partner identity providers and partner portals
- Providing enterprise identity federation using ADFS 2.0 and enterprise applications running in Windows Azure

Service bus – The service bus is a generic Internet service bus based on the Windows Communications Foundations (WCF) programming model. It is analogous to the Enterprise Service Bus (ESB) popularly seen in large enterprises. Unlike the ESB, the Azure AppFabric Service Bus is designed for Internet scale and messaging with cross-enterprise and cross-cloud scenarios in mind. The service bus provides key messaging patterns like publish/subscribe, point-to-point, and durable buffers for message exchanges across distributed applications in the cloud as well as on-premise. You can expose an on-premises line-of-business application interface as a Service Bus endpoint and then consume that endpoint from a Windows Azure application or any other application. The most common uses of Service Bus are:

- Accessing line-of-business data from Windows Azure applications
- Providing connectivity to data residing in on-premises applications at the web service level (Windows Azure Connect provides connectivity at the network level)

Caching – The caching service was announced during PDC 2010. It is a late comer to the Azure AppFabric family, but one of the most valuable. The caching service provides distributed caching for applications running Windows Azure. Caching is an essential component for any internet-scale application. Before the caching service, developers either built custom caching components or modified third-party components like memcached (<http://memcached.org/>) to run on Windows Azure. The Azure AppFabric caching service makes caching a first-class citizen of the Windows Azure platform. It provides in-memory as well as external datacenter co-located caching service. The most common uses of the caching service in Windows Azure are:

- In-memory caching of SQLAzure database data
- Caching on-premises data to be consumed by Windows Azure applications
- Caching multi-tenant user interface data for providing high-performance interactive applications
- Caching user state in multi-player gaming applications
- Caching location-based usage information
- Caching session state of users

■ **Tip** Like SQLAzure, Windows Azure AppFabric also runs a labs program where you can try upcoming features for free. You can login and start playing with these features here <https://portal.appfabriclabs.com>.

Now that we have covered the Windows Azure technology fundamentals, let's see how the pricing for all these features is structured. In the cloud, every feature you use is metered based on usage like your electricity bill, therefore architects and developers need to choose appropriate features and are naturally forced to not over-architect an application.

Windows Azure Platform Pricing

Each Windows Azure platform component is priced differently and within each component there are further pricing choices. Typically, architects are used to designing applications assuming capital investment for supporting maximum capacity. But, cloud platforms give you the flexibility for designing applications for minimum capacity and scale up dynamically based on demand. Such flexibility adds a new variable “operating cost” to your design. Every cloud resource you plan to utilize in your design has a cost associated with it. Cloud computing gives rise to a new architecture paradigm I call cost-driven architecture (CDA). In CDAs, an architect iteratively evaluates the operating cost of the architecture and modifies the architecture according to the cost boundaries of the application as shown in Figure 1-19.

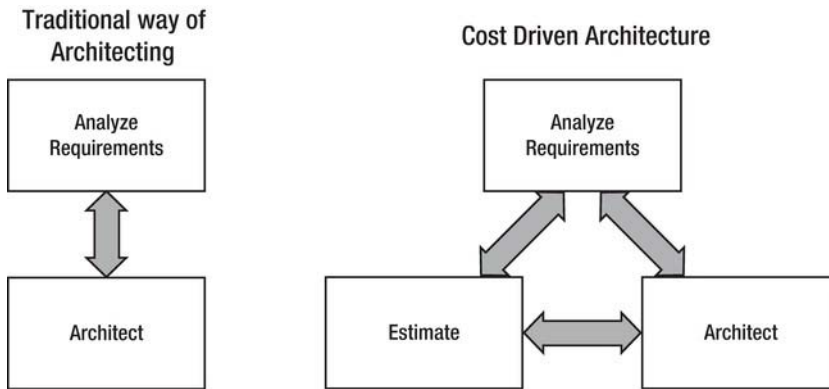


Figure 1-19. Cost driven architecture

CDAs are tied directly to the pricing of the cloud services. The pricing for Windows Azure platform components are available publicly at:

www.microsoft.com/windowsazure/pricing/default.aspx

Figure 1-20 illustrates the pricing for Windows Azure Compute and Storage services.



- Compute = \$0.12 / hour for small instance and \$0.05 / hour for extra small instance*
- Windows Azure Connect = No charge during CTP**
- Storage = \$0.15 / GB stored / month
- Storage transactions = \$0.01 / 10K
- Data transfers (excluding CDN) = \$0.10 in / \$0.15 out / GB - (\$0.10 in / \$0.20 out / GB in Asia)***
- CDN data transfers = \$0.15 GB for North America and Europe (\$0.20 GB elsewhere)**
- CDN transactions = \$0.01 / 10K**

* Extra small compute instances will begin availability in beta this calendar year and will be billed separately from other compute instance sizes.

** The Windows Azure Connect service will begin its Community Technology Preview (CTP) this calendar year.

*** No charge for inbound data transfers during off-peak times through March 31, 2011

Click "Sign up now" to view our offers page.



Windows Azure Service Level Agreement

For compute, we guarantee that when you deploy two or more role instances in different fault and upgrade domains, your internet facing roles will have external connectivity at least 99.95% of the time. For storage and CDN, we guarantee that at least 99.9% of the time we will successfully process correctly formatted requests. [More information on Service Level Agreements.](#)

Measuring Windows Azure Consumption

- **Compute time, measured in service hours:** Windows Azure compute hours are charged only for when your application is deployed. When developing and testing your application, developers will want to remove the compute instances that are not being used to minimize compute hour billing. Partial compute hours are billed as full hours.
- **Storage, measured in GB:** Storage is metered in units of average daily amount of data stored (in GB) over a monthly period. For example, if a user uploaded 30GB of data and stored it on Windows Azure for a day, her monthly billed storage would be 1 GB. If the same user uploaded 30GB of data and stored it on Windows Azure for an entire billing period, her monthly billed storage would be 30GB. Storage is also metered in terms of storage transactions used to add, update, read and delete storage data. These are billed at a rate of \$0.01 for 10,000 (10k) transaction requests
- **Data transfers measured in GB (transmissions to and from the Windows Azure datacenter):** Data transfers are charged based on the total amount of data going in and out of the Azure services via the internet in a given 30-day period. Data transfers within a sub region are free.
- **Transactions, measured as application requests.**

Figure 1-20. Windows Azure compute and storage pricing


A couple of things to observe in Figure 1-20 are as follows:

The compute service charges are per hour of consumption and the price is based on the size of the virtual machine instance. You will be charged not only for a running application but also for an application that is deployed but not running. You have to unload the application to avoid any charges.

The Storage service charges are for data storage per GB as well as transactions (ingress and egress) in and out of the datacenter. So, if your compute instances and storage are located in the same datacenter, there will be no transaction cost for data transfer between the two, but if the compute instances access storage service from another datacenter, you will incur transaction costs.

■ **Note** Microsoft no longer charges for input data bandwidth (ingress). There is network bandwidth charge for only egress (data-out); therefore it is easier to migrate your data into Windows Azure.

Figure 1-21 illustrates the pricing for SQL Azure.



Microsoft SQL Azure

- **Web Edition:**
 - Up to 1 GB relational database = \$9.99 / month
 - Up to 5 GB relational database = \$49.95 / month**
- **Business Edition:**
 - Up to 10 GB relational database = \$99.99 / month
 - Up to 20 GB relational database = \$199.98 / month**
 - Up to 30 GB relational database = \$299.97 / month**
 - Up to 40 GB relational database = \$399.96 / month**
 - Up to 50 GB relational database = \$499.95 / month**
- Data transfers = \$0.10 in / \$0.15 out / GB - (\$0.10 in / \$0.20 out / GB in Asia)*

* No charge for inbound data transfers during off-peak times through March 31, 2011

Click "Sign up now" to view our offers page.

☒ Sign up now

SQL Azure Service Level Agreement
 SQL Azure customers will have connectivity between the database and our internet gateway. SQL Azure will maintain a "Monthly Availability" of 99.9% during a calendar month. [More information on Service Level Agreements.](#)

Measuring SQL Azure Consumption

Web Edition Relational Database includes:

- Up to 5 GB of T-SQL based relational database*
- Self-managed DB, auto high availability and fault tolerance
- Support existing tools like Visual Studio, SSMS, SSIS, BCP
- Best suited for Web application, Departmental custom apps

Business Edition DB includes:

- Up to 50 GB of T-SQL based relational database*
- Self-managed DB, auto high availability and fault tolerance
- Additional features in the future like auto-partition, CLR, fanouts etc
- Support existing tools like Visual Studio, SSMS, SSIS, BCP
- Best suited for SaaS ISV apps, custom Web application, Departmental apps

A monthly fee is charged for each user database of SQL Azure. That database fee is amortized over the month and charge on a daily basis. You pay for the user databases you have on the days you have them. Master databases are not charged.

Figure 1-21. SQL Azure pricing

A few things to observe in Figure 1-21 are:

- There are two types of database editions: Web and Business (5GB or more is the Business Edition).
- The maximum size per database is 50GB, but you can create multiple 50GB databases and partition data between those instances.
- “The database fee is amortized over the month and charged on a daily basis.” If you dynamically create and delete databases frequently, you have to make sure you factor-in this daily cost.
- Similar to storage service, SQL Azure also charges for data transfer between datacenters. There will be no charge for data transfer within the same datacenter.

Figure 1-22 illustrates the pricing for Windows Azure AppFabric.



Windows Azure AppFabric

Access Control

- Access Control transactions = \$1.99/100K

Service Bus connections

- \$3.99 per connection on a "pay-as-you-go" basis
- \$9.95 for a pack of 5 connections
- \$49.75 for a pack of 25 connections
- \$199 for a pack of 100 connections
- \$995 for a pack of 500 connections

Data transfers

- Data transfers = \$0.10 in / \$0.15 out / GB - (\$0.10 in / \$0.20 out / GB in Asia)*

* No charge for inbound data transfers during off-peak times through March 31, 2011

Click "Sign up now" to view our offers page.

☒ Sign up now

AppFabric Service Level Agreement
Uptime percentage commitments and SLA credits for AppFabric are similar to those specified in the Windows Azure SLA. [More information on Service Level Agreements](#)

Measuring AppFabric Consumption

AppFabric Service Bus connections can be provisioned individually on a "pay-as-you-go" basis or in a pack of 5, 25, 100 or 500 connections. For individually provisioned connections, you will be charged based on the maximum number of connections you use for each day. For connection packs, you will be charged daily for a pro rata amount of the connections in that pack (i.e., the number of connections in the pack divided by the number of days in the month). You can only update the connections you provision as a pack once every seven days. You can modify the number of connections you provision individually at any time.

For AppFabric Access Control transactions, customers will be charged the actual number of transactions utilized for the billing period (i.e., not in discrete blocks of 100,000 transactions), plus data transfers in or out.

Figure 1-22. Windows Azure AppFabric pricing

■ **Tip** For quick cost estimation, there is a nice pricing calculator on the Windows Azure web site at www.microsoft.com/windowsazure/pricing-calculator/. I encourage you to try it out by mapping any of your on-premise application to Windows Azure.

All of the Windows Azure platform components can be managed from the Windows Azure platform management portal. Let's review that now.

Management Portal – Let's Provision

■ **Note** By the time this book will be released, the portal experience will be much different, but the concepts will still remain the same.

Microsoft has designed the Windows Azure platform management portal in Silverlight, providing better interactivity and a central place for managing all the Windows Azure platform components. You can access the portal via the following URL <https://windows.azure.com>. Before accessing the portal, you will need to create a subscription using your Live Id. Microsoft offers several introductory specials that can be found here www.microsoft.com/windowsazure/offers/. Sometimes there are one month free trials. If you have MSDN subscription, you also receive some free hours per month. After you create a subscription, you can log in to the management portal using the same Live Id you used for creating the subscription.

Once you create a subscription, you get access to all the Windows Azure platform components. After you login to the management portal, you will be taken to the main portal page. Figure 1-23 illustrates the screenshot of the main portal page.

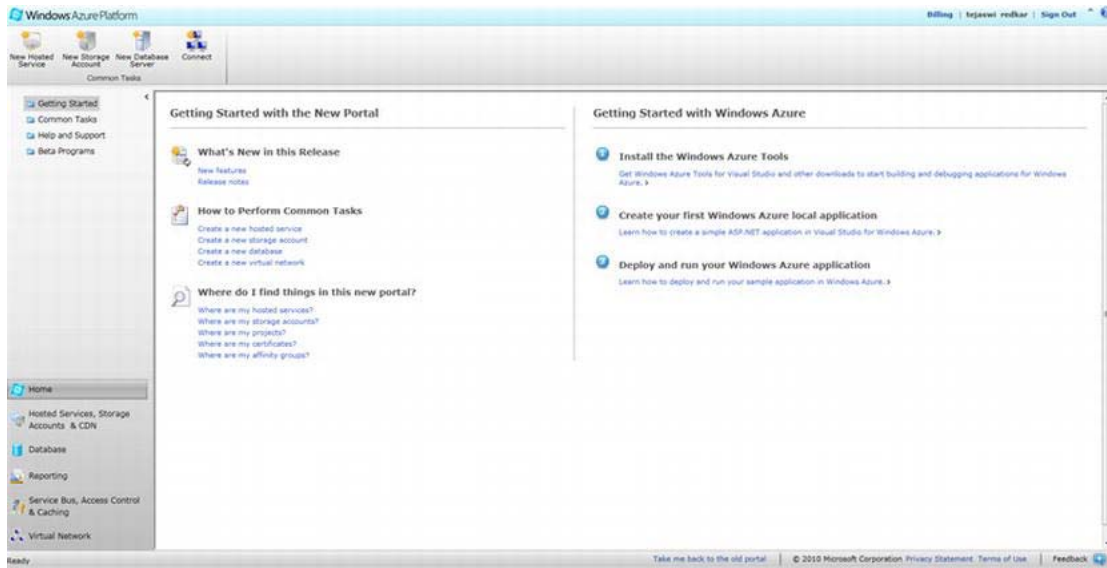


Figure 1-23. Windows Azure platform management portal

The management portal user interface is highly interactive and gives you the feeling of interacting with the desktop software. The left navigation list all the services from the Windows Azure platform and the top navigation bar lists commands in context of your navigation. For example, on the main page you can create new services: Hosted Service, Storage Service, Database Server, and Windows Azure Connect service. When you navigate to the Database tab, you will see the commands change in the context of the SQL Azure database server, as shown in Figure 1-24.



Figure 1-24. Management portal commands in Database context

For running examples from this book, you will need a provisioned account. Even though some of the applications you can run in the Windows Azure development fabric, for experiencing the real cloud environment, you will need a Windows Azure platform account.

Figure 1-25 illustrates a typical developer workflow on the Windows Azure Platform.

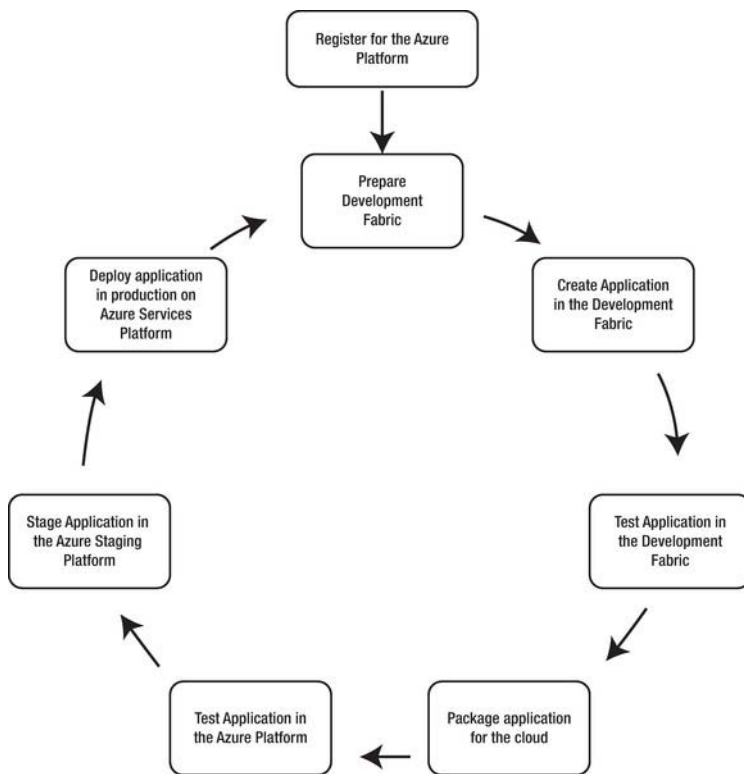


Figure 1-25. Windows Azure platform developer workflow

The typical developer workflow steps for Windows Azure Platform follow:

1. Create a Windows Azure account (i.e., an account for Windows Azure, AppFabric, or SQL Services).
2. Download and prepare the development fabric to create a local cloud platform.
3. Create an application in the development fabric.
4. Test the application in the development fabric.
5. Package the application for cloud deployment.
6. Test the application on Windows Azure in the cloud.
7. Stage the application in the Windows Azure staging environment in the cloud.
8. Deploy the application in the production farm.

Windows Azure platform is a living platform; new features are added every few weeks. One such feature that I have not covered in detail is the Windows Azure Marketplace DataMarket. I will give you a brief overview of the topic on this subject that will conceptually understand the service offering.

Windows Azure Marketplace DataMarket

Windows Azure Marketplace DataMarket is a data service broker that runs on Windows Azure. It standardizes the data consumption and data publishing interfaces with a standard web protocol called Open Data Protocol (OData). Let's say you want to build a mobile application that gives end users insights into real estate sales and rental rates in relationship to local crime statistics. What would be your process for building such an application? The application is completely driven by public domain data and three different data sources: Real Estate Sales, Rental Data, and Crime Statistics. Then you realize that the programmatic interfaces for these data sources are different. Ultimately, you end up building your own service that transforms and aggregates the data from these three sources and presents it to the user. It would have been ideal if all the three data sources had a standardized interface so that you don't have to transform the data and only worry about aggregating and presenting it in the user's context.

The DataMarket standardizes this conversation by brokering such data feeds in OData format. OData is a web protocol that standardizes data publishing and consumption so that tightly integrated data can be exposed as standardized cross-platform feeds. You can find more information on OData at www.odata.org. The web site also provides client (consumption) and server (publisher). As an ISV or a developer, DataMarket provides you with a single point for consuming public domain as well as premium commercial data in a standard format. You maintain a single billing relationship with Microsoft and not worry about signing multiple checks for data providers. As a data provider, DataMarket provides you with a global brokerage service for monetizing your data by delivering it anywhere in the world. DataMarket also provides an authorization model for delivering your data only to your customers. You don't have to worry about maintaining separate customer relationships and separate infrastructure for publishing your data globally. You can find more information on the DataMarket on its web site <https://datamarket.azure.com/>. Figure 1-26 illustrates the high-level architecture of the DataMarket platform.

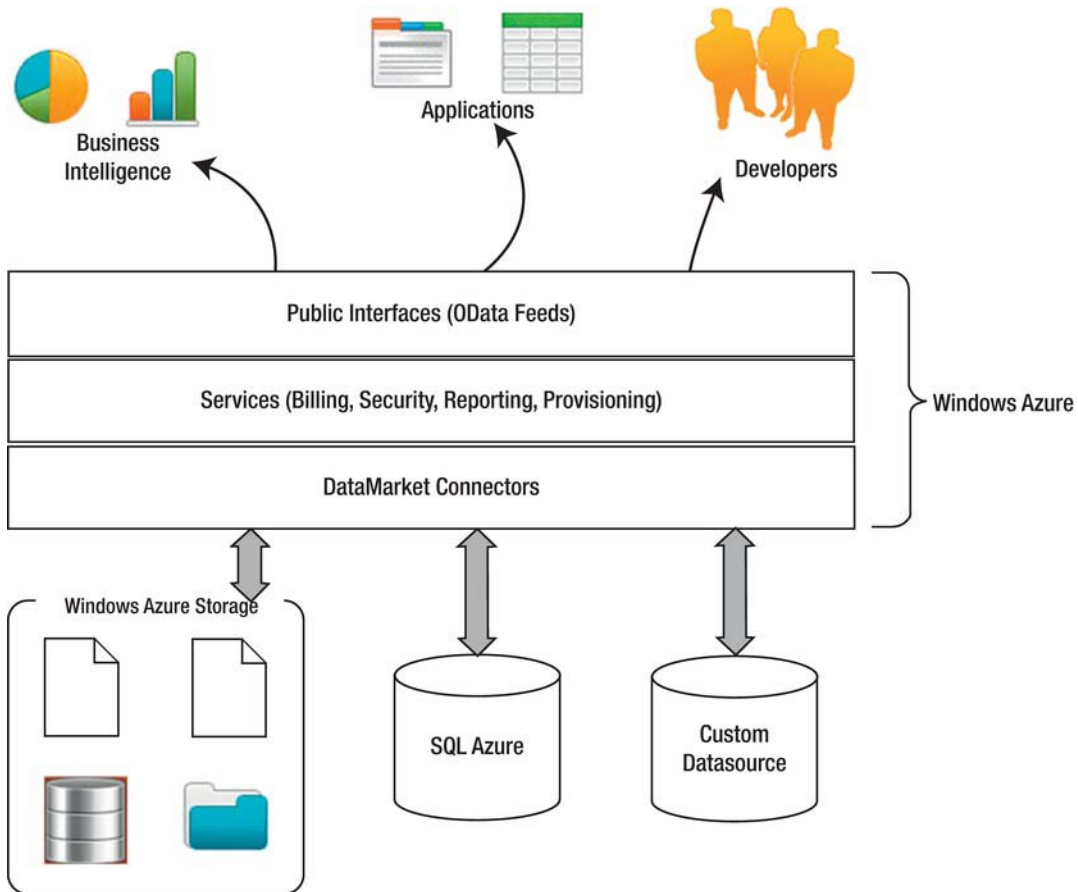


Figure 1-26. Windows Azure Marketplace DataMarket

As shown in Figure 1-26, you can expose your custom data source, SQL Azure data source or a data source from Windows Azure Storage to the DataMarket. The DataMarket itself does not store your data, but provides you with a marketplace for your data. The DataMarket connectors provide the API services for publishing your data to the DataMarket. The DataMarket in turn exposes the data to the consumers as OData feeds. The services layer manages the billing, security, reporting, and provisioning.

Knowledge is best applied in the context of real-world scenarios. In the next section, you will see some common scenarios in the context of all the Windows Azure platform components I have covered so far. This section will help set the stage for all the following chapters and will give you a broader picture from a solution perspective.

Windows Azure Platform Common Scenarios

After working with several customers over the past few years on the Windows Azure platform, I have compiled a list of most commonly used scenarios. I have grouped these scenarios into three primary categories: Foundational, Enterprise, and ISV.

Foundational Scenarios

Foundational scenarios are core scenarios commonly found in the architectures of the cloud applications. These scenarios leverage the core features of PaaS offerings. Table 1-4 lists the descriptions of these foundational scenarios.

Table 1-4. Foundational Scenarios

Foundational Scenarios	Description
Dynamic Scaling	This is a core capability of PaaS. Windows Azure platform offers this capability through APIs. In this scenario, a scaling engine keeps track of the performance of your instances and then based on a business rule like performance threshold or time, dynamically increases or decreases the number of instances in your cloud application.
Distributed Caching	Cloud applications run on shared hardware in Microsoft's datacenter. You do not have access to server and network hardware for optimizing performance at the hardware layer. Therefore, you have optimize software for performance by employing techniques like distributed caching. Windows Azure AppFabric caching provides such a service that you can use in your cloud applications.
Multi-tenancy	For serving multiple customers from a single application, you have to implement multi-tenancy in all the tiers of your application. Every tier has its own design patterns for building multi-tenant applications. You can also deploy separate application per tenant if your business model dictates that. Billing and metering per tenant is the biggest challenge in building multi-tenant applications. You need to provide data capture hooks within the application for capturing each tenant's usage of the application.
Geo-replication	With the spread of consumer and enterprise applications at a global scale, replicating data around the world and also employing bi-directional synchronizations for that data has become a necessity. SQL Azure DataSync and Microsoft Sync Framework are two technologies you can use to not only replicate data across the globe but also trickle down the data all the way to consumer devices for optimal performance and pleasant user experiences.
Identity	Identity management is one of the core requirements for building an extensible and backwards-compatible cloud application. Integrating

Management	enterprise and consumer identities seamlessly in your cloud application is important to make it pervasive and backwards compatible without modifying any code. Windows Azure AppFabric Access Control Service and Windows Identity Foundation (WIF) provide you with a service and framework for integrating different kinds of identity providers in your application.
Storage Management	Even though Windows Azure platform provides you with storage service and SQL Azure, you have to architect security and the flow of data from data sources to cloud storage and back. You have to explicitly storage management solution in your cloud applications.

Enterprise Scenarios

Enterprise scenarios are typically seen in enterprise cloud applications. Enterprises are business entities that manage a business and also interact with other businesses called partners. Enterprise scenarios are tightly bound to enterprise boundaries like enterprise identities and enterprise security policies. Table 1-5 lists the descriptions of these enterprise scenarios.

Table 1-5. Enterprise Scenarios

Enterprise Scenarios	Description
Enterprise Identity Integration	Enterprise cloud applications need access to enterprise identities for authentication and authorization. Windows Azure platform currently does not provide you with Active Directory Service; therefore you have to leverage tools like ADFS 2.0 and WIF for integrating enterprise on-premises identities into your cloud applications. These tools also enable you to integrate with identities of your business partners seamlessly (Partner portals, Disaster Recover Application, etc.).
Application Migrations	Application migration is a common scenario in enterprises. Windows Azure platform provides you with a runtime for running consolidated sets of applications. Enterprises build application consolidation strategies around Windows Azure platform in order to reap the benefits of cost, agility and high-availability.
Data Migrations	Lot of enterprises would like to relieve themselves of managing any non-sensitive data. Typically, data in enterprises reside in structured and unstructured sources. Enterprises leverage Blob storage and SQL Azure for storing their unstructured and relational data. Once the data is migrated and organized in the cloud, any application from anywhere can access this data securely.
Third-party applications	Enterprises are seeing VMRole as an attractive option for deploying third-party applications in Windows Azure. Typically, these

applications have lengthy installation process and do not need to scale.

Business Intelligence (BI) in the cloud	With unlimited storage and compute capacity in the cloud, enterprises have the opportunity to mine business data without worrying about the computing power needed for it. SQL Reporting Services in the cloud and Silverlight are popular BI presentation tools available in the Windows Azure platform. I commonly see departmental BI reporting applications using SQL Azure and Silverlight built on the Windows Azure platform. SQL Server Analysis Services (SSAS) and SQL Server Integration Services (SSIS) are still not available in the cloud to move the end-to-end BI process in the cloud, but I do see it coming in the next couple of years.
Hybrid Applications	Large enterprise applications like line-of-business applications are not yet available in the cloud, and enterprises have to settle down running them on-premises. There are also scenarios where 90% of the application can run comfortable in the cloud, but the 10% requires access to sensitive data that cannot be moved to the cloud or the data resides in a line-of-business application. In this case, enterprises leverage either Windows Azure AppFabric ServiceBus or Windows Azure Connect for retrieving the 10% data from the on-premises source.

ISV Scenarios

Windows Azure platform is very attractive for ISVs because they can deploy their software service and offer it to multiple customers through same of different endpoints. They don't have to manage any hardware and can dynamically scale as the number of customers increase. Table 1-6 lists the descriptions of these foundational scenarios.

Table 1-6. ISV Scenarios

ISV Scenarios	Description
High Scale/Batch Compute	High Scale compute requires dynamically scaling compute power and the ability to turn the capacity off once the workload completes for avoiding too much idle time. Similarly, any batch processing system only requires a specific timeslot for processing the load. The system usually runs idle till the next workload is initiated. Windows Azure platform provides you with unlimited compute power, dynamic scalability, and you can stop the system after the workload finishes so that you don't have to run any idle capacity.
High-Growth Sites	Startup companies or new initiatives in larger companies leverage the dynamic scaling capability of the Windows Azure platform by quickly building and deploying a new application scale up

dynamically as the demand grows. If the demand goes down, the application can be easily scaled-down or removed. There is no capital investment required upfront for designing the system for maximum capacity.

Software
Modernization

There are a lot of ISV packaged software installations that run as islands within enterprises and small businesses. With the wave of mobile applications, the demand for these packaged software applications is going down. Some ISVs are modernizing these existing software installations by providing Windows Azure AppFabric Service Bus interfaces and providing mobile access to these service interfaces.

Cloud Bursts
(Predictable/
Un-
predictable)

Bursts are sudden change in the usage demand of the application. For example, during super-bowl season, there is a sudden change in demand for Pizza orders. If the system is not designed to handle these bursts, the company may lose business. Bursts can be predictable or unpredictable. In Windows Azure, you can detect these bursts by monitoring performance of the instances or input queues to the application. Then, based on business rules, you can increase the capacity of your application by dynamically starting more instances.

These categories are just a guidance and not specifically driven by either ISV or Enterprise applications. In real-world, you will see a mix and match of these scenarios across different types of businesses.

Summary

Windows Azure platform is the most comprehensive PaaS offering in the industry today. In this chapter, I gave you a high-level overview of all the Windows Azure platform features. I also went over some of the common scenarios and trends I am observing in the cloud computing industry. In the next few years, you will clearly see the disruptive effect of the cloud in enterprises. I always recommend my customers to have a top-down strategy in handling this disruption rather than jumping on it on a per-application basis. As the platform matures in features, more and more enterprises will be moving their applications to the cloud.

In the next chapter, I will go over the Windows Azure Compute service in detail. You will also learn to build applications for the Windows Azure Compute service.

Bibliography

Apache Software Foundation. (n.d.). *Apache Hadoop*. Retrieved from <http://hadoop.apache.org>

Factor, A. (2001). *Analyzing Application Service Providers*. Prentice Hall.

Google. (n.d.). *Google AppEngine*. Retrieved from Google: <http://code.google.com/appengine>

Google. (n.d.). *Google Apps*. Retrieved from Google Apps:

<http://www.google.com/apps/intl/en/business/index.html>

Mario Barbacci, M. H. (1995). *Quality Attributes*. Pittsburgh, Pennsylvania 15213: Software Engineering Institute, Carnegie Mellon University.

Microsoft Corporation. (n.d.). *About Windows Azure*. Retrieved from Windows Azure:

<http://www.azure.com/>

Microsoft Corporation. (n.d.). *Windows Azure Pricing*. Retrieved from Windows Azure:

<http://www.microsoft.com/azure/pricing.mspx>

Open ID Foundation. (n.d.). Retrieved from <http://openid.net/foundation/>

Staten, J. (2008). *Is Cloud Computing Ready For The Enterprise?* Forrester Research, Inc.

Windows Azure Compute

Enterprises today run on several flavors of operating systems such as Windows, UNIX, and mainframes. As businesses grow, enterprises have to expand their data and processing capacities by buying more servers and operating systems to support the new capacity. Typically, businesses have to plan for growth well in advance to budget the expenses. The tipping point, where investments in new server systems may not justify the value they provide to the business, is not far away. This is because server systems are expensive to provision and maintain, and they become obsolete before they provide any return on investment (ROI) to the business. As a result, IT managers face constant struggle in justifying server upgrades. On the other hand, businesses should also plan for shrinking capacity and should be able to quickly reduce costs and expenses to compete better in the marketplace.

By adopting Windows Azure, businesses can outsource their infrastructure elasticity to Microsoft, and thus dynamically adjust to the real-time business needs.

In this chapter, I will discuss Windows Azure compute architecture and service management components in detail.

Compute Service

In Chapter 1, you learnt the high-level compute service architecture. In this chapter, I will discuss some of the details of the compute service.

Figure 2-1 illustrates the key components of the Windows Azure compute service.

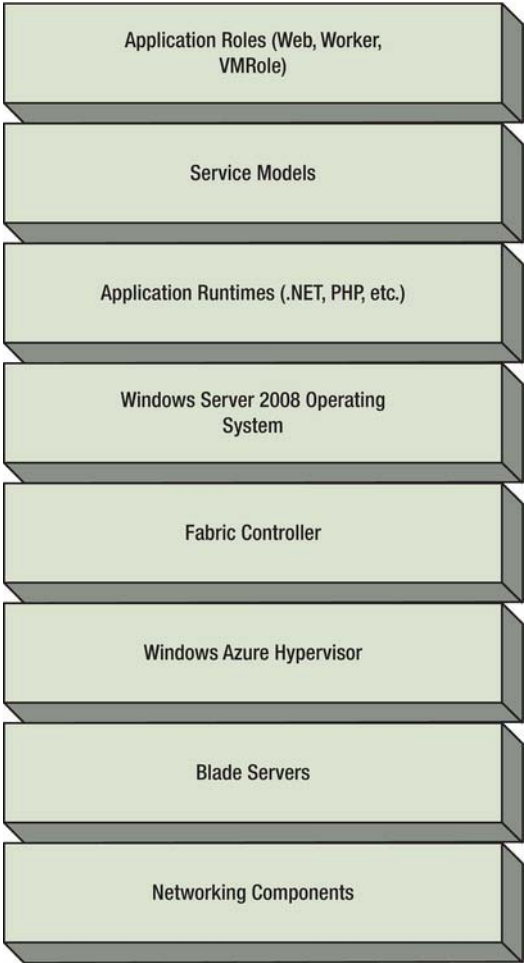


Figure 2-1. Windows Azure compute service components

Networking Components

The networking hardware and software forms the networking backbone for all the internal and external communications. Typical hardware components include gateways, routers, switches, hubs, and fiber optic cables across datacenters. Typical software networking components include software load-balancers and virtual routers. The hardware networking components are fixed, whereas the software networking components are dynamic and therefore can be provisioned on demand. From previous experience, the hardware networking components are more difficult to replace than software ones and therefore need to be redundant. One of the common mistakes is deploying two instances of the load-balancer (or router) from the same vendor as a redundancy mechanism. So, if there is a defect in the product version that freezes the device, the redundant device is also likely to fail, because it is identical. If the networking backbone fails, everything above the stack fails. Therefore, designing network redundancy with multi-vendor products is critical for maintaining high-availability of the datacenter services.

Blade Servers

The server farm in Windows Azure datacenters consists of commodity-grade blade servers. The server hardware is low cost and is replaced instead of repaired in case of failures. This is because the replacement cost of the hardware is less than the repair cost. All the hardware is capable of running Windows Server 2008 64-bit compatible operating systems.

Windows Azure Hypervisor

The Windows Azure Hypervisor is a customized version of Hyper-V for Windows Azure. It is specifically tailored for Windows Azure.

Fabric Controller

The Fabric Controller is the heart of Windows Azure and is responsible for the following:

- Provisioning operating systems on server hardware.
- Allocating resources for cloud services.
- Managing cloud service lifecycle.
- Maintaining the cloud service quality attributes defined by the service level agreement (SLA).

Operating Systems

Table 2-1 shows the versions of guest operating systems supported by Windows Azure.

Table 2-1. Windows Azure Operating System Support

Windows Azure Operating System	Windows Server OS version	Roles
Guest OS 1.x	Windows Server 2008 (64 bit)	Web role, Worker role
Guest OS 2.x	Windows Server 2008 R2 (64 bit)	Web role, Worker role, VM role

Application Runtimes

Application runtimes, like .NET Framework and Java Runtime, are responsible for running your applications on Windows Azure. By default, .NET Framework is installed on these instances, but you can also install Java runtime or any runtime that runs on Windows Server 2008.

Service Model

A service model contains the metadata for your service. It describes the behavior of the service after it is deployed in Windows Azure. The Fabric Controller reads the service model definition of your service and deploys it to the desired end state. You can modify the configuration portion of the service model, such as the number of instances at runtime. Any modification to the definition portion of the service mode like the number of endpoints requires a restart. The service model must contain at least one role definition.

Application Roles

Application roles abstract the dependency of specific operating features from your application. For example, if you build a Web role, the web server in which the Web role is hosted is abstracted from you. The underlying web server in Windows Azure is IIS, but your application does not have to know about it. Windows Azure deploys your Web role to an IIS server transparently, thus relieving you of web application deployment task. Similarly, a Worker role is deployed as a background service in Windows Azure without requiring you to specifically install and deploy the service on each individual instance. The compute service follows a role-based model in which the provisioned instances run the roles defined by your application.

■ **Note** In Windows Azure terminology, an application is called as a cloud service. A cloud service is a grouping of one or more roles into a single packaged solution that can be deployed to Windows Azure as a distributed application farm.

Windows Azure supports three types of roles: Web role, Worker role, and VM role. When you design your cloud service, you define your architecture in terms of roles. Then, you group multiple roles as a cloud service and deploy to Windows Azure. Windows Azure deploys each of these roles into a separate virtual machine instance but within the same cloud service. Figure 2-2 illustrates the concept of cloud service and roles.

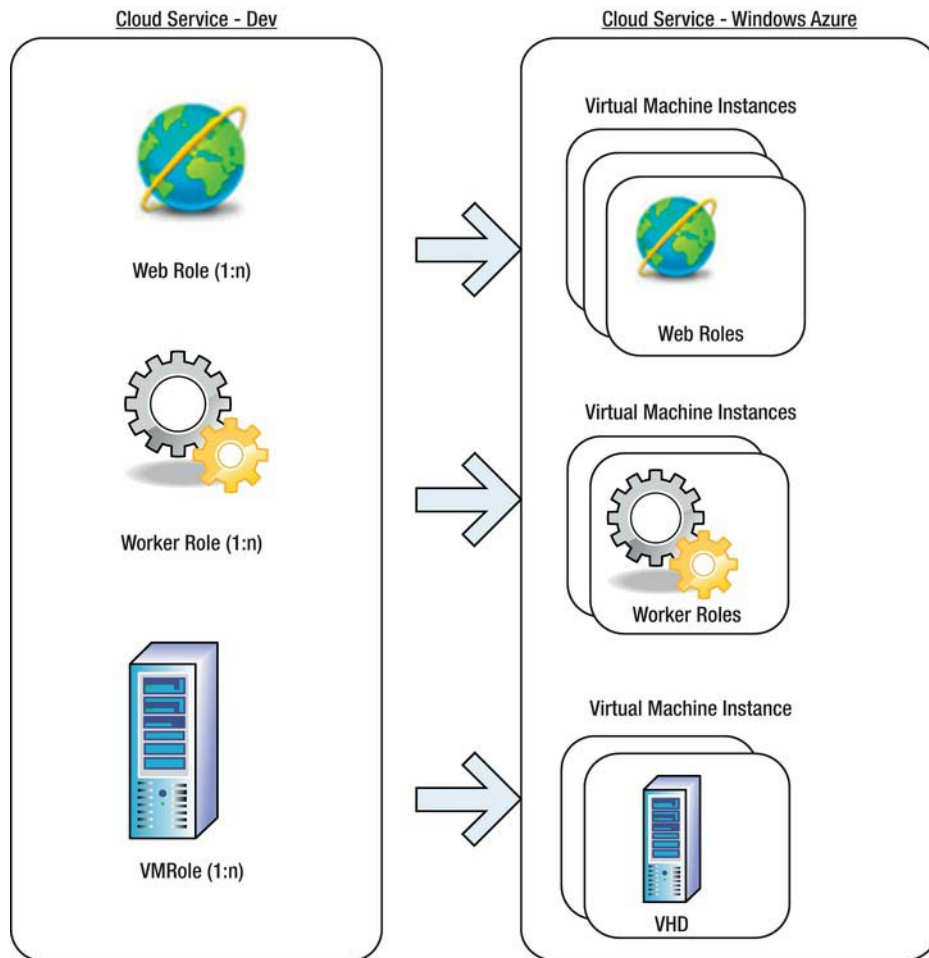


Figure 2-2. Windows Azure roles and cloud service

Each role can be deployed to Windows Azure as multiple instances, and can also be scaled up and down dynamically via the service management API.

Upgrade Domains and Fault Domains

The Service Level Agreement (SLA) for Windows Azure states, “For compute, we guarantee that when you deploy two or more role instances in **different fault and upgrade domains** your Internet facing roles will have external connectivity at least 99.95% of the time.”¹

¹ Windows Azure SLA <http://www.microsoft.com/windowsazure/sla/>

One of the value propositions of Windows Azure is the ability to automatically update instances in your service. But, if the update requires a server reboot, it directly affects the availability of your service.

Upgrade Domains are a logical separation of your role instances, ensuring your service does not go down during upgrades. There are five upgrade domains by default, but you have control over the number of upgrade domains your application will use through the service model definition.

Fault domains are logical separation of your role instances for avoiding single point of failure due to software and hardware failures. The Fabric Controller always deploys more than one instances of your service in two separate fault domains. Therefore, the SLA specifically requires you to have at least two instances of a role for 99.95% availability. The SLA also states, “Additionally, we will monitor all of your individual role instances and guarantee that 99.9% of the time we will detect when a role instance’s process is not running and initiate corrective action.” You don’t have any control over the number of Fault Domains and the assignment process. The Fabric Controller internally defines a unit of failure and never deploys two instances of the same role in the same unit of failure. Therefore, if there is a hardware failure in a rack on which one instance is running, the second instance still remains available. After detecting failure, the Fabric Controller then restarts a new instance in a third fault domain. All this happens automatically without any user intervention. If there are three instances of the role: the Fabric Controller decides which Fault Domain to place these instances. It may even place two instances in the same Fault Domain and the third in another. This is not an ideal configuration, but it may occur. By default, your service gets two Fault Domains, therefore the likelihood of the first and the third instance running in the same Fault Domain is high. Windows Azure SDK provides you with a method call to get the Upgrade Domain and the Fault Domain of the role instance. I have covered this later in the chapter. The Fabric Controller also makes sure that one Upgrade Domain spans more than one Fault Domain as shown in Figure 2-3.

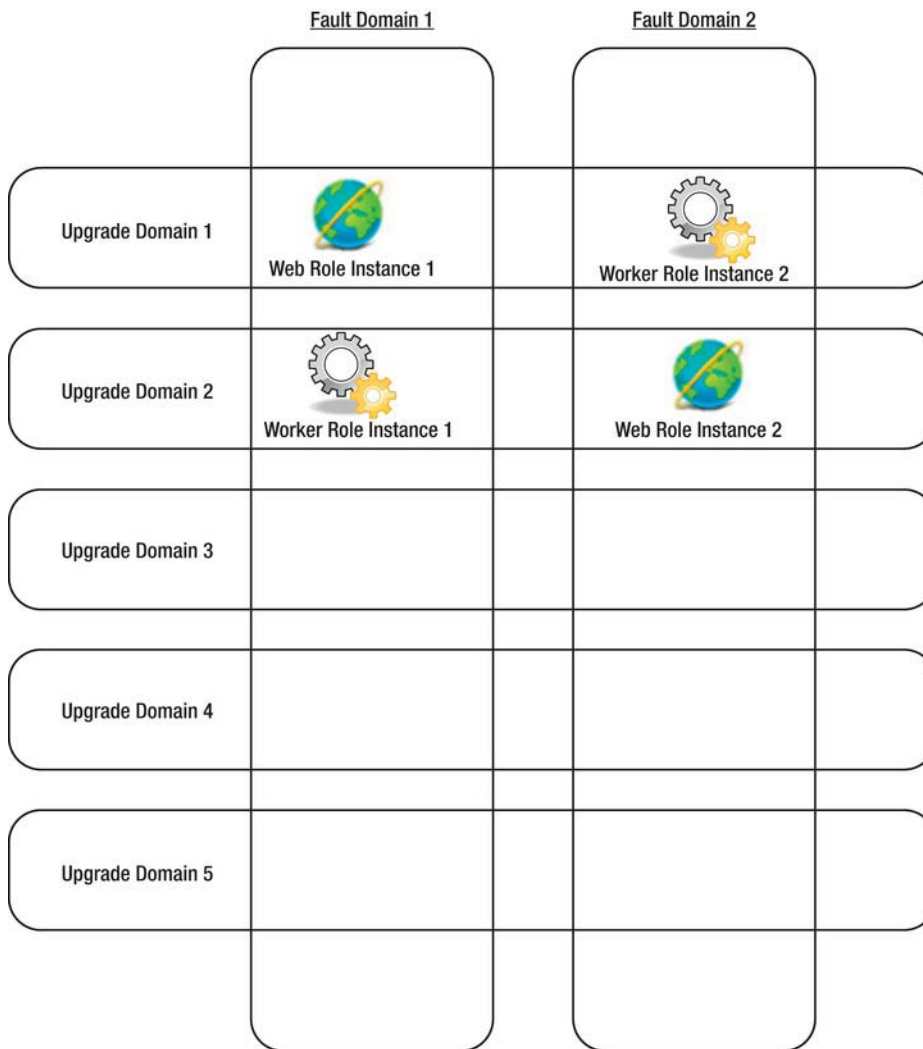


Figure 2-3. Upgrade Domain and Fault Domain

Understanding Domains in the Application Context

To further understand the concept of Upgrade and Fault Domains, let's take an example of a Windows Azure cloud service named "MyFirstService" that has only one web role, "MyWebRole." Initially, let's assume that MyWebRole has two instances, and that you are using two Upgrade Domains. The Fabric Controller will deploy each instance of MyWebRole into different Upgrade Domains and different Fault Domains as shown in Figure 2-4.

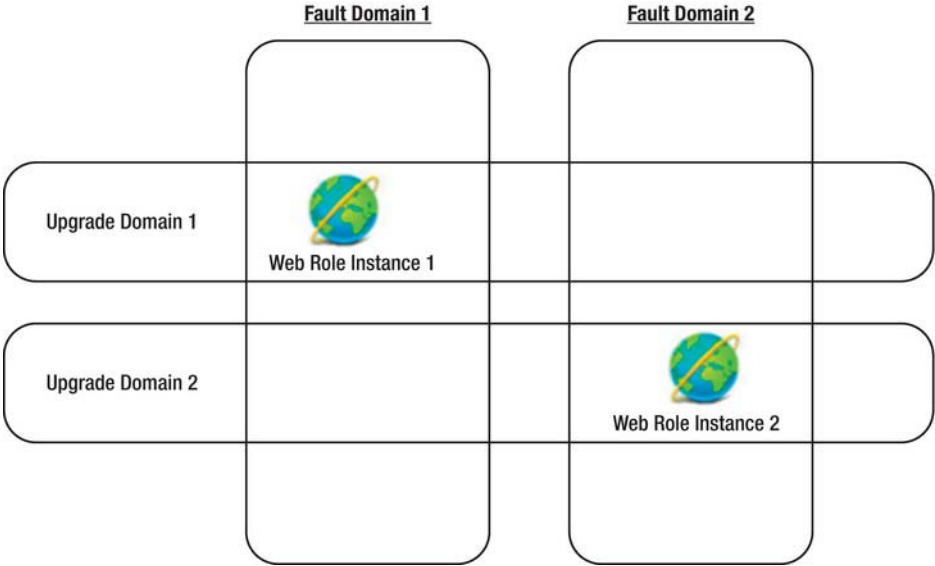


Figure 2-4. Two Web role instances in Upgrade and Fault Domains

Now, if you increase instances to three and add one more upgrade domain, then even though the Fabric Controller may deploy three instances in three different Upgrade Domains, there is no guarantee that the third instance will be deployed in a different Fault Domain. It may deploy the third instance to the same Fault Domain as the second instance, as shown in Figure 2-5.

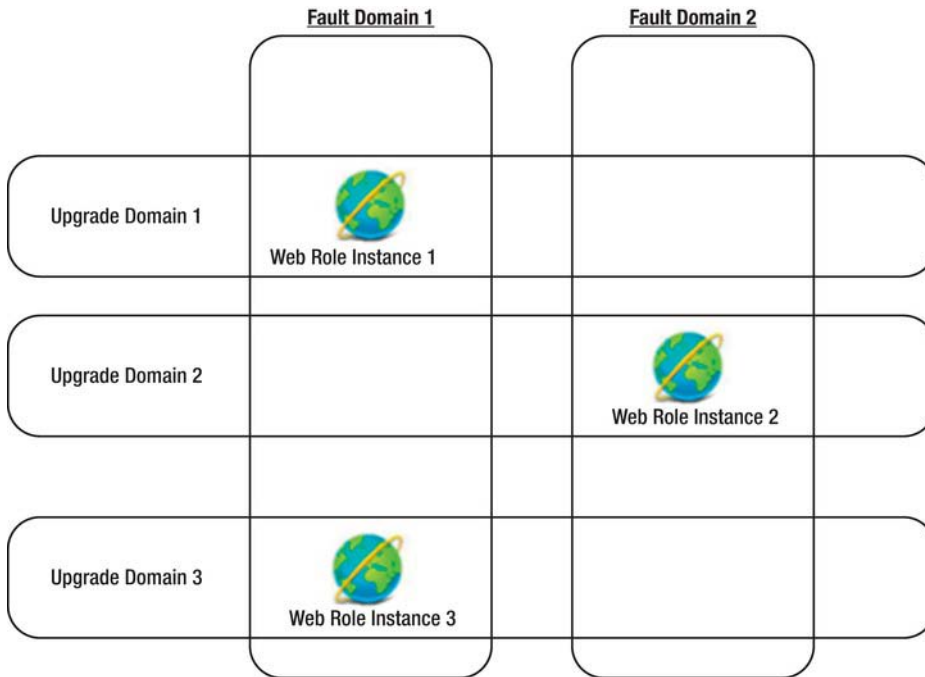


Figure 2-5. Three Web role instances in Upgrade and Fault Domain

While designing highly available services, it is important to understand the functioning of Upgrade and Fault Domains and to deploy the services with appropriate numbers of Upgrade domains for avoiding any downtime. Windows Azure SDK provides you with method calls to find out the Upgrade and Fault Domain numbers in which a role instance is running. I cover these later in this chapter.

Compute Service Security

When you run services in someone else's datacenter, there is always a concern about the security of application and data. In Compute, the instances of your service run in dedicated virtual machines. These virtual machines are managed by the hypervisor. Your deployed service runs as a least privileged account in the dedicated virtual machine. The root virtual machine is trusted by the hypervisor, but the guest virtual machine, in which your service runs, is not trusted. The root virtual machine and the hypervisor also have network packet filters that prevent unauthorized traffic to the virtual machines running the service. The isolation of virtual machines from one another is managed by the hypervisor and does not depend on Windows security. For avoiding side channel attacks, each virtual machine is isolated into a separate core. Figure 2-6 illustrates the virtual machine isolation and some security features of the compute service.

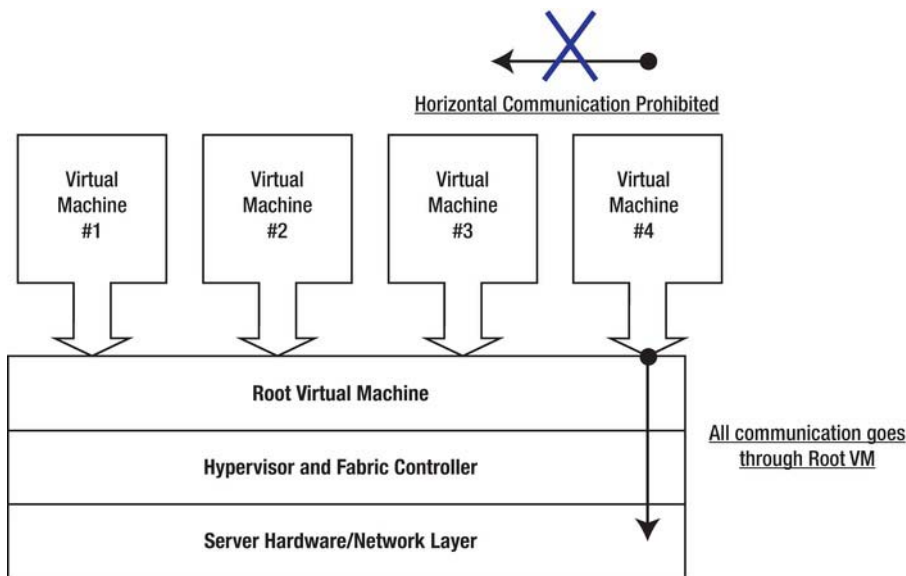


Figure 2-6. Compute service security.

The commonly exposed attack surface for a Web role is the external endpoint. Therefore, it is important to secure the external endpoints exposed by your service using secure channels like https or transport/message security in case of web services. You still have to manage any denial of service attacks in your application if they are not detected by the Windows Azure platform infrastructure.

Developing Windows Azure Services

Windows Azure and Windows Azure AppFabric have separate software development kits (SDKs), but Visual Studio and the .NET Framework are the common programming tools used for building applications for all the Windows Azure components.

■ **Note** If you are a Java or a PHP developer, you can also build services in Windows Azure. Check out the following web sites:

www.interoperabilitybridges.com/projects/windows-azure-tools-for-eclipse.

www.interoperabilitybridges.com/projects/windows-azure-sdk-for-java

www.windowsazure4j.org

Windows Azure SDK has a local development fabric that simulates the cloud environment at a miniature scale. Developers can utilize their existing .NET development skills for developing services for Windows Azure platform. The development fabric gives you a local environment to test your code. The development fabric is not meant for running any production applications.

Windows Azure API Structure

Windows Azure SDK provides a set of APIs to complement the core services offered by Windows Azure. These APIs are installed as part of Windows Azure SDK, and can be used locally for developing Windows Azure applications. The `Microsoft.WindowsAzure.ServiceRuntime` assembly and namespace consists of classes used for developing applications in the compute service. The `Microsoft.WindowsAzure.Diagnostics` namespace consists of classes used for diagnostics and logging in the compute service.

The `Microsoft.WindowsAzure.StorageClient` assembly and namespace consists of classes used for developing applications to interact with the storage service. The assembly makes REST calls to the storage service REST interface.

The service management API is exposed as a REST interface, and the `csmanage.exe` application in Windows Azure code samples (<http://code.msdn.microsoft.com/windowsazuresamples>) can be used to call the service management APIs.

Developer Environment

The development environment of Windows Azure consists of two main components: Windows Azure Tools for Visual Studio and the Windows Azure SDK. In this section, I will cover these in detail.

Windows Azure Tools for Visual Studio

Windows Azure Tools for Visual Studio is a Visual Studio extension supporting Windows Azure development. You can download it from the Azure SDK web site at www.microsoft.com/azure/sdk.msp.

Visual Studio Project Types

The Windows Azure Tools for Visual Studio creates a project type named Cloud Service containing project templates for Web role and Worker role. After you install Windows Azure Tools for Visual Studio, open Visual Studio and create a new Project by selecting File ► New ► Project. Figure 2-7 shows the New Project Dialog box.

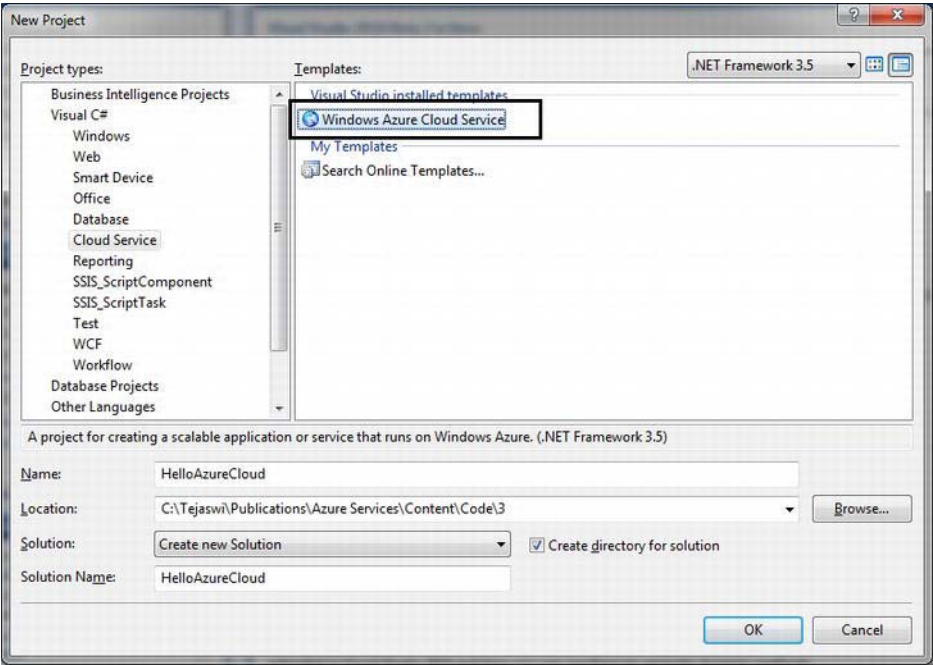


Figure 2-7. New project

The Windows Azure Cloud Service template defines the cloud service project. Click OK to choose from the available roles (see Figure 2-8).

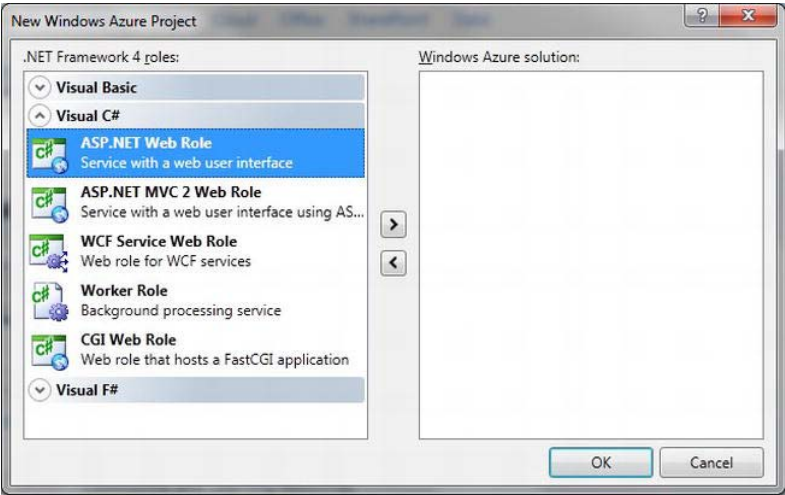


Figure 2-8 Cloud service roles

The available cloud service roles are as follows:

- **ASP.NET and ASP.NET MVC Web role:** As the name suggests, this role consists of an ASP.NET project. You can build any ASP.NET compatible project for deploying to the cloud.
- **WCF Service Web role:** This role consists of a WCF project with basic HTTP binding.
- **Worker role:** The Worker role project is a background process application. It is analogous to a Windows service. A Worker role has start and stop methods in its super class and can expose internal and external endpoints for direct access.
- **CGI Web role:** The CGI Web role is a FastCGI-enabled Web role. It does not consist of a Cloud Service project.

Choose the roles you want, as shown in Figure 2-9. I have selected a Web role, a WCF Web role, and a Worker role.

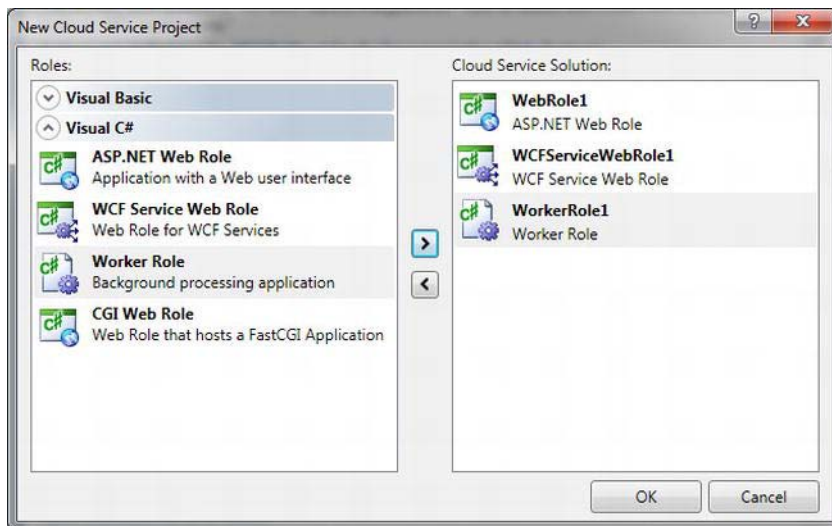


Figure 2-9. Selected roles

Click OK to create the cloud service project, as shown in Figure 2-10.

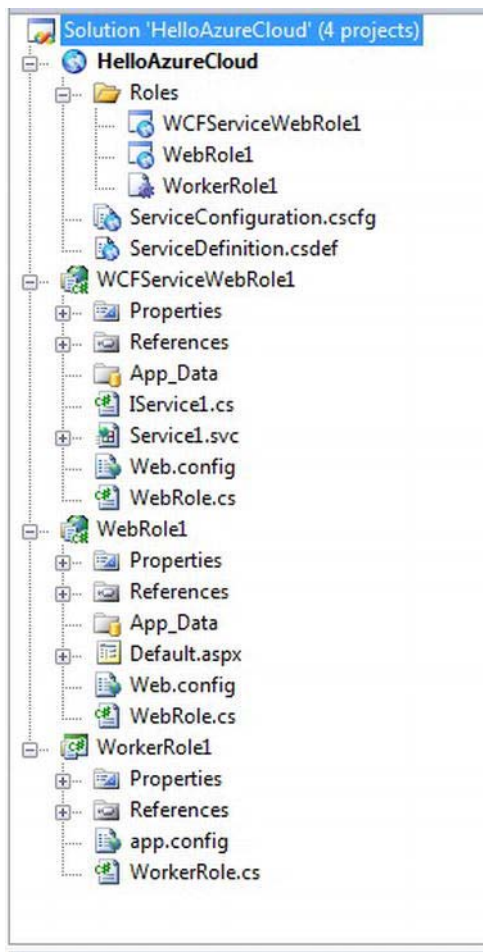


Figure 2-10. Empty cloud service project

In Figure 2-10, the HelloAzureCloud cloud service project holds references in the Roles subfolder to all the role projects in the solution. The cloud service project also contains `ServiceDefinition.csdef` and `ServiceConfiguration.cscfg` files that define the configuration settings for all the roles in the cloud service.

The WCF service Web role project includes a sample service and its associated configuration in the `web.config` file. The `WebRole.cs` file implements the start and configuration changing events fired by the Windows Azure platform. This file is created for all the roles with default start and configuration changing event handlers. You can handle additional events like `StatusCheck` and `Stopping` depending on your application needs. The `WebRole` class inherits the `RoleEntryPoint` class from the `Microsoft.WindowsAzure.ServiceRuntime` namespace. The `WebRole.cs` class is the background class that instantiates a background processing object at runtime. The Web Role does not need to have this class, but it is beneficial for performing background and startup tasks within the Web role.

The ASP.NET Web role project consists of a `Default.aspx` file and its associated code-behind and `web.config` file.

■ **Caution** Even though Windows Azure supports native code execution, the code still runs with Windows user, not administrator, privileges. Therefore, some WIN32 APIs that require system administrator privileges will not be accessible

Finally, the Worker role project consists of `WorkerRole.cs` file and its associated `app.config` file. In addition to inheriting the `RoleEntryPoint` class, it also overrides the `Run()` method in which you add your continuous processing logic. A Worker role class must inherit from the `Microsoft.WindowsAzure.ServiceRuntime.RoleEntryPoint` class. `RoleEntryPoint` is an abstract class that defines functions for initializing, starting and stopping the Worker role service. A Worker role can stop either when it is redeployed to another server or when you have executed the Stop action from the Windows Azure developer portal. Because a Worker role is not designed to have any external interface by default, it does not contain any ASP.NET or WCF files.

In summary, the cloud service defined in this project consists of a WCF service, an ASP.NET web application, and a Worker role service. The entire package constitutes a Windows Azure cloud service.

■ **Note** In the interest of keeping this book conceptual, I will not be covering FastCGI applications.

Role Settings and Configuration

In the cloud service project, you can configure each role's settings by double-clicking the role reference in the `Roles` subdirectory of the cloud service project. Figure 2-11 shows the role settings page in Visual Studio.

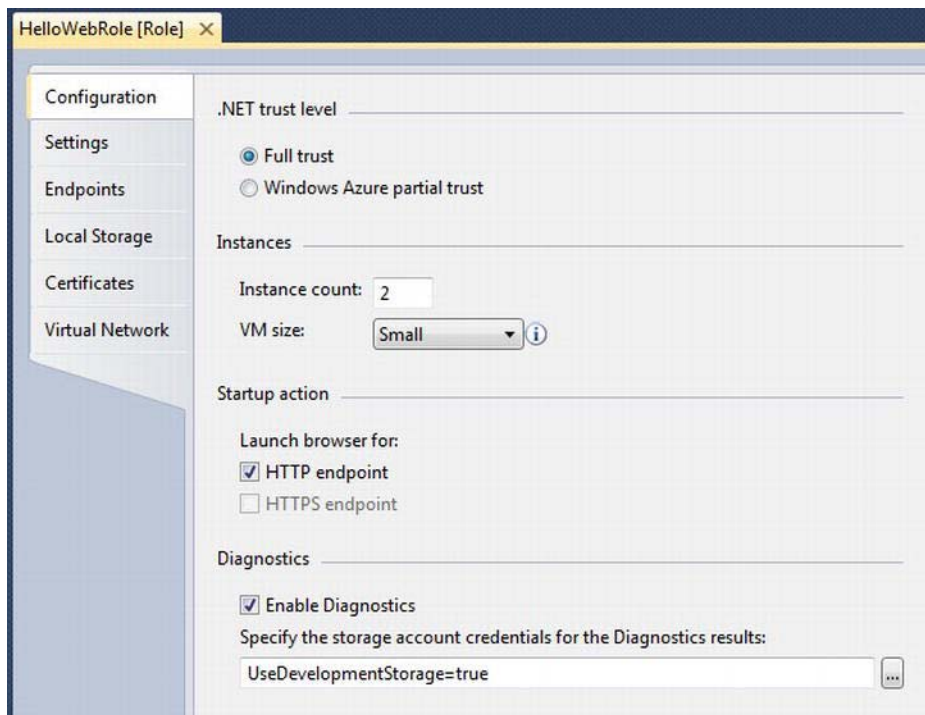


Figure 2-11. Role settings (the default is Configuration)

■ **Note** The Role Settings UI actually updates the `ServiceDefinition.csdef` and `ServiceConfiguration.cscfg` files behind the scenes. You can achieve the same effect by directly modifying these XML files. The concept is important if you want to dynamically modify these files for auto-scaling of auto-deployments.

The role settings page has six tabs: Configuration, Settings, Endpoints, Local Storage, Certificates, and Virtual Network.

- **Configuration:** The Configuration tab is selected by default and displays the following configuration options:

- **.NET Trust Level:** The .NET Trust Level specifies the trust level under which this particular role runs. The two options are Full Trust and Windows Azure Partial Trust. Full Trust options gives the role privileges to access certain machine resources and execute native code. Even in Full Trust, the role still runs in the standard Windows Azure user's context and not the administrator's context. In the Partial Trust option, the role runs in a partially trusted environment and does not have privileges for accessing machine resources and native code execution.
- **Instances:** The instance count defines the number of instances of each role you want to run in the cloud. For example, you can run two instances of ASP.NET Web role and one instance of the Worker role for background processing. The two instances of ASP.NET Web role will give you automatic load-balancing across the instances. By default, all the roles run as single instance. This option gives you the ability to scale-up and scale-down your role instances on demand.

The VM size option gives you the ability to choose from a list of virtual machines preconfigured in the Windows Azure virtual machine pool. You can choose from the following list of predefined virtual machines depending on your deployment needs:

- **Extra small:** 1x1.0 GHz core processor, 768GB RAM, 20GB hard disk (low IO)
- **Small:** 1x1.6 GHz core processor, 1.75GB RAM, 225GB hard disk (moderate IO)
- **Medium:** 2 core processors, 3.5GB RAM, 500GB hard disk (high IO)
- **Large:** 4 core processors, 7GB RAM, 1000GB hard disk (high IO)
- **Extra large:** 8 core processors, 15GB RAM, 2000GB hard disk (high IO)

■ **Note** For any operations that require moderate to high I/O, I recommend using VM sizes of medium or higher. The I/O may include disk I/O, network I/O or even memory I/O like caching. I have also seen specific situations where 16 small VMs performed better than 2 extra large instances.

The Web roles have a startup action that defines the endpoint on which the browser should launch. This is not a cloud service setting, but a project setting for launching the Web role in the development fabric. The HTTPS checkbox will be disabled until you add an HTTPS endpoint to the role.

In the Diagnostics section, you can enable or disable the diagnostics capabilities for the role and also specify the destination of the diagnostics logs. By default, the destination points to the development storage: "UseDevelopmentStorage=true." If you have a Windows Azure storage service created, you can add the connection string pointing to that storage service. The format for the storage service connection string is

DefaultEndpointsProtocol=https;AccountName=[Your storage service account name];AccountKey=[You can get this from the management portal]

- **Settings:** The Settings tab, shown in Figure 2-12, defines any custom settings you can add to the role configuration.

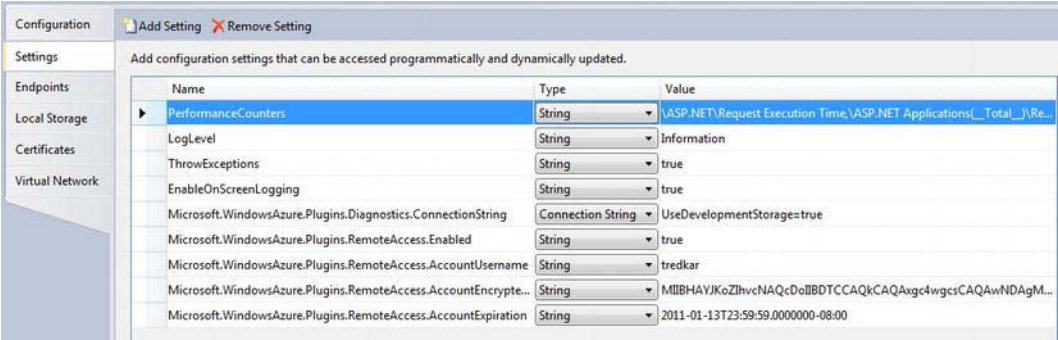


Figure 2-12. Settings tab

These custom name-value pairs are analogous to the name-value `appSettings` in an `app.config` or `web.config` file. You can retrieve the values of these settings in your code by calling the `RoleEnvironment.GetConfigurationSettingValue`. If you enable diagnostics or remote desktop access plug-ins, custom settings are created for these plug-ins in the Settings section. You can add your custom settings like the database connection string to this section.

- **Endpoints:** The Endpoints tab contains endpoints your role will expose when it is deployed. There is a hard limit of five endpoints you can expose from a role. Figure 2-13 shows the Endpoints tab for a Web role and a Worker role respectively.

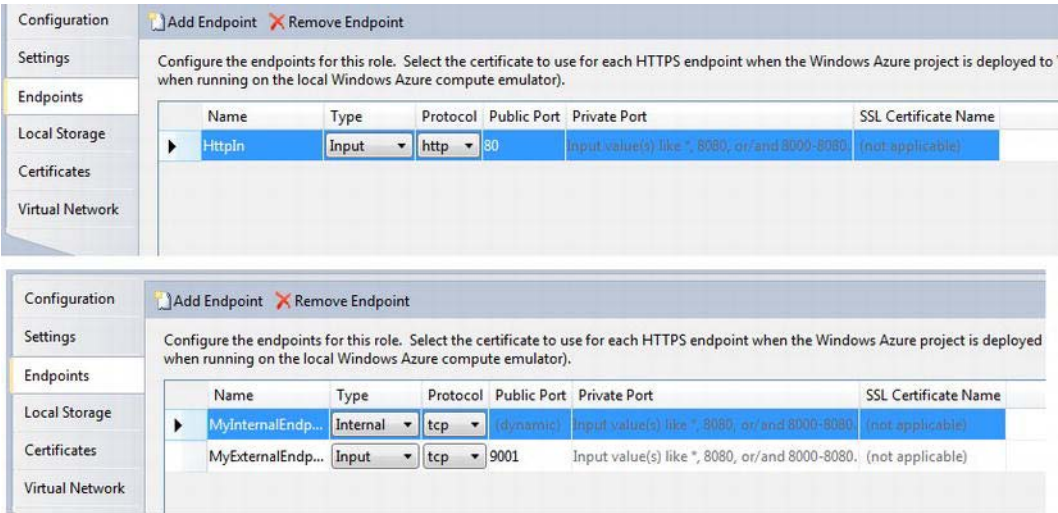


Figure 2-13. Endpoints tab

The roles can have Input Endpoints and an internal endpoint. Input endpoints are exposed externally, whereas the Internal endpoints are exposed internally within the cloud service role instances. The port number defines the port your will use while accessing the endpoint. In case of an HTTPS

endpoint, you can upload the X.509 certificate for accessing the web page or service using an HTTPS encrypted connection.

The internal endpoint is accessible to other roles within the cloud service. For example, a Web role can get a reference to the internal endpoint of a worker role in the same cloud service for making web service method calls to it.

By default, a Worker role has no defined endpoints like a Web role, because it is intended to be used as a background process. To define an endpoint, you have to add one to the list and select its type (input or internal), protocol (tcp, http, and, https), port, and, optionally, an SSL certificate name.

- **Local Storage:** The Local Storage tab defines local directories that will be created on the server machine of the role for storing files locally. Figure 2-14 shows the settings on the Local Storage tab.

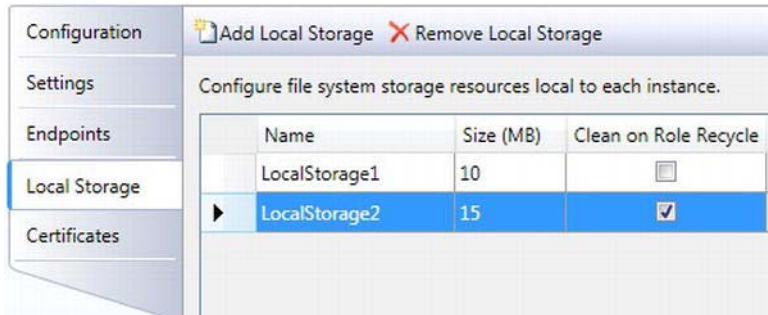


Figure 2-14. Local storage

The name of the local storage will be the names of directories created on the server. The size column defines the maximum size of the folder contents and the “Clean on Role Recycle” column defines whether you want the contents of the directory cleaned up when a role recycles. You can use this option for creating sticky storage for maintaining state of the role across reboots and failures. The local storage uses the overall hard disk capacity available for the role instance and therefore you have to make sure you don’t exceed the maximum capacity available for the virtual machine to avoid any errors.

- **Certificates:** The Certificates tab is used for referencing the certificates in your role. You can use the certificates referenced here for configuring HTTPS in your Web role and also for remote desktop connections and virtual network. The certificates listed here can be referenced throughout the role configuration. At the time of this writing, you still had to use the Windows Azure management portal or the service management API for uploading the certificate to the server and then reference the certificate in the settings as shown in Figure 2-15.

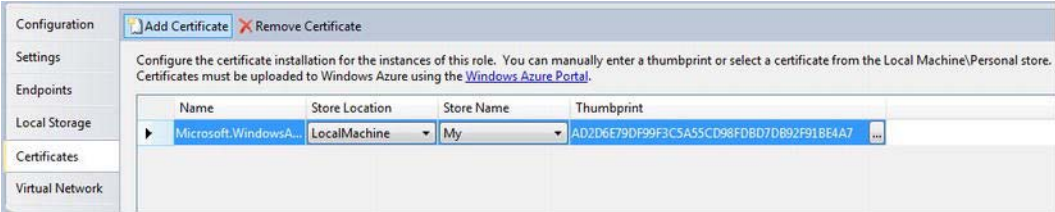


Figure 2-15. Certificate configuration

- **Virtual Network:** The Virtual Network tab allows you create a virtual network between the Windows Azure role instances and your on-premises servers by adding a Windows Azure Connect activation token. You acquire the token from the Windows Azure Management portal and copy it in this section. Once you activate the Virtual Network, Visual Studio creates several settings elements in `ServiceConfiguration.cscfg` for configuring the network, as shown in Figure 2-16.

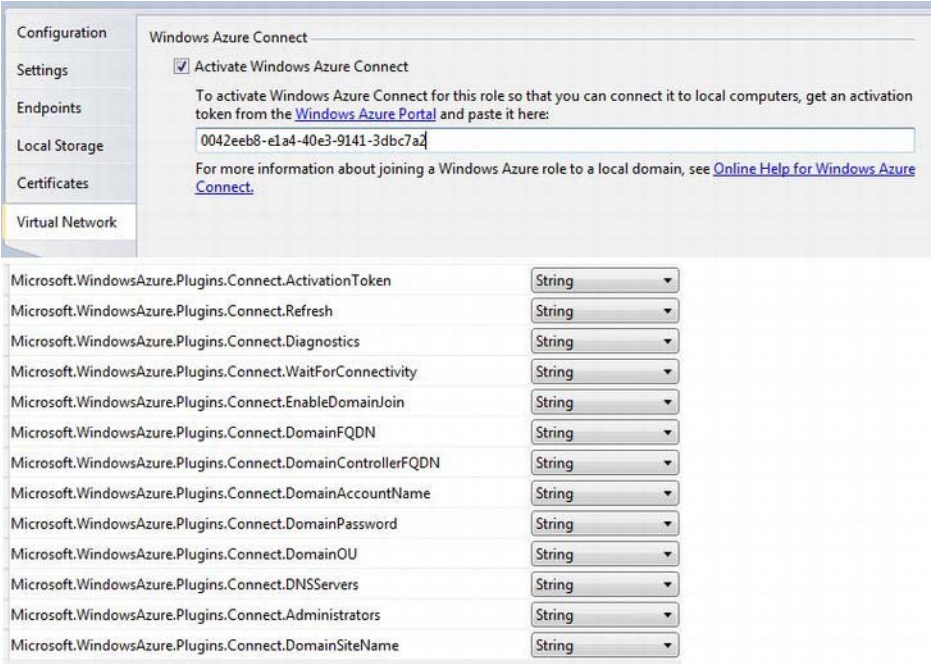


Figure 2-16. Virtual network

The settings configuration elements allow you to not only configure point-to-point connectivity, but also join the Windows Azure role instances to your on-premises domain. I cover Windows Azure Connect in the next chapter.

Visual Studio Project Actions

Once you have created a Windows Azure cloud service project, you can work with the cloud service roles, work with storage services, or work on the debug and deployment of the cloud service.

Working with Cloud Service Roles

You can associate an existing Web role or a Worker role from a solution to the cloud service project, or create a new role by right-clicking on the Roles subdirectory and selecting Add, as shown in Figure 2-17.

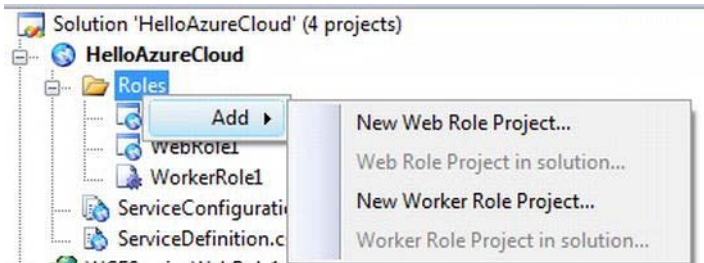


Figure 2-17. Adding associate roles to cloud service

By selecting New Web Role or New Worker Role project, you can create a new Web role project in the solution that is associated with the cloud service project. By selecting a Web role or Worker role project in the solution, you can associate an existing project in the solution to the cloud service project. Figure 2-18 shows option for adding a new role to the existing cloud service.

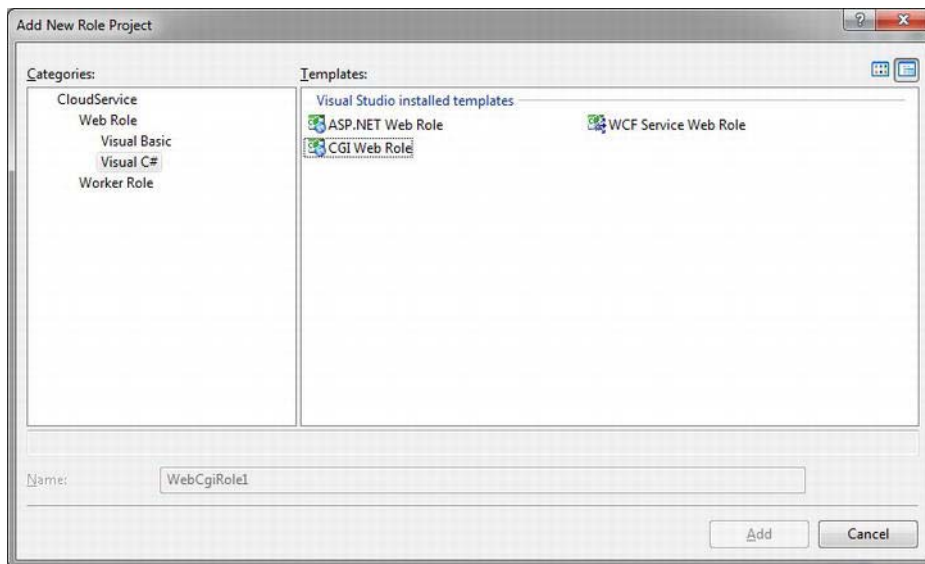


Figure 2-18. Adding new roles

Working with Storage Services

The Windows Azure development fabric includes a local storage environment that resembles the cloud storage service. It has development-specific blob, queue, and table services that simulate the ones in the Windows Azure cloud. These services depend on SQL Server 2005 or 2008 database. So, you need to have SQL Server 2005 or 2008 installed on your machine to work with storage services development environment (also called Development Storage).

To start the development storage:

- Select Start ► All Programs ► Windows Azure SDK ► Storage Emulator, as shown in Figure 2-19.

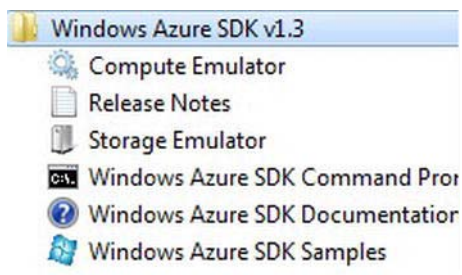


Figure 2-19. Development storage

When you debug your service within Visual Studio, it starts the development storage, which you can access by right-clicking the Windows Azure system tray icon and selecting Show Storage Emulator UI. Figures 2-20 and 2-21 illustrate the system tray options and the development storage user interface. Figure 2-21 shows the local machine endpoints for each storage service.

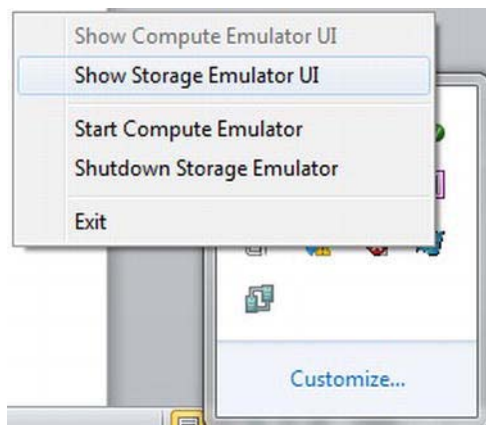


Figure 2-20. Windows Azure system tray options

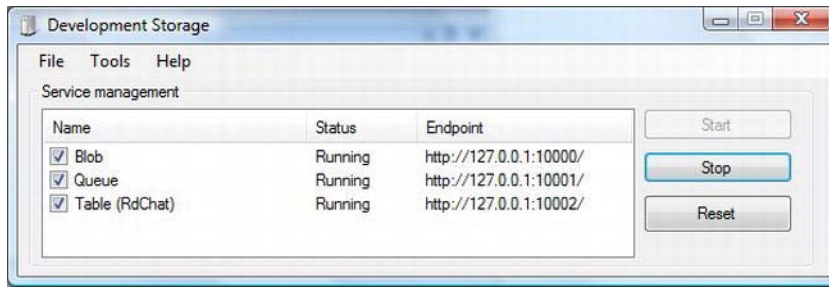


Figure 2-21. Developer storage user interface

Debugging in Visual Studio

In the Windows Azure Cloud environment, no direct debugging is available. You have two options: IntelliTrace and logging. IntelliTrace is available only in Visual Studio Ultimate, but it's worth the investment when developing for Windows Azure. IntelliTrace records a series of events and allows you to playback that recording with the existing code on your machine. This suits perfectly in cloud environments, because you cannot directly debug services running Windows Azure from Visual Studio. I have covered IntelliTrace later in the chapter. In the development fabric, you can debug by adding breakpoints in the code and by viewing the logging information in the development fabric user interface. As with any .NET application, Visual Studio attaches the debugger to the application when run in debug mode in the development fabric. The debugger will break to the breakpoint set in the Web and Worker role projects. In the Windows Azure cloud, Visual Studio debugging environment is not available, so one of the best options is to log. Logs can be automatically transferred to Windows Azure storage and then collected via tools. I will discuss diagnostics and logging later in this chapter. Figure 2-22 illustrates the Development Fabric UI used for logging.

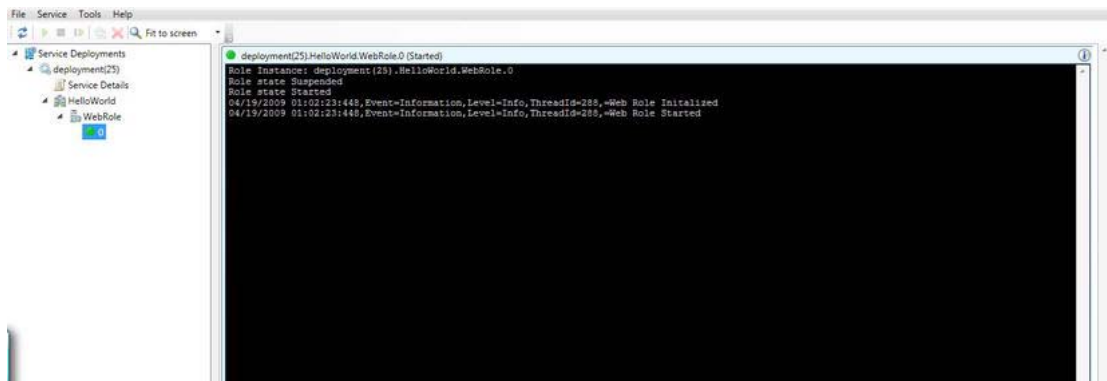


Figure 2-22. Development Fabric UI used for logging

■ **Tip** I recommend inserting logging statements to the Windows Azure application right from the beginning. This way, you can debug the application in the development fabric as well as in the Windows Azure cloud without making any code changes.

To enable native code debugging in a Web role project, right-click the Web role project, select Properties, go to the Web tab, and select the Native Code checkbox in the Debuggers section, as shown in Figure 2-23.

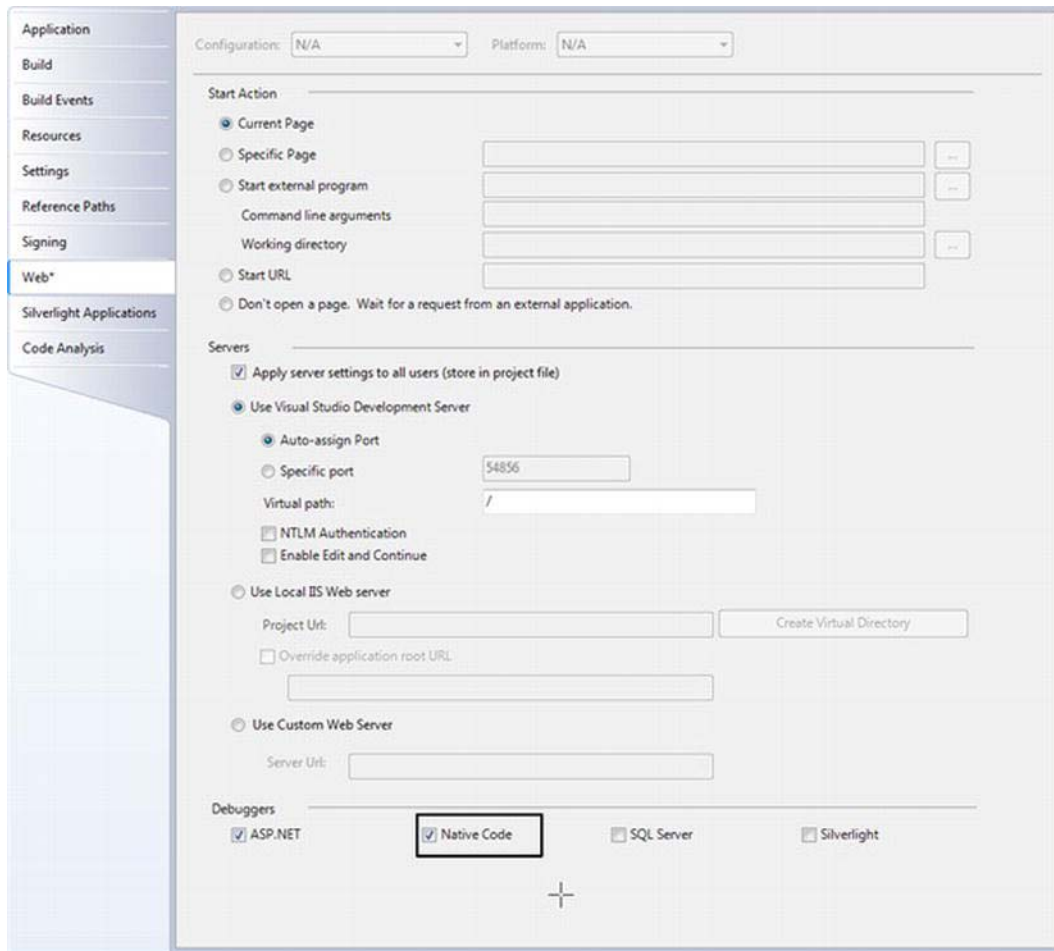


Figure 2-23. Web role unmanaged code debugging

To enable native code debugging in a Worker role project, right-click the Worker role project, select Properties, go to the Debug tab, and select the “Enable unmanaged code debugging” checkbox, as shown in Figure 2-24.

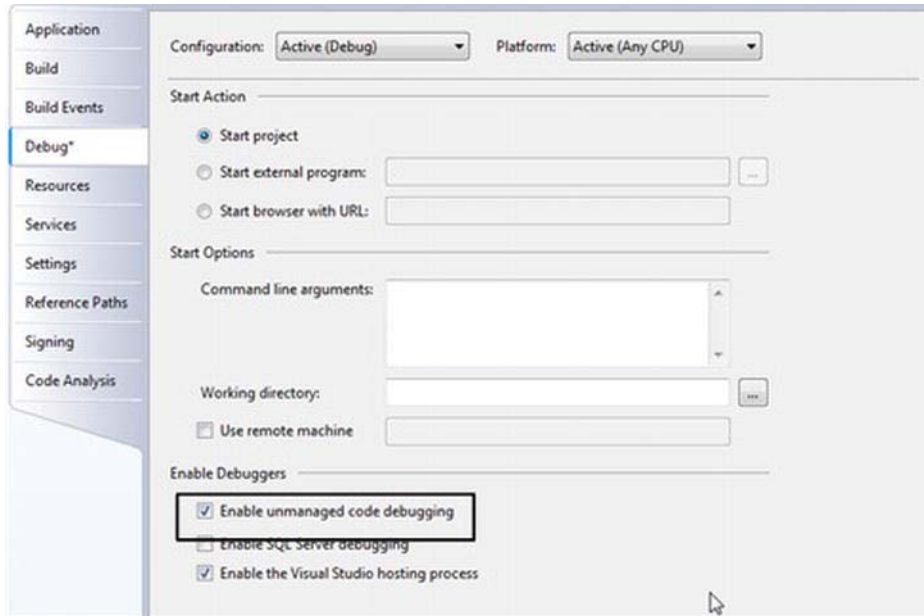


Figure 2-24. Worker role unmanaged code debugging

Packaging the Service

To deploy the Windows Azure cloud service in the cloud, you have to package it into a .cspkg file containing all the assemblies and components, and upload the package to Windows Azure developer portal. To package a service, right-click the cloud service project, and select Publish, as shown in Figure 2-25.

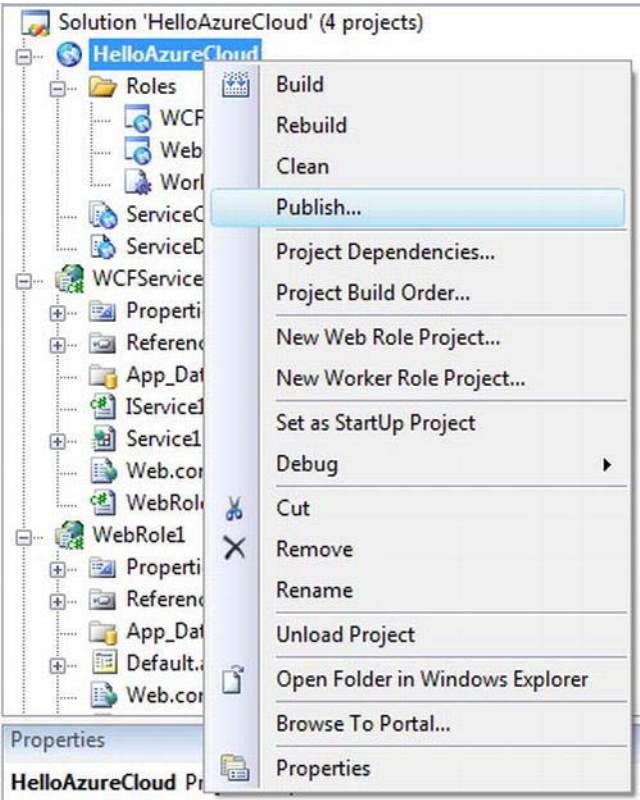


Figure 2-25. Packaging a Windows Azure service

When you select Publish, Visual Studio opens up Deploy Windows Azure project dialog box, as shown in Figure 2-26.

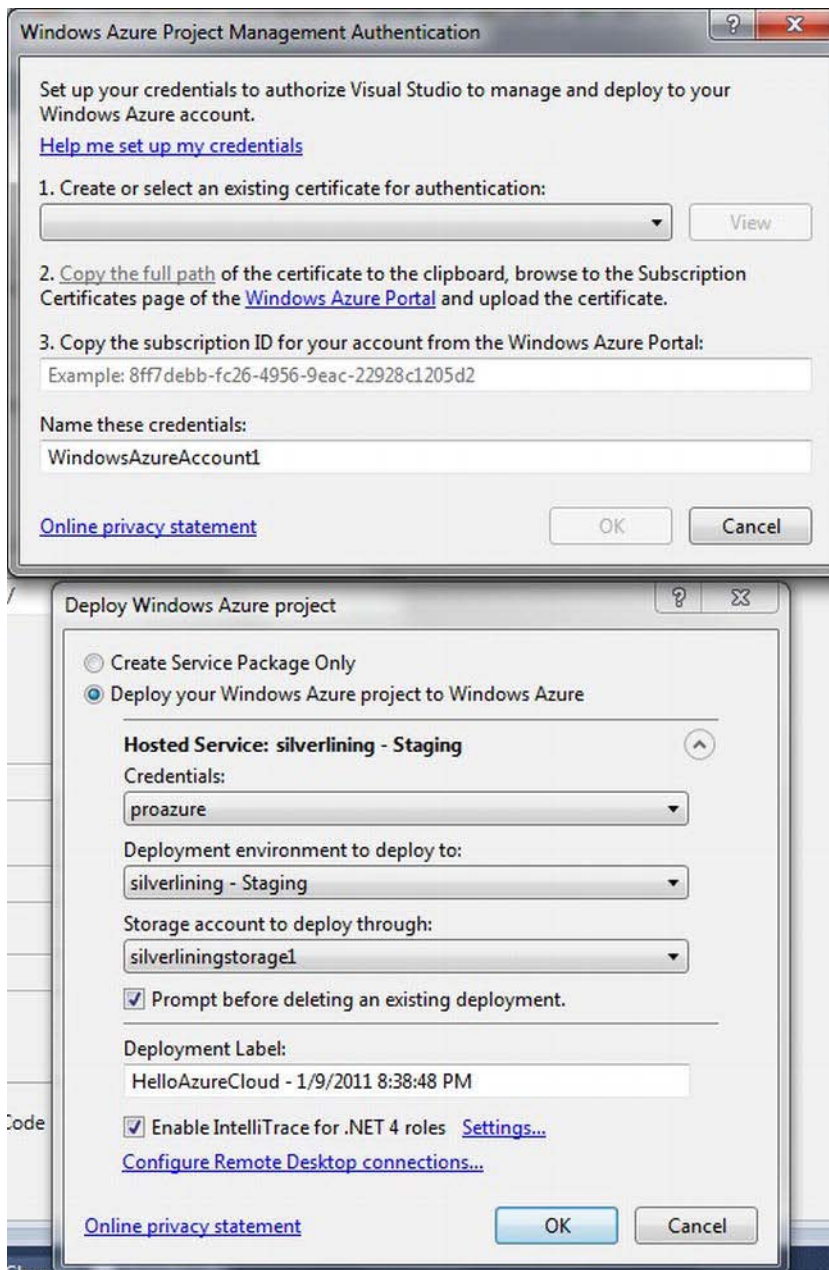


Figure 2-26. Deploy Windows Azure

Visual Studio lets you deploy the project directly from Visual Studio. Though it is not recommended to deploy production cloud services from a developer's desktop, this is a handy feature for deploying cloud services in testing and staging environments. To deploy cloud services from Visual Studio, you need to create an X.509 certificate and upload it to your account in the management portal. Visual Studio uses the Windows Azure Service Management API for deploying cloud services, and X.509 certificate is required for accessing the Service Management API. Visual Studio lets you create a new certificate or choose an already existing one from the Credentials dropdown. Once you create or select a certificate, you can then provide a subscription ID of your account, which is needed by the Service Management API. Once you specify the credentials, you can choose your hosted service from the dropdown with the deployment slot (staging or production), the storage account for uploading the deployment package and a deployment label. You can also select if you want to enable IntelliTrace. When you click OK, Visual Studio opens up a Windows Azure Activity Log window, as shown in Figure 2-27, and shows displays the status of your deployment.

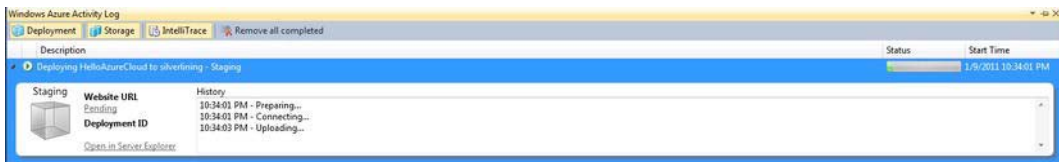


Figure 2-27. Windows Azure Activity Log

Once the deployment is complete, the activity log will display the URL of the service, as shown in Figure 2-28.

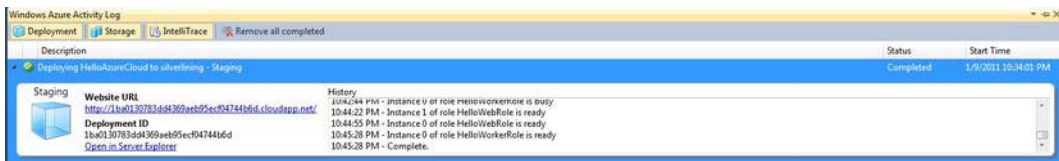


Figure 2-28. Windows Azure Activity Log on completion

On the Deploy Windows Azure Project dialog box, you also have the option to only create the service package. Visual Studio creates two files: [Service Name].cspkg and ServiceConfiguration.cscfg. The [Service Name].cspkg is the service package containing all the service components required by Windows Azure to run the service in the cloud. The .cspkg file is a zip archive, and you can explore its contents by renaming it to .zip and extracting it. The ServiceConfiguration.cscfg file is the configuration file for the service instances. It is a copy of the ServiceConfiguration.cscfg file from the cloud service project. You can deploy the .cspkg either by manually uploading to the management portal or through PowerShell scripts that use the Service Management API. In a typical software development lifecycle, the release management team will deploy the package using custom scripts that call the Service Management API.

You can also configure Remote Desktop Connections for your roles from the Deploy Windows Azure Project dialog box. As shown in Figure 2-29, the remote desktop configuration requires an X.509 certificate, username, and password for remote login, and an expiration date for the login.

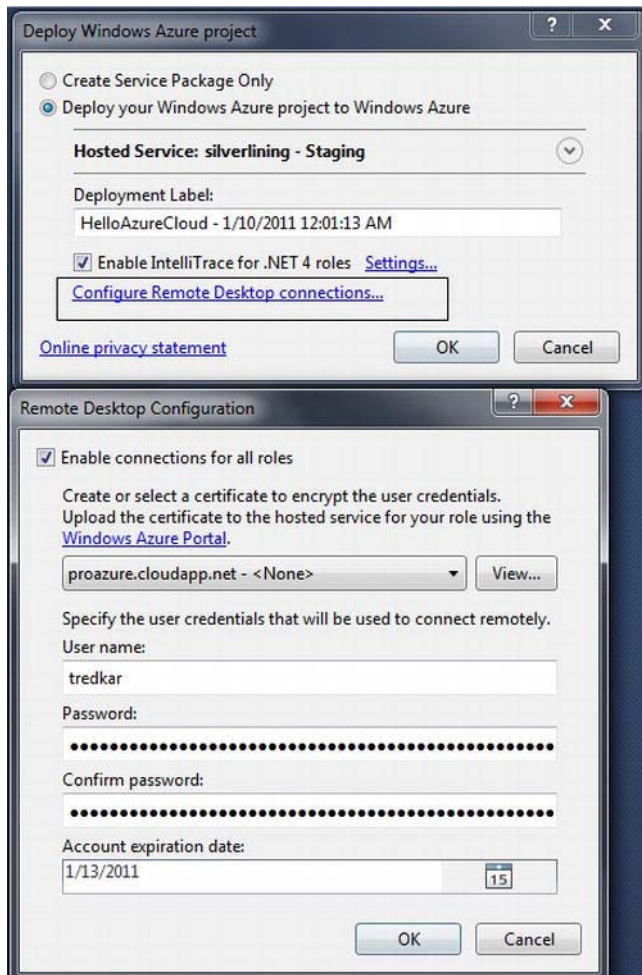


Figure 2-29. Remote desktop configuration

■ **Note** The password accepted by the Remote Desktop Connection needs to be encrypted. The procedure for encrypting the passwords is listed here <http://msdn.microsoft.com/en-us/library/gg432965.aspx>.

Windows Azure SDK Tools

The Windows Azure SDK tools are located in the directory C:\Program Files\Windows Azure SDK\v1.0\bin for a default Windows Azure installation. Table 2-2 lists the tools included in the Windows Azure SDK.

Table 2-2. Windows Azure SDK Tools

Tool	Description
CSPack.exe	This tool is used to package a service for deployment. It takes in a ServiceDefinition.csdef file and outputs a .cspkg file.
CSRun.exe	This tool deploys a service into the local development fabric. You can also control the run state of the development fabric from this tool. This tool depends on the service directory structure created by the CSPack.exe /copyonly option.
Csupload.exe	CSUpload.exe is used for uploading VHD images to the Windows Azure management portal. I have cover this tool in Chapter X.
DSInit.exe	This tool initializes the development storage environment. It is automatically called by Visual Studio and DevelopmentStorage.exe when you run a cloud application in the development fabric for the first time.

Service Models

A service model of Windows Azure cloud service consists of two main configuration files: ServiceDefinition.csdef and ServiceConfiguration.cscfg. ServiceDefinition.csdef defines the metadata and configuration settings for the service. ServiceConfiguration.cscfg sets the values of configuration settings for the runtime instance of the service. You can modify the ServiceConfiguration.cscfg while the service is running, but modification to ServiceDefinition.csdef requires an update to the service package. The overall service model defines the metadata and configuration parameters and the end state of the service. Windows Azure reads these files when deploying instances of your service in the cloud and the Fabric Controller takes appropriate deployment actions.

ServiceDefinition.csdef

The ServiceDefinition.csdef file defines the overall structure of the service. It defines the roles available to the service, the input/external endpoints, web sites, local storage, plug-in modules, and certificates. With Windows Azure SDK version 1.3, Microsoft introduced a plug-in model for Windows Azure features. For example, diagnostics, remote desktop, and Windows Azure Connect are plug-in modules specified in the <Imports> elements in Listing 2-1. It also defines the custom configuration settings for the service. The values of these configuration parameters are set in the ServiceConfiguration.cscfg file. Listing 2-1 shows the contents of a ServiceDefinition.csdef file.

Listing 2-1. ServiceDefinition.csdef

```

<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition name="HelloAzureCloud"
  xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
  <WebRole name="HelloWebRole" enableNativeCodeExecution="true">
    <Sites>
      <Site name="Web">
        <Bindings>
          <Binding name="HttpIn" endpointName="HttpIn" />
        </Bindings>
      </Site>
    </Sites>
    <LocalResources>
      <LocalStorage name="HelloAzureWorldLocalCache" sizeInMB="10" />
    </LocalResources>
    <ConfigurationSettings>
      <Setting name="PerformanceCounters" />
      <!-- This is the current logging level of the service -->
      <Setting name="LogLevel" />
      <Setting name="ThrowExceptions" />
      <Setting name="EnableOnScreenLogging" />
    </ConfigurationSettings>
    <Endpoints>
      <InputEndpoint name="HttpIn" protocol="http" port="8080" />
    </Endpoints>
    <Imports>
      <Import moduleName="Diagnostics" />
      <Import moduleName="RemoteAccess" />
    </Imports>
    <Certificates>
    </Certificates>
  </WebRole>
  <WorkerRole name="HelloWorkerRole" enableNativeCodeExecution="true">
    <Endpoints>
      <!-- Defines an internal endpoint for inter-role communication that can be used to
communicate between worker or Web role instances -->
      <InternalEndpoint name="MyInternalEndpoint" protocol="tcp" />
      <!-- This is an external endpoint that allows a role to listen on external
communication, this could be TCP, HTTP or HTTPS -->
      <InputEndpoint name="MyExternalEndpoint" port="9001" protocol="tcp" />
    </Endpoints>

    <Imports>
      <Import moduleName="Diagnostics" />
      <Import moduleName="RemoteAccess" />
      <Import moduleName="RemoteForwarder" />
      <Import moduleName="Connect" />
    </Imports>

    <ConfigurationSettings>
      <Setting name="PerformanceCounters" />

```

```

</ConfigurationSettings>
<Startup>
  <Task commandLine="Startme.cmd" executionContext="limited" taskType="simple">
  </Task>
</Startup>
</WorkerRole>
</ServiceDefinition>

```

Listing 2-1 is the service definition for the HelloAzureCloud. It has a Web role and a Worker role instance defined. The <LocalStorage> element defines the local storage space for the service role. The <ConfigurationSettings> element defines some custom settings for the service role.

■ **Note** The ServiceDefinition.csdef file of a service cannot be changed at runtime, because it defines the shape and non-changeable parameters of the service. You have to republish the service after changing its ServiceDefinition.csdef for the changes to take effect. For more details on the ServiceDefinition.csdef schema, please visit <http://msdn.microsoft.com/en-us/library/dd179395.aspx>.

Endpoints

Windows Azure roles can have two types of endpoints: internal and input. The internal endpoints are used for inter-role communications within the same cloud service, whereas the input endpoints can be accessed from anywhere. Figure 2-30 illustrates an example of internal endpoints.

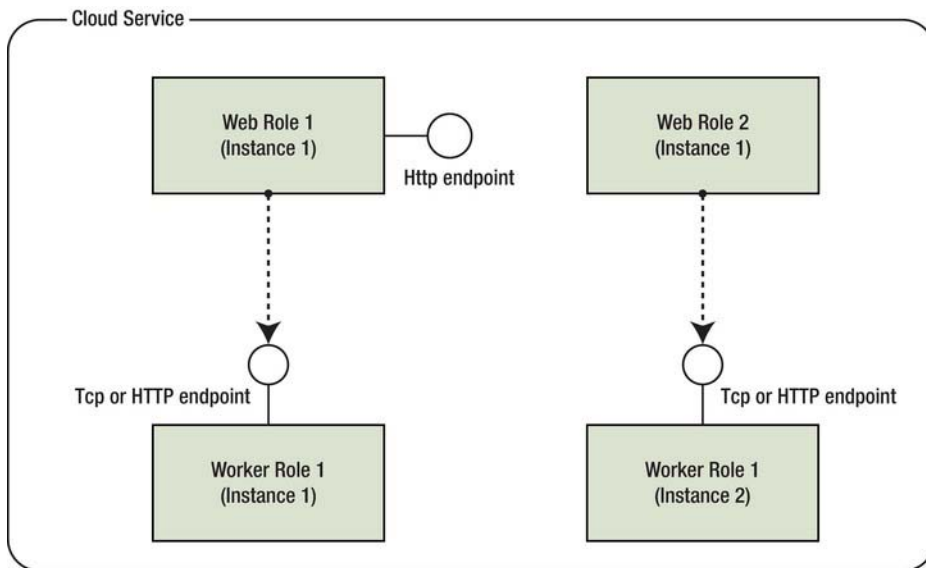


Figure 2-30. Internal endpoints for interrole communication

In Figure 2-30, there are two Web roles with one instance each and two instances of a Worker role. The Worker role exposes an endpoint that is consumed by both the Web roles. Note that each Web role can communicate with the exact instance of the Worker role. The Web role also exposes an HTTP endpoint that can be consumed by any of the roles in the cloud service. The endpoint only publishes the IP address and port of the instance; you still have to write TCP or HTTP service code for sending and receiving requests. You can get a reference to the internal endpoint of an instance as follows:

```
IPEndPoint internale = RoleEnvironment.Roles["HelloWorkerRole"].Instances[0]
.InstanceEndpoints["MyInternalEndpoint"].IPEndPoint;
```

where `HelloWorkerRole` is the name of the Worker role and `MyInternalEndpoint` is the name of the endpoint. You can get the IP address of an internal instance end point in the following manner:

```
string ipaddress = RoleEnvironment.Roles["HelloWorkerRole"].Instances[0]
.InstanceEndpoints["MyInternalEndpoint"].IPEndPoint.ToString();
```

Figure 2-31 illustrates the input endpoints of a Web role and a Worker role.

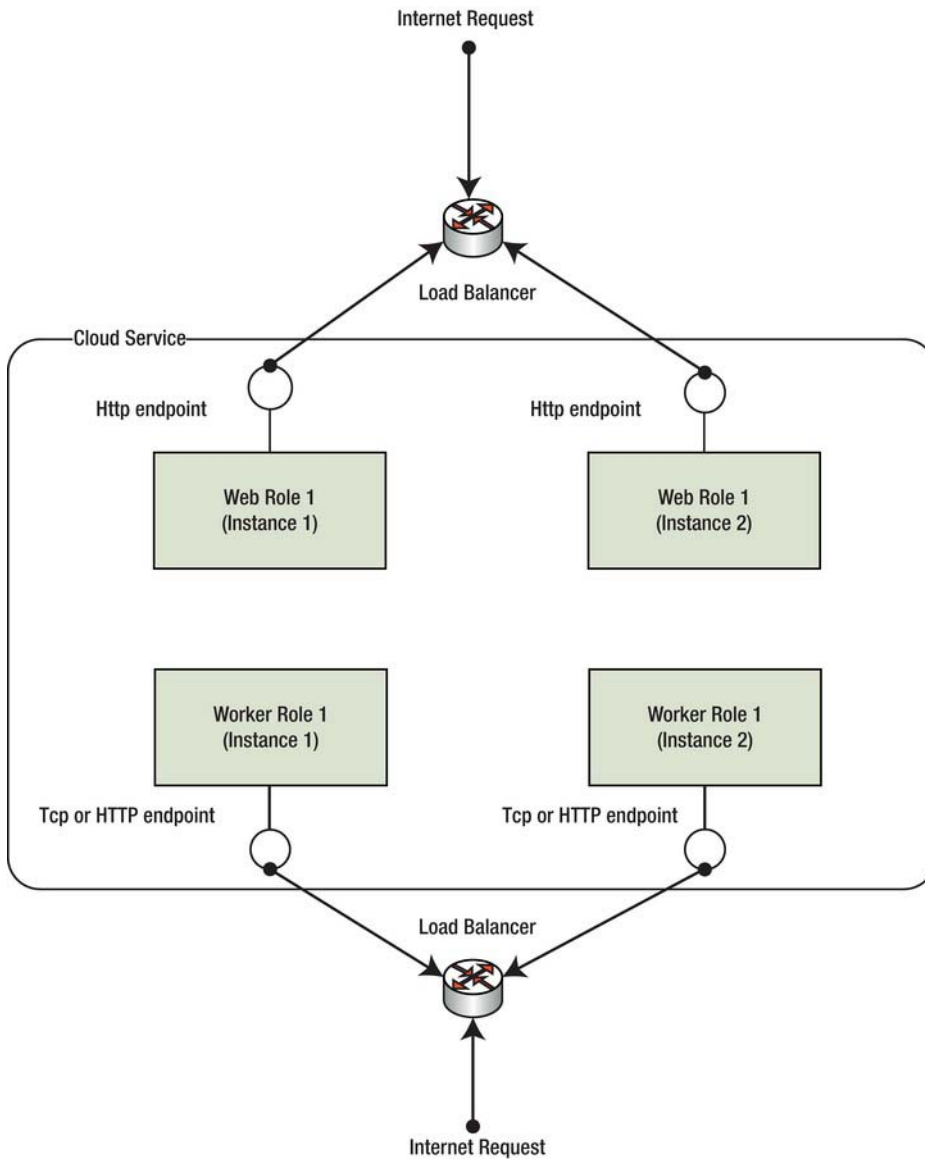


Figure 2-31. Input endpoints for external communication

The Web role instances have default HTTP input endpoints for accepting Internet requests. Windows Azure also allows Worker roles to have HTTP and TCP input endpoints for accepting connections over the Internet. Like the internal endpoint, the access to input endpoint is not limited within the cloud service; any external application can communicate with the input endpoint of the role. In Figure 2-31, Web Role 1 and Worker Role 1 have input endpoints available for communication. Any

application can now communicate with endpoints of these roles over the Internet. Because the input endpoints are exposed externally, they are automatically load-balanced by Windows Azure between instances. In some documentation, input endpoints are also referred to as external endpoints. You can get a reference to the input endpoint of an instance as follows:

```
IPEndPoint inpute = RoleEnvironment.Roles["HelloWorkerRole"].Instances[0]
.InstanceEndpoints["MyInputEndpoint"].IPEndPoint;
```

where `HelloWorkerRole` is the name of the Worker role and `MyInputEndpoint` is the name of the endpoint. Once you have the `IPEndPoint` object, you can get the IP address and port number of the endpoint to initiate communications.

■ **Note** You can run multiple web sites on one Web Role instance. For more information, below are two blogs I recommend:

By Wade Wegner, www.wadewegner.com/2011/02/running-multiple-websites-in-a-windows-azure-web-role/

By Andy Cross, blog.bareweb.eu/2011/01/azure-running-multiple-web-sites-in-a-single-webrole/

Local Storage

Windows Azure runtime provides a static function `LocalResource GetLocalResource (string localResourceName)` in the `Microsoft.WindowsAzure.ServiceRuntime.RoleEnvironment` class to get reference to the `LocalResource` class, which represents the local storage space reserved for the service. The `localResourceName` function parameter is the name of storage space defined as the name attribute of `<LocalStorage>` element. In Listing 3-1, I am allocating a space of 10MB for the storage space named `L1` on local machine of the service role instance. I can now get a reference to the local storage space by calling the function `LocalResource resource = RoleEnvironment.GetLocalResource("L1");` and calls `System.IO` file operations on the local path.

■ **Caution** Local storage space allocated on the local machine is local for that instance. If the `cleanOnRoleRecycle` attribute is set to `false`, the data from local directory will be lost on role recycle. So, while developing applications, you should consider local storage purely for unreliable caching purposes with data loss checks built into the application.

Startup Tasks

The `<Startup>` element defines startup tasks that can be executed while the role is starting. Typical scenarios for using startup tasks are as follows:

- Pre-installation of custom software before the role starts (e.g., anti-virus software, Java virtual machine, and so on)
- Installation of COM DLLs
- Making changes to the IIS or any Windows Service before the cloud service is deployed

■ **Note** It is important to understand that role configurations do not survive upgrades and reboots. Therefore, startup tasks will be executed during every startup cycle for maintaining the consistency of instance configuration.

Listing 2-2 illustrates the format for using the startup tasks.

Listing 2-2. ServiceDefinition.csdef

```
<WebRole name=" HelloWebRole ">
  <Startup>
    <Task commandline="[relative path of the executable file]"
      executionContext="limited|elevated"
      taskType="simple|foreground|background"/>
  </Startup>
</WebRole>
```

The executionContext attribute defines the permissions under which the command will be run. The value “limited” indicates the task will be run under the same permission sets as the role, i.e., in user context. The value “elevated” indicates the task will be run under administrator’s privileges. Elevated tasks are typically used for modifying system-level resources like registry and Windows Services. The taskType attribute specifies the behavior of the role while the task is executing. Simple task type is the default value and blocks the role till the task finishes its execution. This task type is commonly used in installing pre-requisite applications like anti-virus software or application frameworks. The background task type does not block the role instantiation while the task is in progress. The task is executed as a background task. The foreground task does not block the role instantiation but blocks the recycle of the role until the task finishes.

■ **Note** In Windows Azure SDK 1.5 (September 2011), new features were included for specifically for launching executables and including local folder contents in application packages.

Sample schema for adding these to your ServiceDefinition file is listed below.

```
<Runtime executionContext="[limited/elevated]"> <Environment> <Variable name="<variable-name>"
value="<variable-value>"> <RoleInstanceValue
```

```

xpath="<xpath-to-role-environment-settings>"/> </Variable> </Environment> <EntryPoint>
<NetFxEntryPoint assemblyName="<name-of-assembly-containing-
entrypoint>"targetFrameworkVersion="<.net-framework-version>"/> <ProgramEntryPoint
commandLine="<application>" setReadyOnProcessStart="[true/false] ""/> </ EntryPoint></Runtime>

```

You can find more information on these features here [msdn.microsoft.com/en-us/library/gg441573\(MSDN.10\).aspx](http://msdn.microsoft.com/en-us/library/gg441573(MSDN.10).aspx)

Nathan Totten also has a good blog article on this topic. ntotten.com/2011/09/running-processes-in-windows-azure/

■ **Tip** You can also run an entire role with elevated privileges by adding the Runtime element with elevated execution context in your role definition. This is not recommended, but there might be some background applications that need elevated privileges. Make sure you don't have any input endpoints defined on such role instances.

```

<WebRole name="WebRole1">

    <Runtime executionContext="elevated" />

</WebRole>

```

Full IIS Support

From Windows Azure 1.3 SDK, you have access to Full IIS running on Windows Azure web roles. You can run multiple web sites pointing to the same physical directory but separated by host headers. For example, in Listing 2-3, there are two sites names, A1 and A2, that are bound to the same HttpIn endpoint, but separated by different host headers. You can also create a virtual applications and virtual directories under the site for further separating the applications within the site.

Listing 2-3. Full IIS Virtual Directories

```

<WebRole name="MyWebApp">
  <Sites>
    <Site name="A1" physicalDirectory="..\MyWebApp">
      <Bindings>
        <Binding name="HttpIn"
          endpointName="HttpIn"

```

```

        hostHeader="www.a1.com" />
    </Bindings>
</Site>
<Site name="Contoso" physicalDirectory="..\MyWebApp">
<VirtualApplication name="myapp"
physicalDirectory="..\..\..\..\apps\myapp">
<VirtualDirectory name="Styles"
physicalDirectory="..\MyWebApp\Styles" />
</VirtualApplication>
    <Bindings>
        <Binding name="HttpIn"
            endpointName="HttpIn"
            hostHeader="www.a2.com" />
    </Bindings>
</Site>
</Sites>
...
</WebRole>

```

Full Trust Execution

By default, Windows Azure applications run under full trust in the cloud environment. When running under partial trust, the code has access only to limited resources and libraries. When running under full trust, cloud services can access certain system resources and can call managed assemblies as well as native code. To enable Full Trust in your application, set the `enableNativeCodeExecution` attribute of the `<WebRole>` or `<WorkerRole>` element in the `ServiceDefinition.csdef` file to `true`:

```
<WebRole name="<role name>" enableNativeCodeExecution="true|false">
```

Table 2-3 lists the permissions for a cloud application role running in partial and full trust execution modes.

Table 2-3. Partial and Full Trust Permissions

Resource	Partial Trust	Full Trust
Call managed code assemblies	Assemblies with <code>AllowPartiallyTrustedCallers</code> attribute	All assemblies
System registry	No access	Read access to HKEY_CLASSES_ROOT HKEY_LOCAL_MACHINE HKEY_USERS HKEY_CURRENT_CONFIG
32-bit P/Invoke	Not supported	Not supported
64-bit P/Invoke	Not supported	Supported
32-bit native sub-process	Not supported	Supported
64-bit native sub-process	Not supported	Supported
Local storage	Full access	Full access
System root and its subdirectories	No access	No access
Windows (e.g., C:\Windows) and its subdirectories	No access	Read access
Machine configuration files	No access	No access
Service configuration file (<code>ServiceConfiguration.cscfg</code>)	Read access	Read access

■ **Note** You can find more information on the Windows Azure partial trust policy in the Windows Azure SDK documentation at <http://msdn.microsoft.com/en-us/library/dd573355.aspx>.

Table 2-3 clearly shows that in partial trust you cannot call native code and the access to the machine resources are limited. Even in full trust execution, the access has been limited to prevent any system-related damage. Partial trust application roles can call only managed code assemblies that have `AllowPartiallyTrustedCallers` attribute, whereas a full trust application role can all any managed code

assembly. A partial trust application role cannot make any P/Invoke native calls. A full trust application role can make P/Invoke calls to a 64-bit library. P/Invoke calls to a 32-bit library are not directly supported in Windows Azure. Instead, you could spawn a 32-bit sub-process from your application role and make P/Invoke calls to 32-bit library from within that sub-process. The system root directory (usually C:\Windows\system32) is not accessible in Windows Azure. A full trust application role has only read access to the Windows directory (usually C:\Windows). Both, full and partial trust roles have full access to the local storage. Local storage is the recommended temporary file and data storage for Windows Azure applications.

■ **Caution** The resource access works differently in the Windows Azure cloud and the development fabric. In the Windows Azure cloud, the application role runs under the privileges of a standard Windows Azure account, whereas the application role in the development fabric runs under the logged-in user account. So, the application role running in the local development fabric may behave differently to the same application role running in the Windows Azure cloud environment.

Certificate Management

In Windows Azure, you can use certificates not only for encrypting the HTTPS endpoints of your web and Worker roles but also for custom message level encryption. You can upload X.509 certificated to your Windows Azure service either from the Windows Azure portal or using the service management API. You can upload any number of certificates for the service and these certificates will be installed in the Windows certificate stores of the role instances.

Once a certificate is uploaded to the service, it can be referenced in the `ServiceDefinition.csdef` and `ServiceConfiguration.cscfg`. The `ServiceDefinition.csdef` defines the name, store location, and store name of the certificate on the instance as shown here:

```
<Certificate name="C1" storeLocation="LocalMachine" storeName="My" />
```

The `ServiceConfiguration.cscfg` file defines the thumbprint and the thumbprint algorithm of the certificate as shown here.

```
<Certificate name="Certificate1" thumbprint="5CA27AF00E1759396Cxxxxxxxxxxxxxx"
thumbprintAlgorithm="sha1" />
```

ServiceConfiguration.cscfg

The `ServiceConfiguration.cscfg` file contains the values for the configuration parameters that apply to one or more instance of the service. It also consists of the `<Instances>` element for scaling your service up and down. For each `<Import moduleName="">`, a series of configuration elements are automatically created in the configuration file for configuring the specific plug-in. You have to configure each plug-in by setting the values of these configuration elements. The service configuration file can be changed dynamically either from the management portal or calling the Service Management API. Listing 2-4 shows the contents of the `ServiceConfiguration.cscfg` file corresponding to the `ServiceDefinition.csdef` file from Listing 2-1.

Listing 2-4. ServiceConfiguration.cscfg

```

<?xml version="1.0"?>

<ServiceConfiguration serviceName="HelloAzureCloud"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration">
  <Role name="HelloWebRole">

    <Instances count="2" />

    <ConfigurationSettings>
      <!--This is the current logging level of the service -->
      <!--Supported Values are Critical, Error,Warning,Information,Verbose-->
      <Setting name="LogLevel" value="Information" />
      <Setting name="ThrowExceptions" value="true" />
      <Setting name="EnableOnScreenLogging" value="true" />
      <Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
value="UseDevelopmentStorage=true" />
      <Setting name="PerformanceCounters" value="\Processor(_Total)% Processor
Time,\Memory\Available MBytes" />
      <Setting name="Microsoft.WindowsAzure.Plugins.RemoteAccess.Enabled" value="true" />
      <Setting name="Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountUsername"
value="tredkar" />
      <Setting name="Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountEncryptedPassword"
value="zBT3zHKb" />
      <Setting name="Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountExpiration"
value="2011-01-13T23:59:59.0000000-08:00" />
    </ConfigurationSettings>
    <Certificates>
      <Certificate name="Microsoft.WindowsAzure.Plugins.RemoteAccess.PasswordEncryption"
thumbprint="AD2D6E79DF99F3C5A55CD98FDBD7DB92F91BE4A7" thumbprintAlgorithm="sha1" />
    </Certificates>
  </Role>
  <Role name="HelloWorkerRole">

    <Instances count="1" />

    <ConfigurationSettings>
      <Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
value="UseDevelopmentStorage=true" />
      <Setting name="PerformanceCounters" value="\Processor(_Total)% Processor
Time,\Memory\Available MBytes" />
      <Setting name="Microsoft.WindowsAzure.Plugins.RemoteAccess.Enabled" value="true" />
      <Setting name="Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountUsername"
value="tredkar" />
      <Setting name="Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountEncryptedPassword"
value="MIIBH" />
      <Setting name="Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountExpiration"
value="2011-01-13T23:59:59.0000000-08:00" />
      <Setting name="Microsoft.WindowsAzure.Plugins.RemoteForwarder.Enabled" value="true" />
      <Setting name="Microsoft.WindowsAzure.Plugins.Connect.ActivationToken" value="0042eeb8-
e1a4-40e3-9141-3dbc7a24f135" />
      <Setting name="Microsoft.WindowsAzure.Plugins.Connect.Refresh" value="" />
    </ConfigurationSettings>
  </Role>
</ServiceConfiguration>

```

```

<Setting name="Microsoft.WindowsAzure.Plugins.Connect.Diagnostics" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.WaitForConnectivity" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.EnableDomainJoin" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainFQDN" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainControllerFQDN" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainAccountName" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainPassword" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainOU" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DNSServers" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.Administrators" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainSiteName" value="" />
</ConfigurationSettings>
<Certificates>
  <Certificate name="Microsoft.WindowsAzure.Plugins.RemoteAccess.PasswordEncryption"
thumbprint="AD2D6E79DF99F3C5A55CD98FDBD7DB92F91BE4A7" thumbprintAlgorithm="sha1" />
</Certificates>
</Role>
</ServiceConfiguration>

```

In Listing 3-2, there are three roles defined, two Web roles and a Worker role. Each role one has only one instance.

■ **Note** For more details on the `ServiceConfiguration.cscfg` and `ServiceDefinition.csdef` schemas, please visit <http://msdn.microsoft.com/en-us/library/dd179398.aspx>.

Web.config Versus ServiceConfiguration.cscfg

The `web.config` file is the configuration file for an ASP.NET web application. It defines the behavior of the ASP.NET application and also custom configuration setting values. Configuration setting values in `web.config`, or `app.config`, for that matter, cannot be dynamically changed at runtime and made available to the application without redeploying the application. Similarly, the ASP.NET behavior defined in a `web.config` file cannot be included in a `ServiceConfiguration.cscfg` file.

`ServiceConfiguration.cscfg` is meant purely for storing configuration-setting values used by the service at runtime; it is not meant to define the behavior of the ASP.NET runtime or the .NET Runtime.

`ServiceConfiguration.cscfg` is the configuration file for one or more instances of a Windows Azure service deployed in the cloud. You can change the configuration values in `ServiceConfiguration.cscfg` at runtime, and they will be available to the application without redeploying the application to the cloud.

Development Fabric

The development fabric simulates the Windows Azure cloud runtime environment on your local machine. The development fabric is specifically designed for development and testing in your local

environment. You cannot attach a development fabric with the Windows Azure cloud service. The development fabric user interface can be started in any of the following manners:

- By debugging or running a cloud service from within Visual Studio
- By running `CSRun.exe` from the command line with valid parameters
- By running `DFUI.exe` from the Windows Azure SDK bin directory
- From the Windows Azure SDK programs Start menu

Once the development fabric starts, you can access it from the development fabric system tray icon. Figure 2-32 illustrates the development fabric user interface hosting a cloud service.

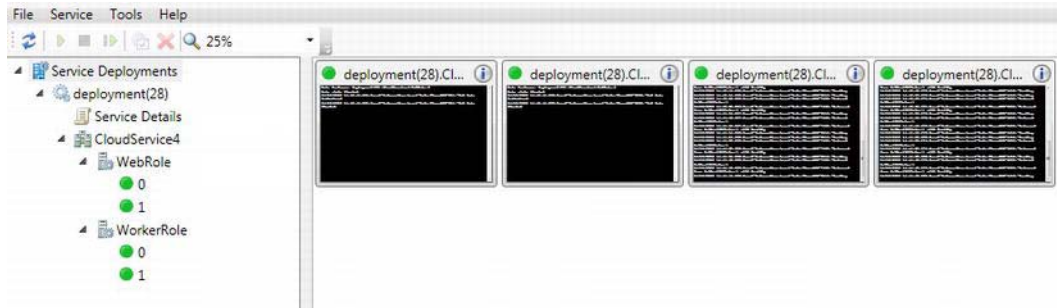


Figure 2-32. Development fabric UI

The development fabric UI shows the service deployments in the local environment and allows you to alter the state of a running service. You can run, suspend, restart, or remove a service deployment from within the development fabric UI.

In the development fabric, you can attach a debugger to the running instance at runtime by right-clicking one of the instance and selecting *Attach Debugger*, as shown in Figure 2-33.

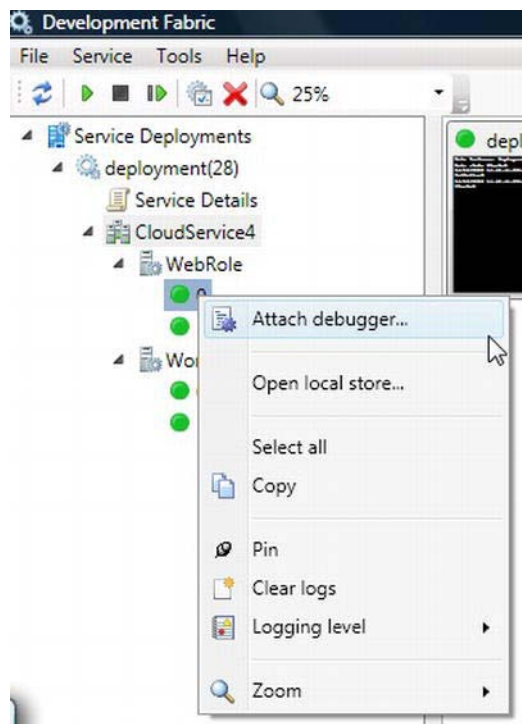


Figure 2-33. Development fabric's Attach Debugger button

The development fabric UI will give you the option of selecting the available debuggers on the local machine. It also allows you to set the logging levels at the service, role, and instance levels.

Development Storage

Development storage simulates the Windows Azure blobs, queues, and table storage services on your local computer over SQL Server Express 2005/2008. Development storage provides a user interface to start, stop, reset, and view the local storage services, as shown in Figure 2-34.

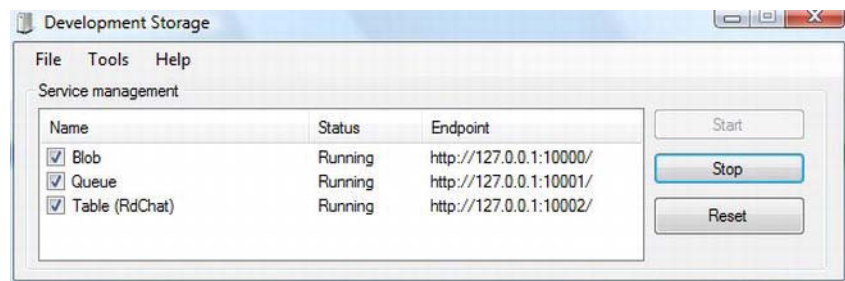


Figure 2-34. Development storage UI

Figure 2-34 shows the name of the service, its status, and the endpoint it is listening on. From Tools ► Table Service Properties, you can change the database to be used by the table storage service.

■ **Note** In Windows Azure SDK 1.3 the Development Storage was renamed to Storage Emulator.

You can change the development storage to point to another database using the DSInit.exe tool that you saw in Table 2-2, with a /sqlInstance parameter.

■ **Note** Use SQL instance name without the server qualifier or use . (a period) for the default instance. To see all the parameters for DSInit.exe, go to the bin directory of the Windows Azure SDK installation, and run DSInit.exe /? from the command prompt.

Table 2-4 lists some key limitations of development storage compared to Windows Azure cloud storage.

Table 2-4. Development Storage Limitations

Attribute	Limitation
Authentication	Development storage only supports a single fixed developer account with a well-known authentication key.
Encryption	Development storage does not support HTTPS.
Scalability	Development storage is not designed to support a large number of concurrent clients. You should use development storage only for functional testing, not for performance or stress testing.
Flexibility	In the CTP version of Windows Azure, the development table storage required a fixed schema to be created before using the table service. The cloud table service did not have this constraint. You can use the table service directly without configuring the schema. The development storage does not require fixed schema any more. String properties in the development table cannot exceed 1,000 characters.
Size	The development blob service supports only 2GB of storage, whereas the cloud Block Blob supports 200GB of storage and Page Blob supports 1TB or storage.

In the case of authentication, the account name, and account key are as follows:

Account name: devstoreaccount1

Account key:

Eby8vdM02xN0cqFlqUwJPLlmEt1lCDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==

■ **Caution** Before deploying storage service application to Windows Azure cloud, please make sure to change the development account information to your cloud account. You cannot use the development storage account to access the Windows Azure storage service in the cloud. Usually, I create two configuration files—one for the development environment and one for the Windows Azure cloud environment—and then I swap based on the deployment destination.

Diagnostics

Logging support in the cloud is one of the biggest concerns of the developer community. With highly interactive integrated design environment (IDE) tools like Visual Studio and runtime environments like the .NET Framework, you can pinpoint problems in your code even in deployed environments when applications are running on-premise. However, the Visual Studio domain is limited to the access it has to the application's runtime environment. Visual Studio communicates with the runtime environment of the application to gather debug information of the application. The application needs to have debug symbols loaded in runtime for Visual Studio to debug. The Windows Azure development fabric has access to the local runtime environment, so you can debug your local Windows Azure application like any other .NET application by adding breakpoints.

Unfortunately, Visual Studio cannot access Windows Azure runtime directly. Once the service is deployed to Windows Azure, it is totally managed by Windows Azure, and you do not have access to its runtime. The Windows Azure team realized this and has added logging capabilities to the Windows Azure runtime and also added IntelliTrace support for Windows Azure deployments from Visual Studio Ultimate edition. The diagnostics service runs along with your role instance, collects diagnostics data as per the configuration, and can save the data to your Windows Azure storage service if configured to do so. You can also communicate with the diagnostics service remotely from an on-premise application or configure it to persist the diagnostics data on a periodic basis. The diagnostics service supports logging of the following data types from your cloud service:

- **Windows Azure Trace logs:** These are the application logs that you dump from your application. These can be any messages emitted from your code.
- **Diagnostic Infrastructure logs:** Infrastructure logs dumped by the diagnostics service.
- **Windows event logs:** These are the Windows event logs generated on the machine on which the role instance is running.
- **Windows performance counters:** These refer to the subscriptions to the performance counters on the machine on which the role instance is running.
- **IIS logs and failed request traces:** These are the IIS logs and the IIS failed request traces generated on the Web role instance.

- **Application crash dumps:** These are the crash dumps generated when an application crashes.

The diagnostics engine aggregates all the logs together and transfers them to the appropriate storage. Windows Azure role instances are stateless and therefore you may lose locally stored logs on recycle. The diagnostics engine named “MonAgentHost.exe” runs on all the role instances by default. Figure 2-35 illustrates the logical architecture of the diagnostics process in one Windows Azure role instance.

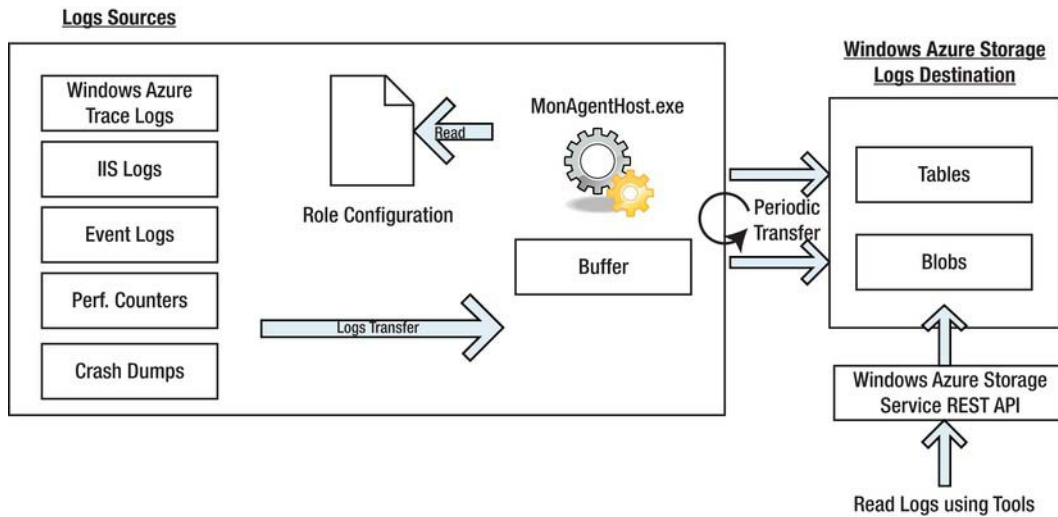


Figure 2-35. Diagnostics in one role instance

Table 2-5 lists the log data sources available in Windows Azure role instances and their respective destinations in the Windows Azure storage.

Table 2-5. Development Storage Limitations

Data Source (on Azure Role instance)	Is it enabled by default in Azure?	Destination Type (in Windows Azure Storage)	Destination Name
Windows Azure Trace Logs	Yes	Table Storage	WADLogsTable
Diagnostic Infrastructure Logs	Yes	Table Storage	WADDiagnosticInfrastructureLogsTable
IIS Logs	Yes	Blob Storage	wad-iis-logfiles \<deployment ID> \<web role name>\

			<role instance>\W3SVC1
Performance Counters	No	Table Storage	WADPerformanceCountersTable
Windows Event Logs	No	Table Storage	WADWindowsEventLogsTable
IIS Failed Request Logs	No	Blob Storage	wad-iis-failedreqlogfiles\<deployment ID>\<web role name>\<role instance>\W3SVC1
Crash Dumps	No	Blob Storage	wad-crash-dumps

You can use the diagnostics management API from outside of the Windows Azure cloud environment (e.g., on-premise) to interact with the diagnostics service on your role instance. Next, using the same API, you can perform scheduled or on-demand transfers of the diagnostics information from role instance machines to your Windows Azure storage account. The diagnostics API is present in the `Microsoft.WindowsAzure.Diagnostics` assembly.

■ **Note** You can find more information about the Windows Azure Runtime API at the Windows Azure MSDN reference site: <http://msdn.microsoft.com/en-us/library/dd179380.aspx>.

Logging

Windows Azure Runtime API consists of a managed code library and an unmanaged code library. In this book, I will cover only the managed code library. The managed code library namespace for diagnostics is `Microsoft.WindowsAzure.Diagnostics`. Associating diagnostics with your cloud service is a three-step process:

1. Configure the trace listener.
2. Define the storage location for the diagnostics service.
3. Start the diagnostics service.

Configuring the Trace Listener

When you create a new role using the role templates template in Visual Studio, the `app.config` and `web.config` files get created automatically in the role project and it consists of a trace listener provider, as shown in Listing 2-5.

Listing 2-5. Diagnostics Trace Listener Configuration

```

<system.diagnostics>
<trace>
  <listeners>
    <add type=
"Microsoft.WindowsAzure.Diagnostics.DiagnosticMonitorTraceListener,
Microsoft.WindowsAzure.Diagnostics, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
name="AzureDiagnostics">
<filter type="" />
    </add>
  </listeners>
</trace>
</system.diagnostics>

```

The `DiagnosticMonitorTraceListener` enables you to use the .NET Tracing API for logging within the code. You can use the `Write()` and `WriteLine()` methods of the `System.Diagnostics.Trace` class for logging from your code as shown here:

```

Trace.WriteLine("INFORMATION LOG", "Information");
Trace.WriteLine("CRITICAL LOG", "Critical");

```

Defining the Storage Location for the Diagnostics Service

In the `ServiceDefinition.csdef` and `ServiceConfiguration.cscfg` files, you have to define the diagnostics connection string pointing to the storage location of your choice (development storage or cloud storage), and custom settings like the list of `PerformanceCounters`. Visual Studio then automatically generates this configuration for you as shown in Listing 2-6.

*Listing 2-6. Diagnostics Connection String Configuration***For development storage:**

```

<ConfigurationSettings>
  <Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
value="UseDevelopmentStorage=true" />
  <Setting name="PerformanceCounters"
value="\Processor(_Total)\% Processor Time,\Memory\Available MBytes" />
</ConfigurationSettings>

```

For cloud storage:

```

<ConfigurationSettings>
  <Setting name="DiagnosticsConnectionString" value=
"DefaultEndpointsProtocol=https;AccountName=proazurestorage;AccountKey=[YOURKEY]"/>
  <Setting name="PerformanceCounters"
value="\Processor(_Total)\% Processor Time,\Memory\Available MBytes" />

```

```
</ConfigurationSettings>
```

Starting the Diagnostics Service

Next, you have to start the diagnostics service in your role by passing in the connection string name you defined in step 2. Once started, the diagnostics monitoring service can start collecting the logged data. You can also choose to further configure the diagnostics service through the `DiagnosticMonitorConfiguration` class, as shown in Listing 2-7.

Listing 2-7. *Programmatically Changing the Diagnostics Configuration*

```
private void SetupDiagnostics()
{
    Trace.WriteLine("Setting up diagnostics", "Information");

    DiagnosticMonitorConfiguration diagConfig =
    DiagnosticMonitor.GetDefaultInitialConfiguration();

    // Add performance counter monitoring for configured counters
    // Run typeperf.exe /q to query the counter list
    string perfCounterString =
    RoleEnvironment.GetConfigurationSettingValue("PerformanceCounters");

    if (!string.IsNullOrEmpty(perfCounterString))
    {
        IList<string> perfCounters = perfCounterString.Split(',').ToList();

        // Setup each counter specified in comma delimited string
        foreach (string perfCounter in perfCounters)
        {
            diagConfig.PerformanceCounters.DataSources.Add(
                new PerformanceCounterConfiguration
                {
                    CounterSpecifier = perfCounter,
                    SampleRate = TimeSpan.FromSeconds(5)
                }
            );
        }

        // Update counter information in Azure every 30 seconds
        diagConfig.PerformanceCounters.ScheduledTransferPeriod =
        TimeSpan.FromMinutes(0.5);
    }

    diagConfig.DiagnosticInfrastructureLogs.ScheduledTransferPeriod =
    TimeSpan.FromMinutes(0.5);

    // Specify a logging level to filter records to transfer
    diagConfig.DiagnosticInfrastructureLogs.ScheduledTransferLogLevelFilter =
    LogLevel.Verbose;
}
```

```

// Set scheduled transfer interval for user's Windows Azure Logs to 5 minutes
diagConfig.Logs.ScheduledTransferPeriod = TimeSpan.FromMinutes(5);

diagConfig.Directories.ScheduledTransferPeriod = TimeSpan.FromMinutes(5);

Microsoft.WindowsAzure.Diagnostics.CrashDumps.EnableCollection(true);

//Event Logs
// Add event collection from the Windows Event Log
diagConfig.WindowsEventLog.DataSources.Add("System!*");
diagConfig.WindowsEventLog.DataSources.Add("Application!*");
diagConfig.WindowsEventLog.DataSources.Add("Security!*");
diagConfig.WindowsEventLog.ScheduledTransferPeriod = TimeSpan.FromMinutes(5);

// Start the diagnostic monitor with this custom configuration
DiagnosticMonitor.Start("Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString",
diagConfig);
}

```

In Listing 2-7, the `SetupDiagnostics()` function performs the following:

1. Initializes the Diagnostic Monitor Configuration.
2. Reads the Performance Counters listed in the `ServiceConfiguration.cscfg`.
3. Adds each Performance Counter as a data source to the Diagnostic Monitor Configuration.
4. Schedules the transfer of infrastructure and directories (e.g., IIS) logs to the Windows Azure Storage.
5. Initializes Diagnostic Infrastructure logs transfer.
6. Initializes crash dump collection and transfer.
7. Adds System, Application, and Security Event Log data sources to the Diagnostic Monitor Configuration.
8. Schedules the transfer of Event Log data.
9. Starts the Diagnostic Monitor service.

Once the logs data gets collected in the Windows Azure storage destinations, you can read it by either writing a custom application that calls the Windows Azure storage REST API or by running a third-party tool like the Cerebrata's Azure Diagnostics Manager (www.cerebrata.com/Products/AzureDiagnosticsManager/Default.aspx).

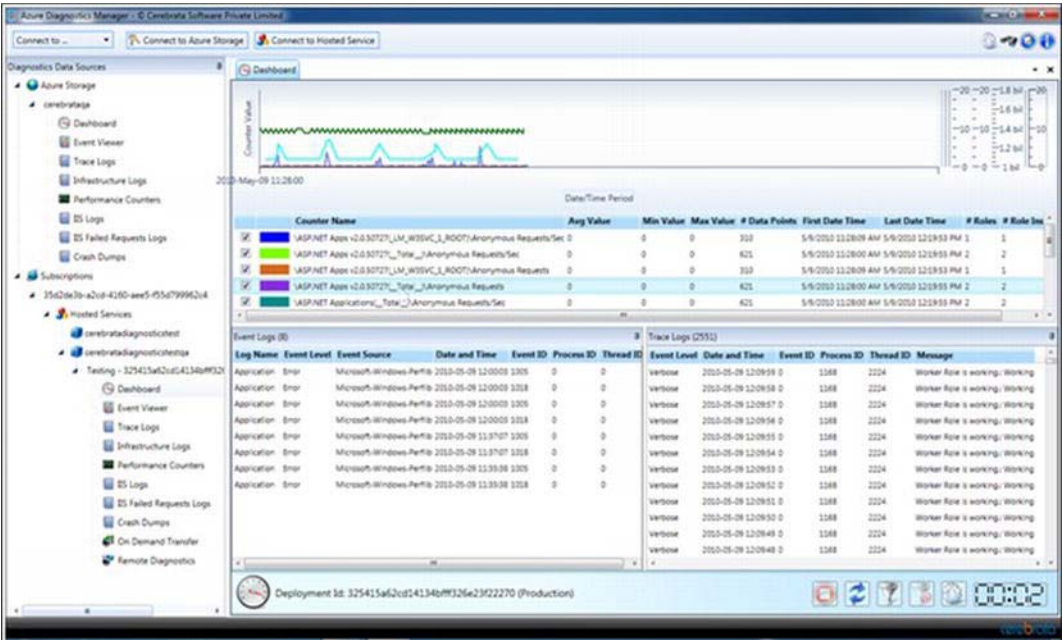


Figure 2-36. Cerebrata Azure Diagnostics Manager

■ **Note** Microsoft has also released System Center Operations Manager (SCOM) pack for Windows Azure. You can download it here microsoft.com/download/en/details.aspx

Walter Myers III has a nice series of blog articles on Windows Azure application monitoring with SCOM: blogs.msdn.com/b/walterm/archive/2011/02/14/adding-azure-application-monitoring-to-scom-2007-r2.aspx

■ **Tip** When designing cloud applications, it is important to design diagnostics and logs reporting right from the beginning. This will save you a lot of debugging time and help you create a high quality application.

Developing Windows Azure Services with Inter-Role Communication

In this example, you will learn to develop Windows Azure services in the local development fabric and in the cloud environment. I will also show you how to communicate between roles using the internal endpoints. You will also learn to use your own Configuration Settings in ServiceDefinition.csdef and ServiceConfiguration.cscfg.

Objectives

The objectives of this example are as follows:

- Understanding inter-role communication in Windows Azure cloud services.
- Accessing local machine resources.
- Understanding the configuration settings for configuring cloud services.

Adding Diagnostics and Inter-role Communication

In this section, I will guide you through the code for adding diagnostics, configuration and inter-role communication to the Windows Azure services. Follow these steps:

1. Open *Ch2Solution.sln* from Chapter 2's source code directory.
2. Expand the *HelloService* folder, as shown in Figure 2-37.

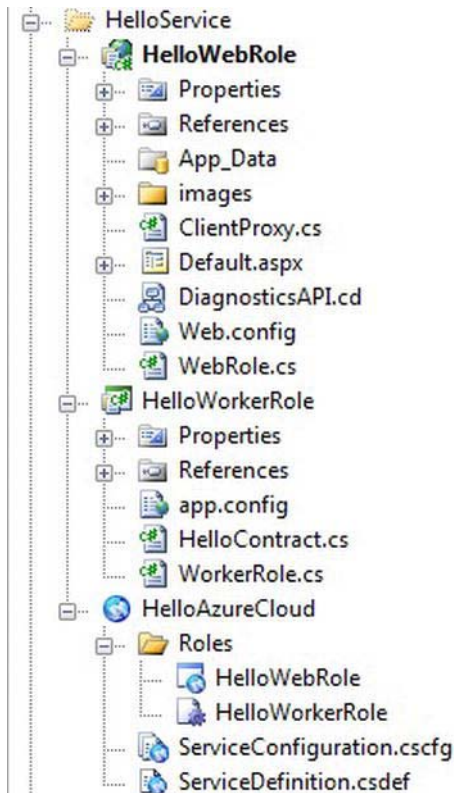


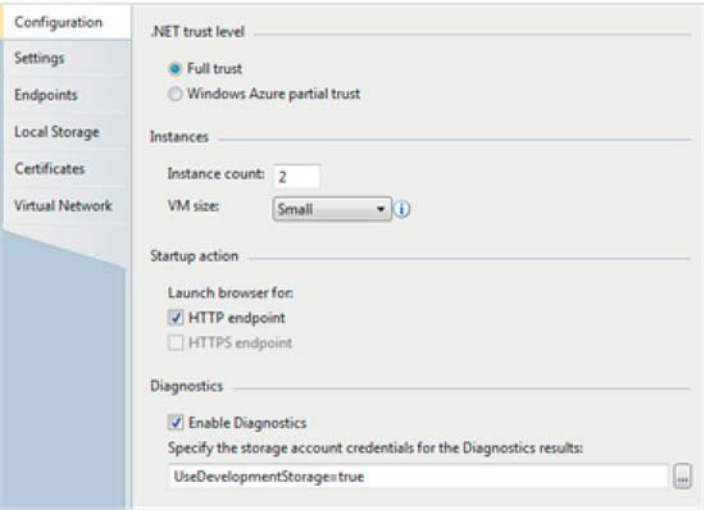
Figure 2-37. *HelloService* folder

The folder contains one Web role, one Worker role, and one cloud service project: HelloWebRole, HelloWorkerRole, and HelloAzureCloud cloud service respectively.

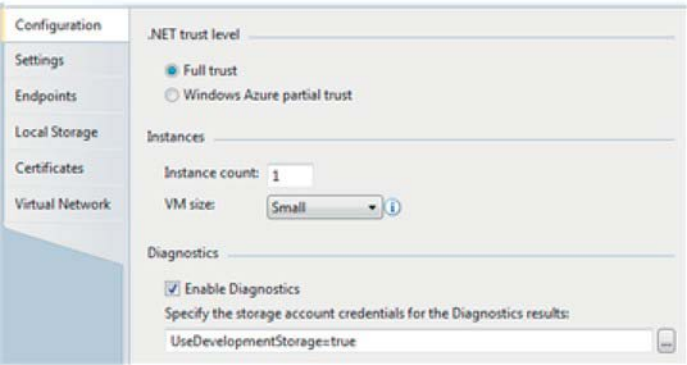
Service Model

The ServiceDefinition.csdef and ServiceConfiguration.cscfg file define the service model and configuration values for the service. Figures 2-38, 2-39, 2-40, 2-41, and 2-42 illustrate the Service of the HelloAzureCloud service.

Configuration



HelloWebRole



HelloWorkerRole

Figure 2-38. HelloAzureCloud – Configuration

Settings

Settings			
Add configuration settings that can be accessed programmatically and dynamically updated.			
Name	Type	Value	
PerformanceCounters	String	[ASP.NET] Request Execution Time	
LogLevel	String	Information	
ThrowExceptions	String	true	
EnableOnScreenLogging	String	true	
Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString	Connection String	UseDevelopmentStorage=true	
Microsoft.WindowsAzure.Plugins.RemoteAccess.Enabled	String	true	
Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountUsername	String	tredkar	
Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountEncryptedPassword	String	MBBHAYJKoZihvcNAQcDoBBOTCC	
Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountExpiration	String	2011-01-13T23:59:59.0000000-08:00	

HelloWebRole

Settings			
Add configuration settings that can be accessed programmatically and dynamically updated.			
Name	Type	Value	
PerformanceCounters	String	[ASP.NET] Request Execution Time	
Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString	Connection String	UseDevelopmentStorage=true	
Microsoft.WindowsAzure.Plugins.RemoteAccess.Enabled	String	true	
Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountUsername	String	tredkar	
Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountEncryptedPassword	String	MBBHAYJKoZihvcNAQcDoBBOTCC	
Microsoft.WindowsAzure.Plugins.RemoteAccess.AccountExpiration	String	2011-01-13T23:59:59.0000000-08:00	
Microsoft.WindowsAzure.Plugins.RemoteForwarder.Enabled	String	true	

HelloWorkerRole

Figure 2-39. HelloAzureCloud – Settings

Endpoints

compute emulator).						
Name	Type	Protocol	Public Port	Private Port	SSL Certificate Name	
HttpIn	Input	http	8080			

HelloWebRole

compute emulator).						
Name	Type	Protocol	Public Port	Private Port	SSL Certificate Name	
MyInternalEndpoint	Internal	tcp				
MyExternalEndpoint	Input	tcp	9001		Input value(s) like *, 8080, or/and 8000-8080. (not applicable)	

HelloWorkerRole

Figure 2-40. HelloAzureCloud – Endpoints



Figure 2-41. HelloAzureCloud – Local Storage

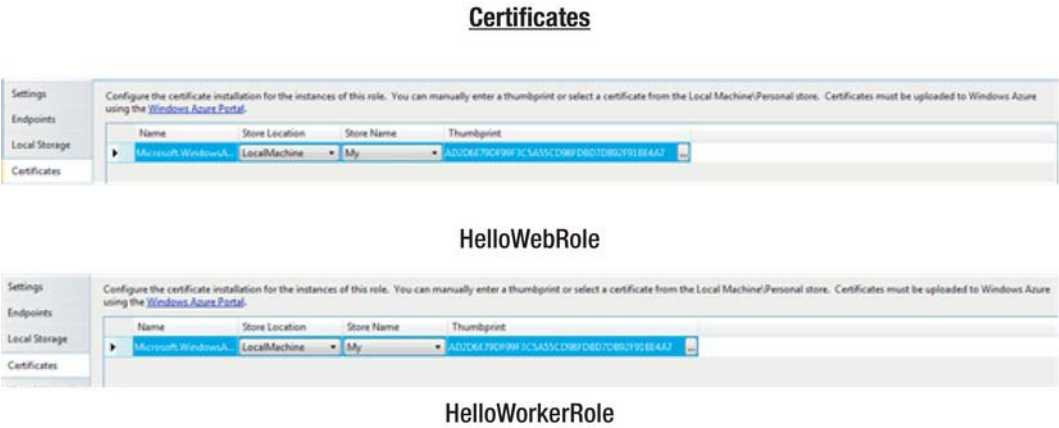


Figure 2-42. HelloAzureCloud – Certificates

This service model defines an external HTTP endpoint (input endpoint) for the HelloWebRole listening on port 80 and internal as well as external endpoints for the HelloWorkerRole. HelloWebRole also defines a local storage named HelloAzureWorldLocalCache with maximum size of 10MB. Both the roles define the Diagnostics and Remote Access plug-in.

The ServiceDefinition.csdef also defines a startup task that calls Startme.cmd, which in turn opens a command prompt.

```
<Startup>
  <Task commandLine="Startme.cmd" executionContext="elevated" taskType="simple">
  </Task>
</Startup>
```

Worker Role

HelloWorkerRole implements two methods, OnStart() and Run(). In the OnStart() method, it also subscribes to the role changing event to catch any configuration changes.

■ **Note** In both the Web and the Worker roles, you need to add references to the following assemblies: Microsoft.WindowsAzure.ServiceRuntime.dll, Microsoft.WindowsAzure.StorageClient.dll and Microsoft.WindowsAzure.Diagnostics.dll. And you need to add the following using statements in code: using Microsoft.WindowsAzure.ServiceRuntime; and using Microsoft.WindowsAzure.Diagnostics;. The storage client is used for transferring logs from Windows Azure instances to the storage service.

Listing 2-8 shows the code for the HelloWorldRole class.

Listing 2-8. HelloWorldRole

```
public override void Run()
{
    Trace.WriteLine("HelloWorkerRole entry point called", "Information");
    var internalEndpoint =
RoleEnvironment.CurrentRoleInstance.InstanceEndpoints["MyInternalEndpoint"];
    var wcfAddress = new
Uri(String.Format("net.tcp://{0}", internalEndpoint.IPEndpoint.ToString()));
    Trace.WriteLine(wcfAddress.ToString());
    var wcfHost = new ServiceHost(typeof(HelloServiceImpl), wcfAddress);
    var binding = new NetTcpBinding(SecurityMode.None);
    wcfHost.AddServiceEndpoint(typeof(IHelloService), binding, "helloservice");
    try
    {
        wcfHost.Open();
        while (true)
        {
            Thread.Sleep(10000);
            Trace.WriteLine("Working", "Information");
        }
    }
    finally
    {
        wcfHost.Close();
    }
}

public override bool OnStart()
{
    // Set the maximum number of concurrent connections
    ServicePointManager.DefaultConnectionLimit = 12;
    SetupDiagnostics();
    Trace.WriteLine("Diagnostics setup...");
    // For information on handling configuration changes
    // see the MSDN topic at http://go.microsoft.com/fwlink/?LinkId=166357.
```

```

RoleEnvironment.Changing += RoleEnvironmentChanging;

return base.OnStart();
RoleEnvironment.Changing += RoleEnvironmentChanging;
return base.OnStart();
}

private void RoleEnvironmentChanging(object sender,
RoleEnvironmentChangingEventArgs e)
{
    if (e.Changes.Any(change => change is RoleEnvironmentConfigurationSettingChange))
        e.Cancel = true;
}

```

The `OnStart()` method starts the diagnostics service (see Listing 2-7 for the `SetupDiagnostics()` method code) with a scheduled log transfer to the storage and also subscribes to the `Changing` event of the Windows Azure runtime to detect any changes to the configuration. You can use the `RoleEnvironmentChanging` event to capture the following changes:

- `RoleEnvironmentConfigurationSettingChange` to detect the changes in the service configuration.
- `RoleEnvironmentTopologyChange` to detect the changes to the role instances in the service.

In addition, you can remove a role instance from the load-balancer after the service has started by subscribing the `RoleEnvironment.StatusCheck` event and calling `SetBusy()` method on the `RoleInstanceStatusCheckEventArgs`. You can also request a recycle of the role instance on-demand by calling the `RoleEnvironment.RequestRecycle()` method. For more information on runtime API, please see the `Microsoft.WindowsAzure.ServiceRuntime` namespace in the Windows Azure SDK class documentation.

Because the diagnostics service is configured to save all the logs to the Windows Azure storage, all the `Trace.WriteLine()` statements will be sent to the storage periodically. The `Run()` method gets a reference to the internal endpoint named `MyInternalEndpoint` from the service definition and retrieves its IP address and creates a WCF service host for the `HelloServiceImpl`. Once the WCF host is opened on the internal IP address and port, any role in the service can make WCF method calls. Listing 2-9 shows the code for `IHelloService` and `HelloServiceImpl`.

Listing 2-9. Hello Contract

```

[ServiceContract (Namespace="http://proazure/helloservice")]
interface IHelloService
{
    [OperationContract]
    string GetMyIp();
    [OperationContract]
    string GetHostName();
    [OperationContract]
    int GetUpdateDomain();
    [OperationContract]

```

```

        int GetFaultDomain();
    }
    [ServiceBehavior(AddressFilterMode=AddressFilterMode.Any)]
    public class HelloServiceImpl : IHelloService
    {

        #region IHelloService Members

        public string GetMyIp()
        {
            IPAddress[] ips = null;

            ips = Dns.GetHostAddresses(Dns.GetHostName());

            if (ips != null)
            {
                foreach (IPAddress i in ips)
                {
                    if(i.AddressFamily ==
System.Net.Sockets.AddressFamily.InterNetwork)
                        return i.ToString(); ;
                }
            }

            return "";
        }

        #endregion

        public string GetHostName()
        {
            return Dns.GetHostName();
        }

        public int GetUpdateDomain()
        {
            return RoleEnvironment.CurrentRoleInstance.UpdateDomain;
        }

        public int GetFaultDomain()
        {
            return RoleEnvironment.CurrentRoleInstance.FaultDomain;
        }

    }
}

```

The IHelloService interface defines four methods to retrieve the IP address, domain name of the machine, Upgrade Domain and Fault domain. HelloServiceImpl class implements these four methods.

Web Role

The user interface for the Web role is in Default.aspx. The user interface is designed to do a few operations when you click the Get Machine Info button. Figure 2-43 illustrates the user interface design of Default.aspx page.

Get Machine Info

Cloud Machine Name	[lblCloudMachineName]
User Host Address	[lblUserHostAddress]
Current Log Level	[lblCurrentLogLevel]
Local Storage Root Path	[lblLocalStoragePath]
Throw Exceptions	[lblThrowExceptions]
Can Access System Directory?	[lblCanAccessSystemDirectory]
Can Access Windows Directory?	[lblCanAccessWindowsDirectory]
Can Access Local Storage?	[lblCanAccessLocalStorage]
Update Domain	[lblUpgradeDomain]
Fault Domain	[lblFaultDomain]
Worker Role Communication	IP Address [lblWRIP] Host Name [lblWRHostName] Endpoint Address [lblWREndpointAddress] Update Domain [lblWRUpgradeDomain] Fault Domain [lblWRFaultDomain]

Upload File to Local Storage

Browse...

Upload

File List in Local Storage

Unbound

Exceptions

[lblException]

Logs

[lblLogging]

Figure 2-43. Default.aspx user interface design

When you click the Get Machine Info button, it retrieves the machine name, host address, and local storage and calls the HelloWorkerRole service through an internal endpoint. You can also upload a file to the local storage using the upload file button. All the functions use traditional .NET APIs for retrieving local file and network information of the machine. If you are developing the service from scratch, you will have to add reference to the HelloWorkRole WCF service. In the HelloWebRole project, the reference has already been added for you in the ClientProxy.cs file. Listing 2-10 shows the code for calling the HelloWorkerRole WCF service.

Listing 2-10. Call Worker Role WCF Service

```
string wrIp = RoleEnvironment.Roles["HelloWorkerRole"].Instances[0].
InstanceEndpoints["MyInternalEndpoint"].IPEndpoint.ToString();
lblWREndpointAddress.Text = wrIp;
```

```

var serviceAddress = new Uri(String.Format("net.tcp://{0}/{1}", wrIp, "helloservice"));
var endpointAddress = new EndpointAddress(serviceAddress);
var binding = new NetTcpBinding(SecurityMode.None);
var client = new ClientProxy(binding, endpointAddress);
lblWRHostName.Text = client.GetHostName();
lblWRIp.Text = client.GetMyIp();
lblUpgradeDomain.Text = client.GetUpdateDomain().ToString();
lblFaultDomain.Text = client.GetFaultDomain().ToString();

```

In Listing 2-10, the Web role gets reference to the internal endpoint of a Worker role instance and instantiates the ClientProxy object to call the IHelloService methods.

■ **Note** An important point to note here is that the endpoints are exposed as IP address of the instance and you still have to build your server in the form of TcpListener, WCF service, or HTTP service on that IP address.

Running the HelloAzureCloud Service

To build and run the solution, press F5 on the HelloAzureCloud project to start it in debug mode. Click the Get Machine Info button. Figure 2-44 illustrates the HelloAzureCloud Web role application running on the local machine.



Figure 2-44. HelloAzureCloud on local machine

Open the development fabric UI by clicking the development fabric icon in the system tray. Figure 2-45 shows the development fabric UI running two instances of the Web role and one instance of the Worker role.

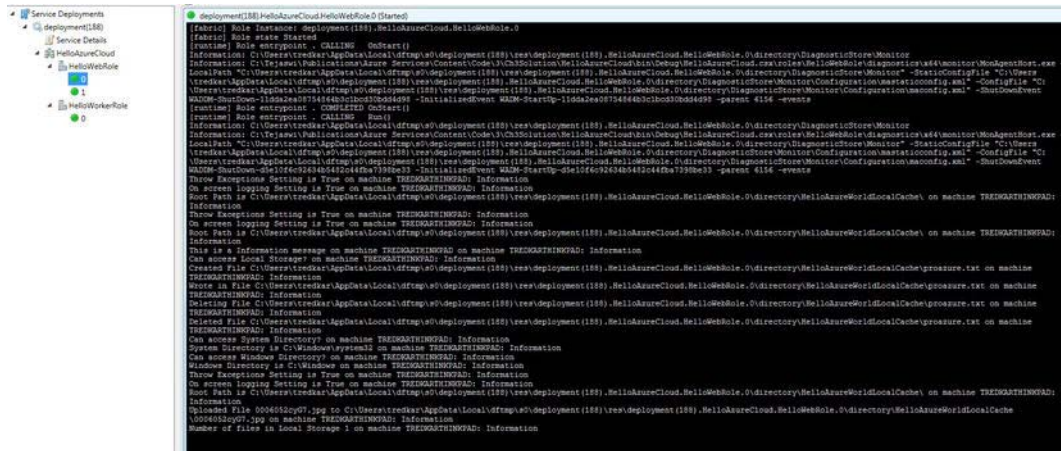


Figure 2-45. HelloAzureCloud development fabric two instances

The information is logged either in the console of instance 0 or instance 1 depending on where the load balancer sends the request. If you click the Get Machine Info button very quickly, you will see that the request gets load balanced across both the instances. Figure 2-46 shows the load-balanced requests across two instances of the Web role application.

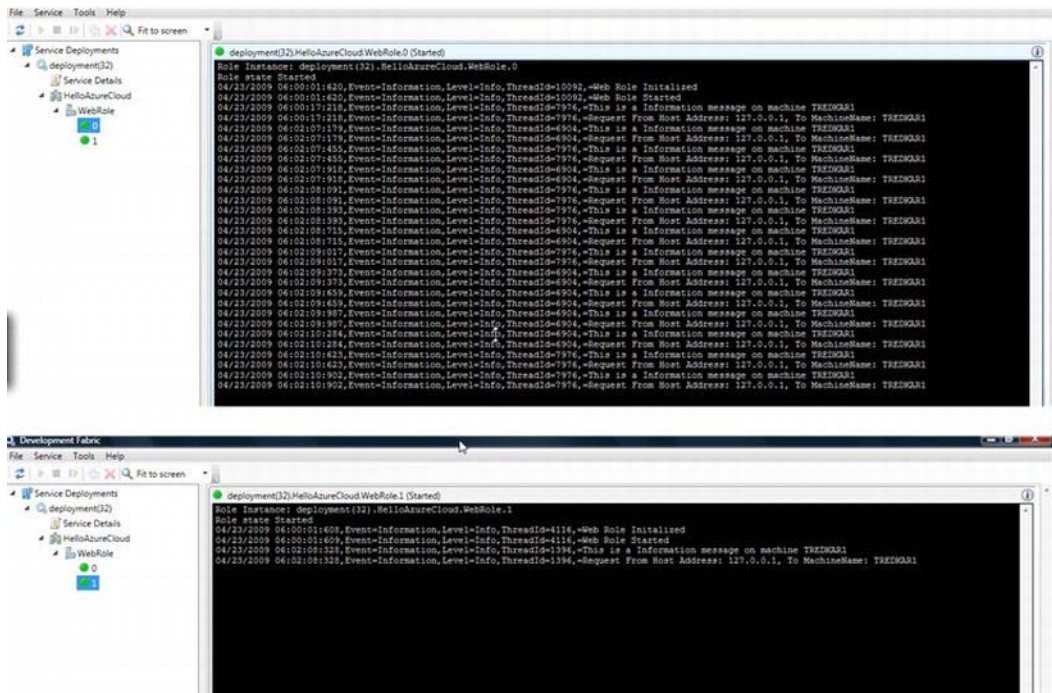


Figure 2-46. Load Balance across two instances of HelloAzureCloud service

In Figure 2-46, observe the logs in the consoles of both the instances of HelloAzureCloud Web roles.

Now that you have tested the cloud service in the development fabric, you can deploy it in the Windows Azure cloud. When you deploy the application to the cloud, the consoles that you see in the development fabric are not available to visually view the logs. So, let's see how you can access and view these logs in the cloud.

Publishing to Windows Azure Cloud

For publishing the HelloAzureCloud service to Windows Azure, you should first provision a service. The following steps outline how:

1. Log in to <https://windows.azure.com>.
2. Go to the Hosted Services section.
3. Click New Hosted Service from the top menu.
4. Enter a name for your service.
5. Choose a URL prefix for your service.
6. Choose a region closest to your location.

7. In the Deployment Options, select “Do not deploy,” because in this exercise you will deploy from the Visual Studio directly.
8. Click OK.
9. Add a certificate under the certificates folder.
10. Repeat these steps to create a new storage service.

Create a new Hosted Service

Choose a subscription
tredkar@northamerica.corp.microsoft.com

Enter a name for your service
tejaswi

Enter a URL prefix for your service
tejaswi.cloudapp.net

Choose a region or affinity group
☒ North Europe ☐ Create or choose an affinity group

Deployment options
☐ Deploy to stage environment
☐ Deploy to production environment
☒ Do not deploy
☒ Start after successful deployment

Deployment name
[Empty text box]

Package location
[Empty text box] Browse Locally... Browse Storage...

Configuration file
[Empty text box] Browse Locally... Browse Storage...

Add Certificate

OK Cancel

Figure 2-47. Create a new Hosted Service

■ **Note** For information on creating your own X.509 certificate, visit <http://msdn.microsoft.com/en-us/library/gg432987.aspx>.

To deploy HelloAzureCloud service to Windows Azure, right-click on the HelloAzureCloud project in Visual Studio, and select Publish to create the service package HelloAzureCloud.cspkg, as shown in the Figure 2-48.

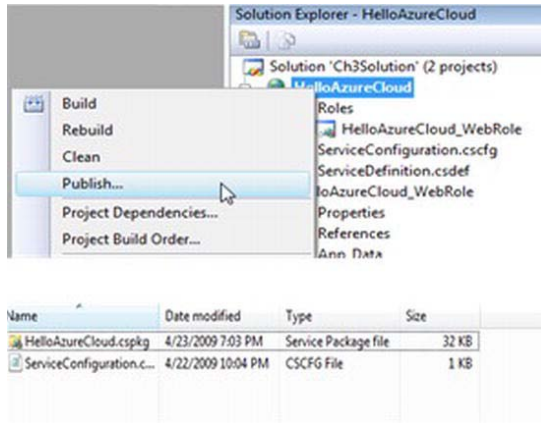


Figure 2-48. Publish to Windows Azure

Open the Deploy Windows Azure Project dialog box, as shown in Figure 2-49.

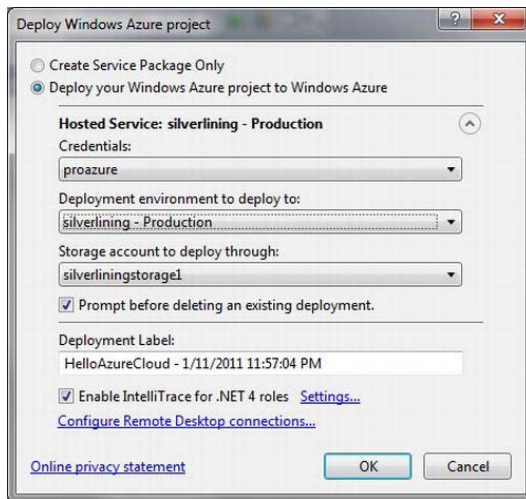


Figure 2-49. Deploy Windows Azure Project Windows

Here, follow these steps:

- 1. Select the certificate your service management certificate you created in the previous section.
- 2. Select the storage location for storing the uploaded package.
- 3. Check Enable IntelliTrace (optional).
- 4. Configure Remote Desktop Connections (optional).
- 5. Click OK to start the Windows Azure Activity Log.
- 6. The windows will show completed status on successful deployment of the project, as shown in Figure 2-50.

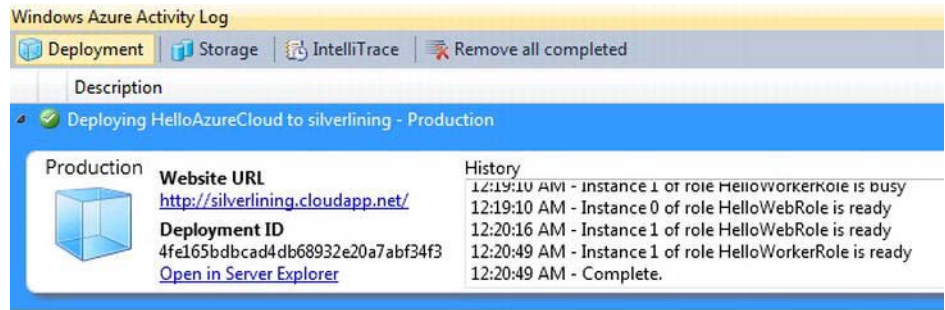


Figure 2-50. Windows Azure Activity Log

For viewing the application, you can point to the URL of the service and the appropriate HTTP port corresponding to the HttpIn input endpoint you created for the web role. In the previous example, the Web role's HTTP input endpoint was configured to listen on TCP port 8080. There the appropriate URL for the HelloAzureCloud service is `http://silverlining.cloudapp.net:8080/Default.aspx`.

Figure 2-51 illustrates the HelloAzureCloud service running in Windows Azure.

Cloud Machine Name	RD00155D323CC6
User Host Address	69.181.83.189
RoleId	HelloWebRole_IN_0
Current Log Level	Information
Local Storage Root Path	C:\Resources\directory\ed8c589079684e2384aa6cf2616071e3.HelloWebRole.HelloAzureWorldLocalCache\
Throw Exceptions	True
Can Access System Directory?	true.D:\windows\system32
Can Access Windows Directory?	true.D:\windows
Can Access Local Storage?	true
Update Domain	0
Fault Domain	0
Worker Role Communication	IPAddress /10.28.199.78/10.28.191.94 Host Name /RD00155D323AF1/RD00155D323596 Endpoint Address /10.28.199.78:20001/10.28.191.94:20001 Update Domain /0/1 Fault Domain /0/1

Cloud Machine Name	RD00155D323528
User Host Address	69.181.83.189
RoleId	HelloWebRole_IN_1
Current Log Level	Information
Local Storage Root Path	C:\Resources\directory\ed8c589079684e2384aa6cf2616071e3.HelloWebRole.HelloAzureWorldLocalCache\
Throw Exceptions	True
Can Access System Directory?	true.D:\windows\system32
Can Access Windows Directory?	true.D:\windows
Can Access Local Storage?	true
Update Domain	1
Fault Domain	1
Worker Role Communication	IPAddress /10.28.199.78/10.28.191.94 Host Name /RD00155D323AF1/RD00155D323596 Endpoint Address /10.28.199.78:20001/10.28.191.94:20001 Update Domain /0/1 Fault Domain /0/1

Figure 2-51. HelloAzureCloud running in Windows Azure

Here are a few things to observe when running the HelloAzureCloud service in Windows Azure:

1. The IPAddress and the machine name changes in Windows Azure.
2. The format of the RoleIds id is different than running in development fabric.
3. Two Upgrade Domains and Two Fault Domains are utilized.
4. If you refresh the page often or start another browser, you will see the changes in the Web role information because the request is load-balanced to the second instance. The second instance will show the second Upgrade Domains and the Fault Domain.

Next, select the deployed service and click the Configure button from the top menu. On the Configure Deployment screen, select the Edit Configuration option and change the number of instances of the Worker role from 2 to 3 as shown in Figure 2-52. While the roles are updating, continue to interact

with the HelloAzureCloud application through Default.aspx. Observe that even if the configuration changes at runtime, the existing role instances continue to run unaffected.

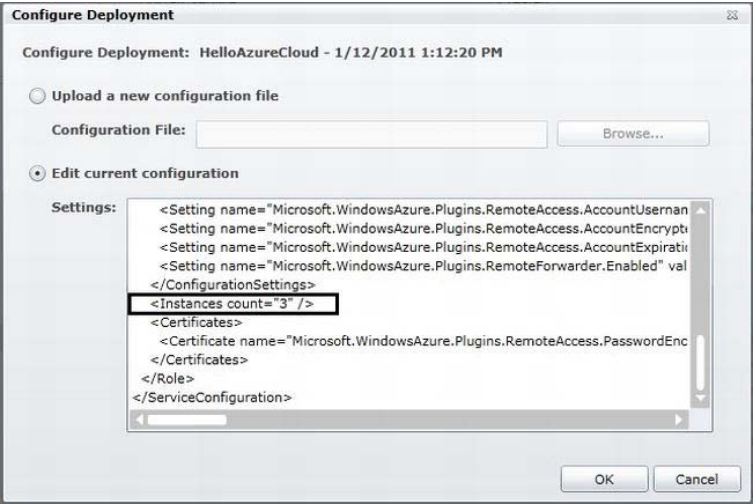


Figure 2-52. Edit the configuration

After all the instances get into the Ready state, go back to the HelloAzureCloud application and click the GetMachineInfo button. You will observe that the Windows Azure Fabric Controller has placed the third worker role instance in a third Upgrade Domain (2) but in one of the two existing Fault Domains (0 or 1) as shown in Figure 2-53. It is important to know these settings when you are planning for high-availability and maintenance. You can also specify the number of Upgrade Domains you want to you in the Service Definition file

<ServiceDefinition name="<service-name>" **upgradeDomainCount="<number-of-upgrade-domains>"**>

Cloud Machine Name	RD00155D323528
User Host Address	69.181.83.189
RoleId	HelloWebRole_IN_1
Current Log Level	Information
Local Storage Root Path	C:\Resources\directory\ed8c589079684e2384aa6cf2616071e3.HelloWebRole.HelloAzureWorldLocalCache\
Throw Exceptions	True
Can Access System Directory?	true.D:\windows\system32
Can Access Windows Directory?	true.D:\windows
Can Access Local Storage?	true
Update Domain	1
Fault Domain	1
Worker Role Communication	IPAddress /10.28.199.78/10.28.191.94/10.28.197.101
	Host Name /RD00155D323AF1/RD00155D323596/RD00155D3239A5
	Endpoint Address /10.28.199.78:20001/10.28.191.94:20001/10.28.197.101:20001
	Update Domain /0/1/2
	Fault Domain /0/1/0

Figure 2-53. Upgrade the Domain and Fault Domain

For upgrading your service in-place, follow these steps:

1. Login to Windows Azure Management Portal.
2. Go to Hosted Services.
3. Select the service you want to upgrade.
4. Click on the Upgrade button from the top menu.
5. Enter the new package and configuration information.
6. In the Upgrade Mode, specify Automatic for upgrading all Upgrade Domains at the same time and Manual for upgrading one Upgrade Domain at a time.
7. Click OK and proceed with a Manual Upgrade when the service state changes to “Ready for Manual Upgrade.”

■ **Note** For more information on in-place upgrades, please visit <http://msdn.microsoft.com/en-us/library/ee517255.aspx>.

Viewing IntelliTrace Logs

When you deployed the HelloAzureCloud service to Windows Azure, you selected the checkbox for enabling IntelliTrace. I have found IntelliTrace logs handy in cases where the deployment did not go well due to missing dependencies or incorrect configuration settings. Such mistakes are difficult to catch in cloud environments.

For viewing the IntelliTrace logs of the deployment, go to Server Explorer, right-click on the cloud service instances and select View IntelliTrace logs as shown in Figure 2-54.

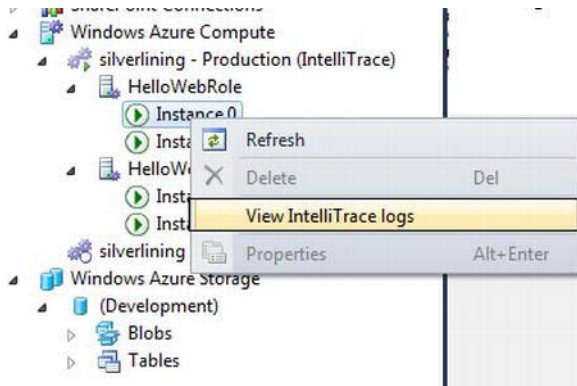


Figure 2-54. Download IntelliTrace Logs

This action will start downloading IntelliTrace logs from the storage service and open the summary in Visual Studio as shown in Figure 2-55.

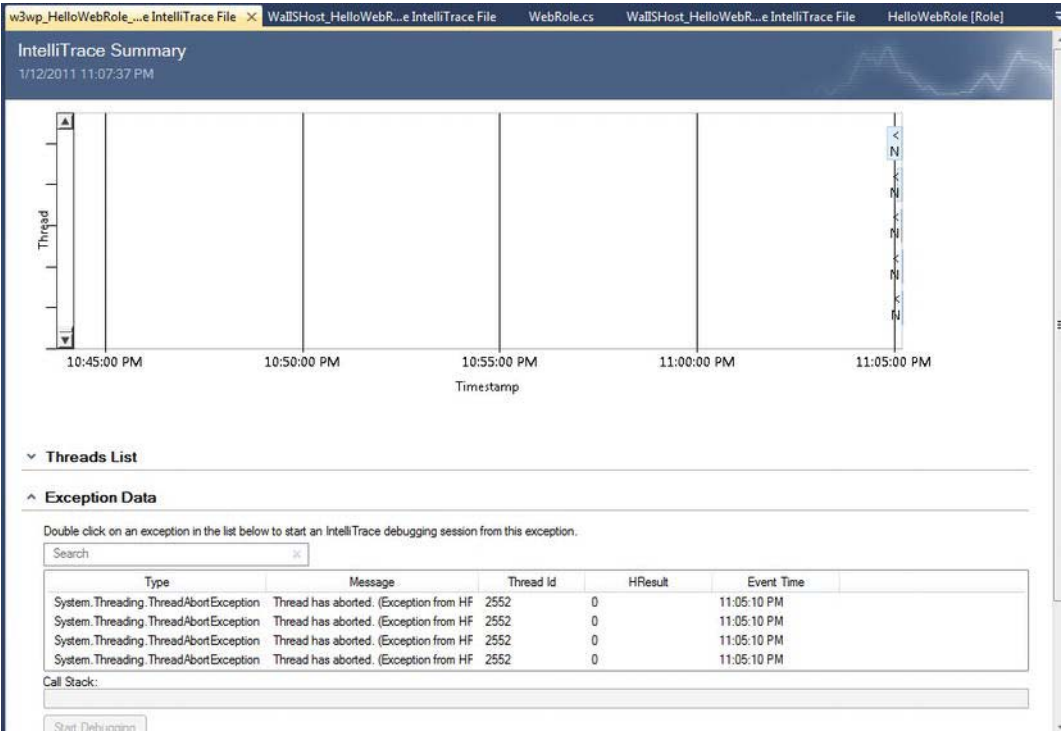


Figure 2-55. IntelliTrace Logs

The IntelliTrace log contains the recording of the deployment. The recording summary includes a time chart, exception data, threads list system information, and modules. With this exception and stack trace data, you can pinpoint certain type of errors very quickly.

Connecting with Remote Desktop Connection

When you published the cloud service to Windows Azure from Visual Studio, you had the option for enabling Remote Access. If you enabled remote access, you can log in to the cloud service instances. Enable remote access as outlined in the following steps:

1. Go to the Hosted Services page on the management portal.
2. Select a specific instance and click Connect from the top menu, as shown in Figure 2-56. If you recall, we have added a startup task in the Worker Role service definition for opening a command window, to check whether the command windows was opened during startup, select the HelloWorkerRole.

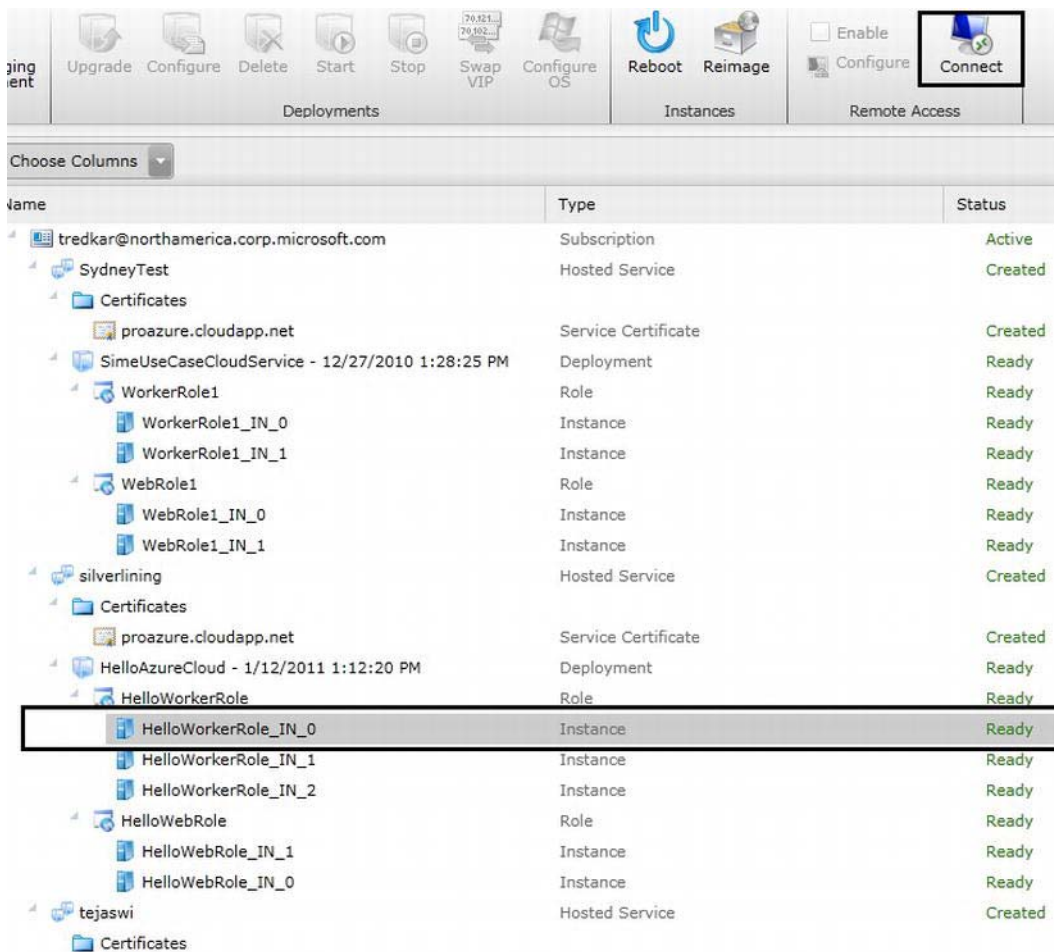


Figure 2-56. Remote Desktop connect

After connection, experiencing Windows Azure instances using Remote Desktop instances is similar to connecting to any other Windows Server 2008. Once you remote login, observe the directory structure and also open the task manager to see the processes running. Figure 2-57 illustrates the overall structure of the Worker Role instance. Similar structure exists for the Web roles.

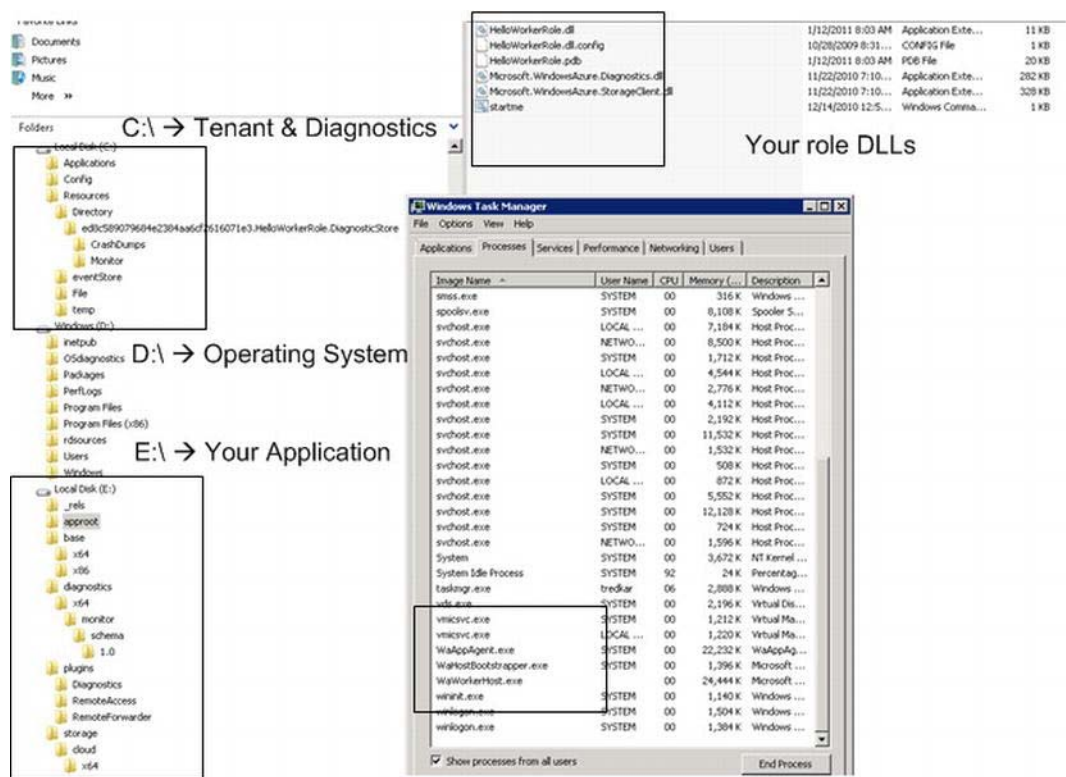


Figure 2-57. Directory structure of a Worker role

■ **Tip** Any modifications to the instance done manually may be erased during a recycle automatically done by the Fabric Controller. Therefore, don't consider any modifications to the instances permanent. Consider each instance as a stateless and maintain all the state in your application package or externally in Windows Azure storage.

Geo-location

Windows Azure is already available in six data centers around the world, and going forward, Microsoft plans to expand into more data centers. In today's enterprise, as well as consumer applications, the common pain point is to design a globally available service. The service needs to be physically deployed into data centers around the world for business continuity, performance, network latency, compliance, or geopolitical reasons. For example, in one project I had the responsibility for architecting a global deployment of a business critical application for a Fortune 100 company. Even though I did not need to travel around the world, I had to plan and coordinate deployment efforts around five data centers across the world. The effort took six months of rigorous planning and coordination. With geo-location support

in Windows Azure, you can choose the geo-location of the storage and the compute at the time of deployment, so you don't need to deploy hardware and software physically in global locations. Table 2-6 lists some of the common geo-location advantages.

Table 2-6. *Geolocation Advantages*

Advantage	Rationale
Business Continuity and Planning	With geo-location features, enterprise data can be replicated across multiple data centers around the world as an insurance shield from natural and political disasters.
Performance and Network Latency	One of the architectural tenets and best practices of cloud services is keeping data close to the application for optimizing performance and end user experience. With geo-location support, a cloud service application can be run in close proximity to the data for improved performance.
Compliance	Compliance laws are different in different countries. Multinational organizations have to deal with compliance regulations in all the countries that they do business in. With Windows Azure, companies can now move data closer to the country offices for adhering to the country specific compliance regulations.
Geopolitical Requirements	Some countries pose restrictions and constraints on enterprises in where they can store enterprise data. Geo-location features can help enterprises better align with such geopolitical requirements.

Geo-location support gives you the ability to choose the affinity of the storage and compute services to a particular geo-location.

Enabling Geographic Affinity

When you create a new storage account or a hosted services project, you can specify the location and affinity group for your project. The steps for creating a geographic affinity between a hosted service project and a storage account follow:

1. Login to the management portal and create a new Hosted Services project.
2. Give the project a name and a label. I have named my project tejaswi.
3. Select a hosted service URL.
4. Next, select Create or choose an Affinity group.

- 5. Select “Create a new affinity group” and choose a name and location for the affinity group, as shown in Figure 2-58.

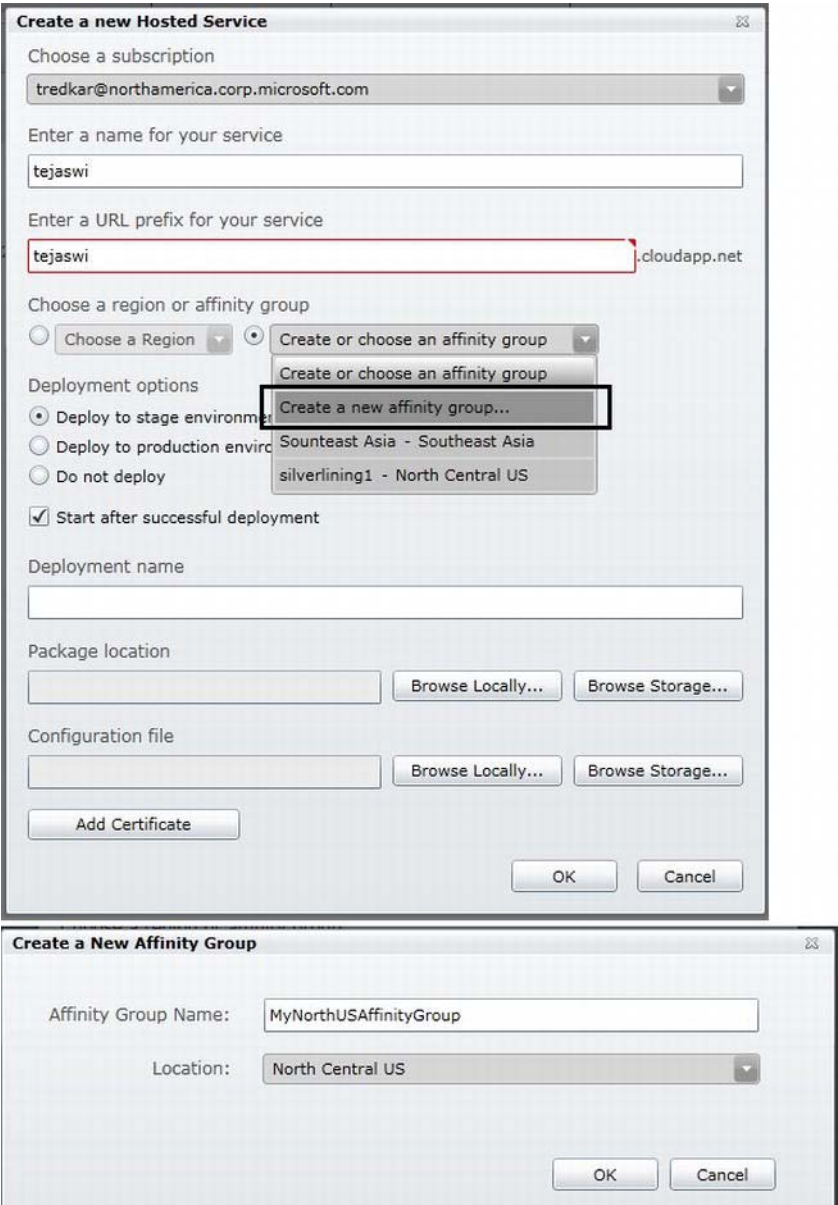


Figure 2-58. Hosted service affinity group

Next, when you create a storage service, you can associate it with the newly created affinity group as shown in Figure 2-59.

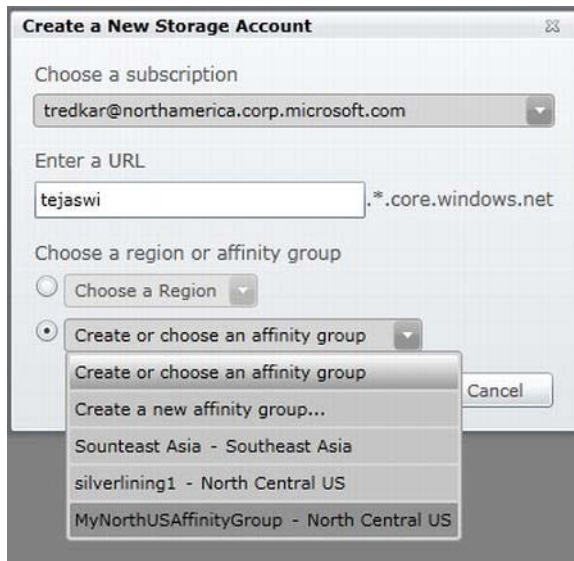


Figure 2-59. Affinity group

Windows Azure Service Management

Unlike on-premise applications, where provisioning requires hardware and network infrastructure in place, the deployment of a cloud services requires only software provisioning. In a scalable environment where enterprises may need to provision multiple services across thousands of instances, you need more programmatic control over the provision process rather than configuring services using Windows Azure management portal or Visual Studio. Manually uploading service packages and then starting and stopping services from the portal interface works well for one or two services. For multiple large-scale services, it becomes a time-consuming and error-prone task. The Windows Azure Service Management API allows you to programmatically perform most of the provisioning functions via a REST-based interface. Using the Service Management API, you can script your provisioning and de-provisioning process end to end in an automated manner. In this section, I will cover some important functions from the Service Management API and also demonstrate some source code for you to build your own cloud service provisioning process.

Service Management API Structure

The Service Management API provides most of the functions you can perform on the storage services and hosted services from Windows Azure developer portal. The Service Management API categorizes the API operations into three primary sections: storage accounts, hosted services, and affinity groups. Operations on storage accounts mainly cover listing of accounts and generation of the access keys. Operation on hosted services cover listing of services, deploying services, removing services, swapping between staging and production, and upgrading services. The affinity groups operations are limited to listing and getting properties of affinity groups in your account.

■ **Note** You can find the Service Management API reference at <http://msdn.microsoft.com/en-us/library/ee460799.aspx>.

The Service Management API uses X.509 client certificates for authenticating calls between the client and the server.

Programming with the Service Management API

To start programming with the Service Management API, you must first create a valid X.509 certificate (or work with an existing one). You can use `makecert.exe` to create a self-signed certificate

```
makecert -r -pe -a sha1 -n "CN=Windows Azure Authentication Certificate" -ss My -len 2048 -sp "Microsoft Enhanced RSA and AES Cryptographic Provider" -sy 24 proazureservicegmt.cer
```

Next, go to the services list in the Hosted Services section, select the Certificates folder under the service and click the Add Certificate button from the top menu. In the upload dialog box, select the certificate you want to upload and click OK.



Figure 2-60. Upload the API certificate

Once the certificate is uploaded, you can call the Service Management REST API by passing the certificate as the `ClientCertificate` property of the `System.Net.HttpWebRequest` object, by using the `csmanage.exe` application from the Service Management API samples, or by building your own application. In Ch2Solution, I have created a sample Windows Application that makes REST calls to the Service Management API. It uses the `Microsoft.Samples.WindowsAzure.ServiceManagement.dll` file from the service management code samples. The `csmanage.exe` uses the same assembly to make the API calls. Eventually, the API assembly may become part of the Windows Azure SDK. Figure 2-61 illustrates the Service Management API windows application in action.

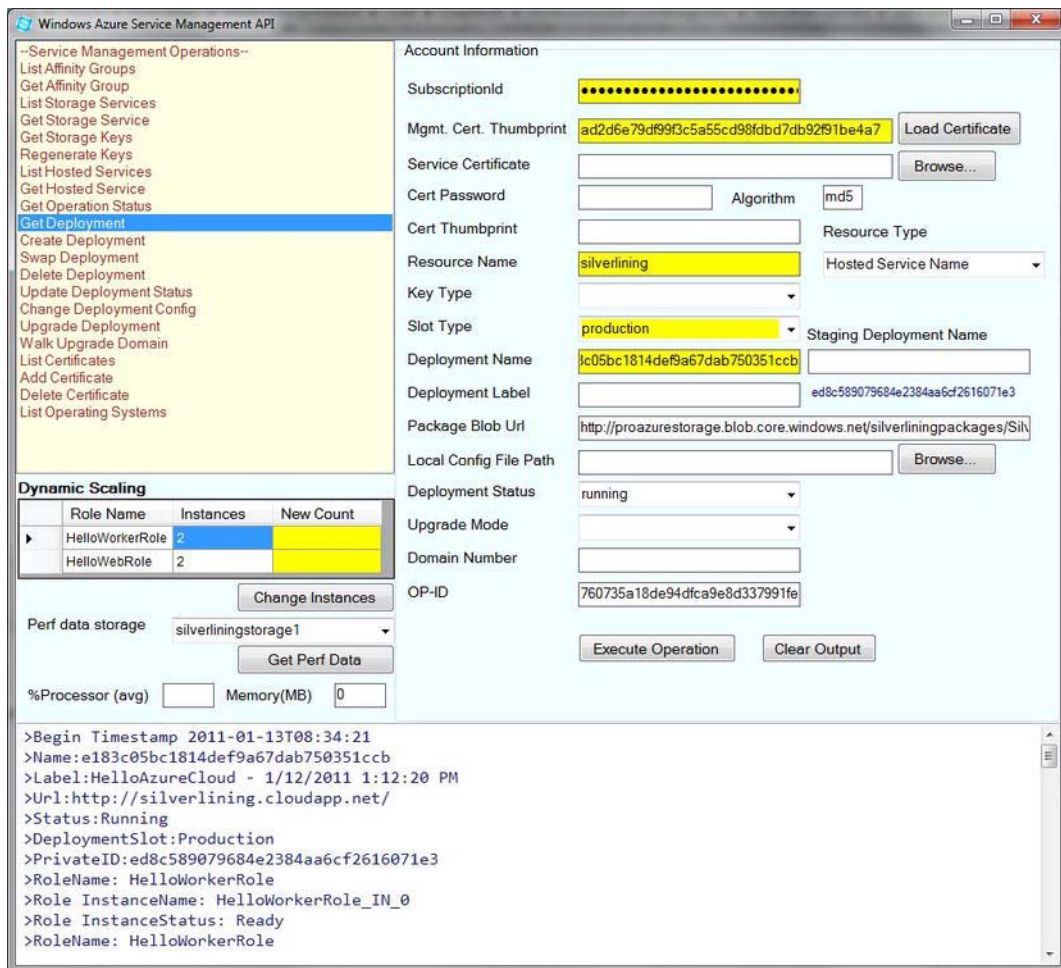


Figure 2-61. The Service Management API windows application

The Service Management Operations section lists the operations that you can invoke on the Service Management API. The output textbox prints the output from the operations. The right-hand side of the user interface consists of input parameters. The input parameters textboxes turn yellow for required parameters. The input parameters are as follows:

- **Subscription Id:** You can get the subscriptionId from the Account page of the developer portal. This parameter is required by all the Service Management API operations.
- **Certificate Path:** This text box points to the API certificate file on the local machine. This certificate must match the one you uploaded to the portal.

- **Resource Type:** This drop-down lists the types of resource you want to access: Hosted Service, Storage Account, or Affinity Group.
- **Resource name:** You should type the name of the resource you want to access (e.g., storage account name, hosted service name, affinity group name).

The remaining input parameters are operation dependent. You can choose an operation from the Service Management operations list, enter input parameters and click Execute Operation. For example, to create a deployment in your hosted service account, you can do the following:

1. Select the Create Deployment operation.
2. Enter your Account SubscriptionId.
3. Select the API certificate from local machine (or add it to the app.config file for automatic loading).
4. Select Hosted Service Name as the Resource Type.
5. Enter the name of the Hosted Service you want to deploy your service to in the Resource Name text box.
6. Select the slot type (staging or production).
7. Choose a deployment name.
8. Choose a deployment label.
9. You have to then point to a service package (.cspkg) on a blob storage in the Package Blob URL text box.
10. Select the path to the ServiceConfiguration.cscfg file of the cloud service.
11. Click Execute Operation.

The OP-ID shows the operation ID returned by the method call, which you can use to track the operation status. To check the status of the deploy operation, select the Get Operation Status method, and click Execute Operation. The status gets displayed in the bottom window. Once the deployment is complete, you can run the deployment by selecting the Update Deployment Status method and selecting the “running” option from the deployment status drop-down. Similarly, you can execute other operations from the Service Management API.

■ **Tip** One of the more important uses of the Service Management API is in dynamic scaling of your Windows Azure Service. The Service Management API can change the configuration of your service on-the-fly, thus increasing or decreasing the number of instances of your roles. You track the performance counters from the storage service for all the instances and then dynamically determine whether to scale-up or scale-down the service. You can find the latest source code for the Service Management application here:
azureplatformbook.codeplex.com

Windows Azure Service Development Life Cycle

The objective of Windows Azure is to automate the service life cycle as much as possible. Windows Azure service development life cycle has five distinct phases and four different roles, as shown in Figure 2-62.

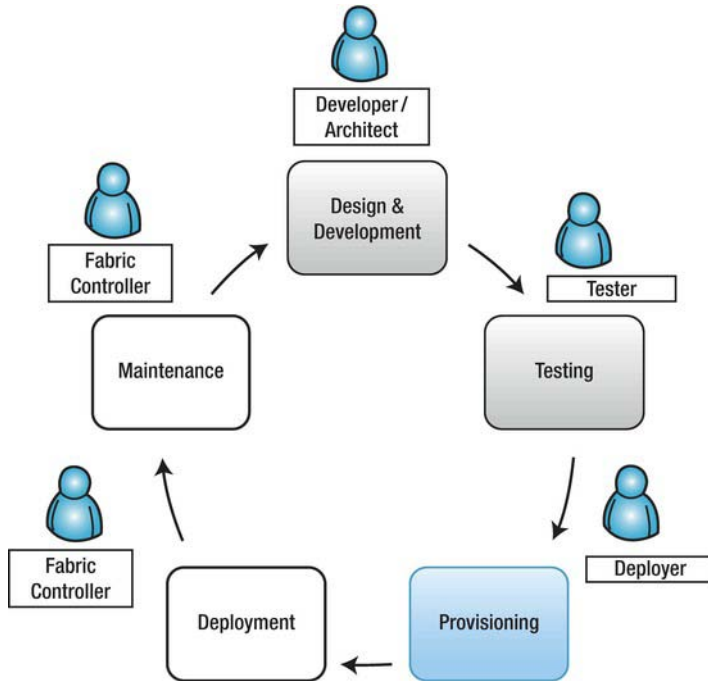


Figure 3-62. The Windows Azure service development life cycle

The five phases are as follows:

Design and development: In this phase, the on-premise team plans, designs, and develops a cloud service for Windows Azure. The design includes quality attribute requirements for the service and the solution to fulfill them. This phase is conducted completely on-premise, unless there is some proof of concept (POC) involved. The key roles involved in this phase are on-premise stakeholders. For the sake of simplicity, I have combined these on-site design roles into a developer role.

Testing: In this phase, the quality attributes of the cloud service are tested. This phase involves on-premise as well as Windows Azure cloud testing. The tester role is in charge of this phase and tests end-to-end quality attributes of the service deployed into cloud testing or staging environment.

Provisioning: Once the application is tested, it can be provisioned to Windows Azure cloud. The release engineer (deployer) role deploys the cloud service to the Windows Azure cloud. The deployer is in charge of service configurations and makes sure the service definition of the cloud service is achievable through

production deployment in Windows Azure cloud. The developer defines the configuration settings, but the deployer sets the production values. In this phase, the role responsibilities transition from on-premise to the Windows Azure cloud. The fabric controller in Windows Azure assigns the allocated resources as per the service model defined in the service definition. The load balancers and virtual IP address are reserved for the service.

Deployment: In the deployment phase, the fabric controller commissions the allocated hardware nodes into the end state and deploys services on these nodes as defined in the service model and configuration. The fabric controller also has the capability of upgrading a service in running state without disruptions. The fabric controller abstracts the underlying hardware commissioning and deployment from the services. The hardware commissioning includes commissioning the hardware nodes, deploying operating system images on these nodes, and configuring switches, access routers, and load-balancers for the externally facing roles (e.g., the Web role).

Maintenance: Windows Azure is designed with the assumption that failure will occur in hardware and software. Any service on a failed node is redeployed automatically and transparently, and the fabric controller automatically restarts any failed service roles. The fabric controller allocates new hardware in the event of a hardware failure. Thus, fabric controller always maintains the desired number of roles irrespective of any service, hardware, or operating system failures. The fabric controller also provides a range of dynamic management capabilities like adding capacity, reducing capacity, and service upgrades without any service disruptions.

In the previous sections, you learned how to design, develop, and deploy a cloud service to Windows Azure on production quality datacenter. In a traditional world, this would take several times longer. Be careful what you wish for, because empowerment also makes it easy to make bad decisions. In the next section, I will give you a checklist for following architecture best practices for developing Windows Azure applications.

Architectural Advice

Finally, here is a list of some practical advice that should serve you well going forward.

1. Performance- and load-test your application on Windows Azure to find out the optimum capacity needed. Don't decide on role sizes before testing.
2. Clearly separate the functionality of the Web role from the Worker role. Do not use Worker role to perform web functions by exposing HTTP (or HTTPS) endpoints.
3. Maintaining stateless role interfaces is important for load balancing and fault tolerance. Keep the roles stateless.
4. Use internal endpoints only for unreliable communications. For reliable communications, use Windows Azure queues.
5. User Worker roles effectively for batch and background processing.

6. Use Service Management API prudently for commissioning, decommissioning, and scaling of the role instances. Do not keep instances running idle for a long period of time, because you are using server resources and will be charged for it. Leverage third-party tools for scaling.
7. Do not use local storage for reliable storage; use Windows Azure storage or SQL Azure as a reliable storage for storing data from roles.
8. Design the system for fault tolerance and always account for failure of role instances.
9. The Worker role works very well as Worker roles for High Scale Compute.
10. For higher I/O operations, use large VM instances.
11. For large caching scenarios, use large VM instances, because they have larger memory capacity.
12. For distributed caching and session management, use Windows Azure AppFabric Caching.
13. Build dynamic scaling capabilities into all the Windows Azure services as a best practice. Most of the configuration for dynamic scaling is external and does not need changes to the source code of the roles.
14. Finally, do not deploy your cloud service for maximum capacity; deploy for optimum capacity, and dynamically provision more instances as demand increases, and vice versa.

Summary

In this chapter, we dove deeply into the computational features of Microsoft's Windows Azure cloud operating system. Through the examples, you were exposed to deploying Windows Azure Web role and Worker role instances, not only in the development fabric but also in the Windows Azure cloud. In the examples, you also learned how to access the configuration settings and local storage. Then, I briefly covered the geo-location and service management features of Windows Azure. In the examples in this chapter, we were storing and retrieving data from the local storage, which is local and machine dependent. The data will be lost as soon as the underlying machine is rebooted or the service redeployed. Windows Azure storage provides you with persistent storage for storing highly available data that can be accessed from anywhere using REST-based API. In the next chapter, you will learn Windows Azure storage components and their programming APIs in detail.

Bibliography

Apache Software Foundation. (n.d.). *Apache Hadoop*. Retrieved from <http://hadoop.apache.org>

Factor, A. (2001). *Analyzing Application Service Providers*. Prentice Hall.

Google. (n.d.). *Google AppEngine*. Retrieved from Google: <http://code.google.com/appengine>

Google. (n.d.). *Google Apps*. Retrieved from Google Apps:
<http://www.google.com/apps/intl/en/business/index.html>

Mario Barbacci, M. H. (1995). *Quality Attributes*. Pittsburgh, Pennsylvania 15213: Software Engineering Institute, Carnegie Mellon University.

Microsoft Corporation. (n.d.). *About Windows Azure*. Retrieved from Windows Azure:
<http://www.azure.com/>

Microsoft Corporation. (n.d.). *Windows Azure Pricing*. Retrieved from Windows Azure:
<http://www.microsoft.com/azure/pricing.mspx>

Open ID Foundation. (n.d.). Retrieved from <http://openid.net/foundation/>

Staten, J. (2008). *Is Cloud Computing Ready For The Enterprise?* Forrester Research, Inc.

Windows Azure Storage

Part I – Blobs and Drives

The previous chapter covered computational and management features of Windows Azure. In this chapter, you will learn about Windows Azure Storage service. Windows Azure Storage is a scalable, highly available, and durable service for storing any kind of application and non-application data. The Storage service provides you with the ability to store data in three different types of storage types: blobs, queues, and tables. Each storage type has advantages; depending on the application requirements, you can choose the appropriate storage type for your data. You can also use multiple storage types within the same application.

The Blob service is designed to store large binary objects with associated metadata like documents, pictures, videos, and music files. The queue is a reliable asynchronous message delivery and storage type. Cloud services as well as on-premises applications can use queues for asynchronous cross-application communications. The table storage type provides structured storage capability to store billions of lightweight data objects occupying terabytes of data.

Windows Azure Drives is a special case of blob storage in which you can upload an NTFS formatted virtual hard disk as a Page blob and then attach it as a drive to a compute instance. The compute instance sees the Windows Azure Drive as any other NTFS formatted drive. The Windows Azure compute instances consist of a special driver that mounts NTFS formatted VHDs from Blob storage as drives. In this chapter, I cover the blob storage type and Windows Azure Drives in detail. This chapter will provide you with enough information to make the right storage decisions for using blobs and drives in your applications. Before you start, I would like to alert you that this chapter is long and detailed, and therefore I would recommend you to read it in phases and not try to read all the pages in a single seating. Relax, drink some coffee or tea, and enjoy the ride.

Table 3-1 lists the Windows Azure storage types and some of their properties.

Table 3-1. Windows Azure Storage

Feature	Blob	Queue	Table
URL schema	<code>http://[Storage Account].blob.core.windows.net/[Container Name]/[Blob Name]</code>	<code>http://[Storage Account].queue.core.windows.net/[Queue Name]</code>	<code>http://[Storage Account].table.core.windows.net/[Table Name]?\$filter=[Query]</code>
Max size	200GB(block blob)/1TB (page blob)	8KB (string)	Designed for terabytes of data but limits combined size of all the data in one entity to 1MB.

Recommended usage	Designed for large binary data types	Designed for cross-service message communication	Designed to store smaller structured objects like the user state across sessions
API reference	http://msdn.microsoft.com/en-us/library/dd135733.aspx	http://msdn.microsoft.com/en-us/library/dd179363.aspx	http://msdn.microsoft.com/en-us/library/dd179423.aspx

■ **Note** The Windows Azure Storage service is independent of the SQL Azure database service offered by the Windows Azure Platform. You can use the Storage Service independent of any other Windows Azure service.

Storage Service Taxonomy

The Windows Azure Storage service allows users to store application data in the cloud and access it from anywhere, anytime. Windows Azure offers REST APIs for interacting with the storage service from your applications. Each storage type in the Storage service has an independent REST programming API. Figure 3-1 illustrates the Windows Azure storage service taxonomy.

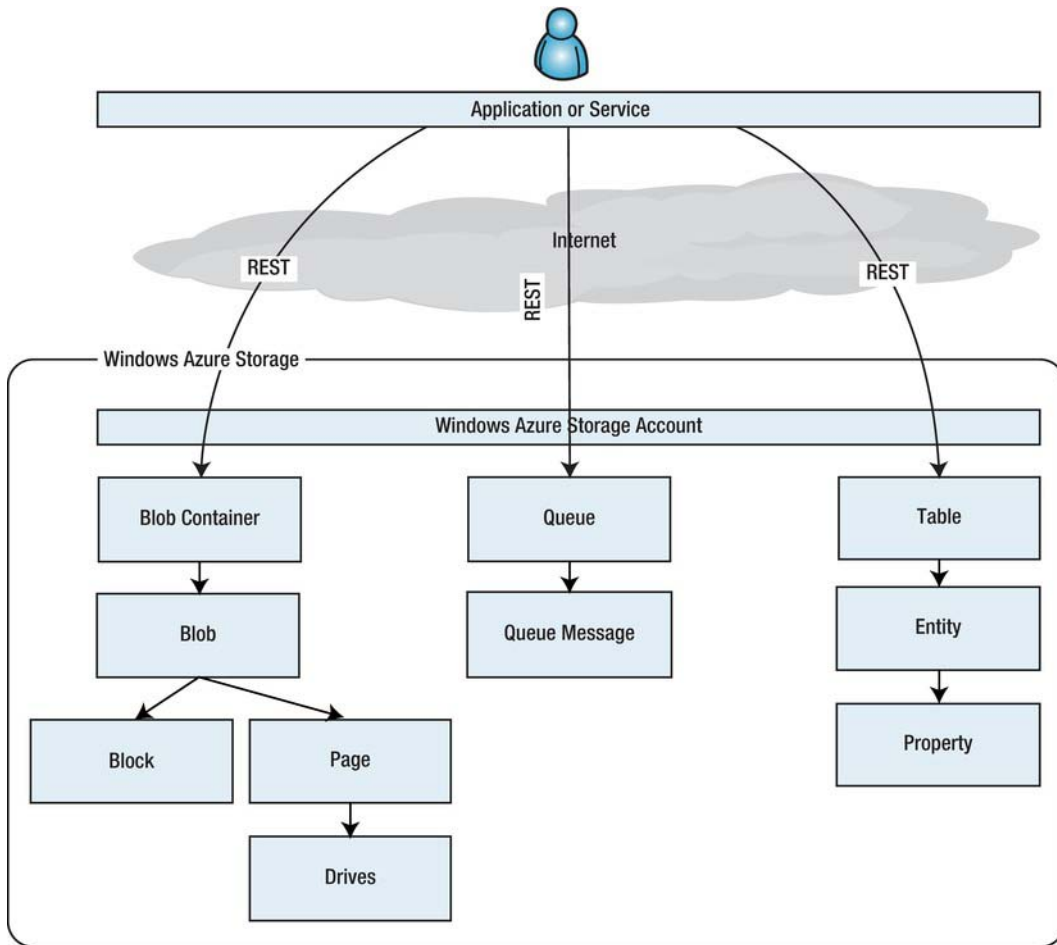


Figure 3-1. Storage service architecture

As shown in Figure 3-1, Windows Azure Storage types are scoped at the account level. This means that when you open a storage account, you get access to all the Windows Azure storage services. The Blob, Queue, and Table services expose REST API functions for application interaction.

A blob account is a collection of containers. You can create any number of containers in an account. A container consists of number of blobs. A blob can be further composed of a series of pages or blocks.

A queue account is a collection of queues. An account can have any number of queues. A queue is composed of queue messages sent by the message sending applications.

The table storage type supports access via REST as well as the ADO.NET Data Services API. You can create any number of tables in an account. A table consists of a set of entities that represent runtime objects or data. Entities are analogous to the rows of data in a relational database. They have properties, which are analogous to the database fields in a relational database table. The table storage type isn't a relational database table; it follows the entity model, where each entity record represents an object instance.

Each storage account gets 100TB of maximum space combining all the storage services within that account. By default, each Windows Azure subscription receives five storage accounts. You can contact Windows Azure support for adding more accounts to your subscription.¹

WHAT IS REST?

The term REST was coined by Roy Thomas Fielding in his Ph.D. dissertation² “Architectural Styles and the Design of Network-based Software Architectures.” Representation State Transfer (REST) is interface architecture for systems deployed and accessible over the network.

The system entry points are abstracted into web resources. In REST, each resource has metadata and is uniquely identified by a URL. The operations of the resource are also exposed via URL. Each URL interaction with the resource returns a representation that can be any document or a binary object. For example, the URLs for blobs, queues, and tables in Table 3-1 represent the REST URLs of these storage type resources. Querying these REST URLs returns appropriate representations from the resources, such as blob files or queue messages. The REST APIs for these storage types also expose operations, which are discussed in detail when I cover the respective storage type.

The URI scheme for addressing the Storage services is

`<http|https>://<account-name>.<storage service name>.core.windows.net/<resource-path>`

`<http|https>` is the protocol used to access Storage services. `<account-name>` is the unique name of your storage account. `<storage service name>` is the name of the storage service you’re accessing (blob, queue, or table). And `<resource-path>` is the path of the underlying resource in the storage services that you’re accessing. It can be a blob container name, a queue name, or a table name.

You used the Blob service in the previous chapter to store logs. In the next section, you study the Blob service in detail and learn to program against the REST programming API for blobs.

Storage Service Architecture

The storage service consists of a three layered architecture. The same architecture hosts blobs, queues, and table services. The three layers of the storage service architecture³ are as follows:

¹ Windows Azure Storage Abstractions and Their Scalability targets
<http://blogs.msdn.com/b/windowsazurestorage/archive/2010/05/10/windows-azure-storage-abstractions-and-their-scalability-targets.aspx>

² Roy Thomas Fielding. “Architectural Styles and the Design of Network-based Software Architectures.”
www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

³ Windows Azure Storage Architecture Overview:-
<http://blogs.msdn.com/b/windowsazurestorage/archive/2010/12/30/windows-azure-storage-architecture-overview.aspx>

- **Front End Servers:** The front-end servers receive all the requests to the storage service. The servers authenticate the requests based on the embedded storage access key. The request is then routed to the appropriate partition server in the partition layer. The front end servers maintain a partition map of the servers maintaining partitions of the data being accessed.
- **Partition Layer:** The partition layer manages the data and their associated partition servers. Each storage object has a partition key and is served by only one partition server. One partition server can serve one or more partitions. The partition layer maintains the mapping between the partition and the data being served by that partition. The Partition Master in the partition layer is responsible for automatic load-balancing between partitioning servers depending on the current load on the partitioning servers.
- **Distributed File System (DFS):** The DFS actually stores the objects on physical disks and replicates across multiple servers for high-availability. The entire DFS is accessible from any partition server. As of writing this book, the data is replicated in the same datacenter, but Microsoft was working on a system for geo-replicating data across multiple datacenters within the same continent.

Figure 3-2 illustrates the Storage service architecture.

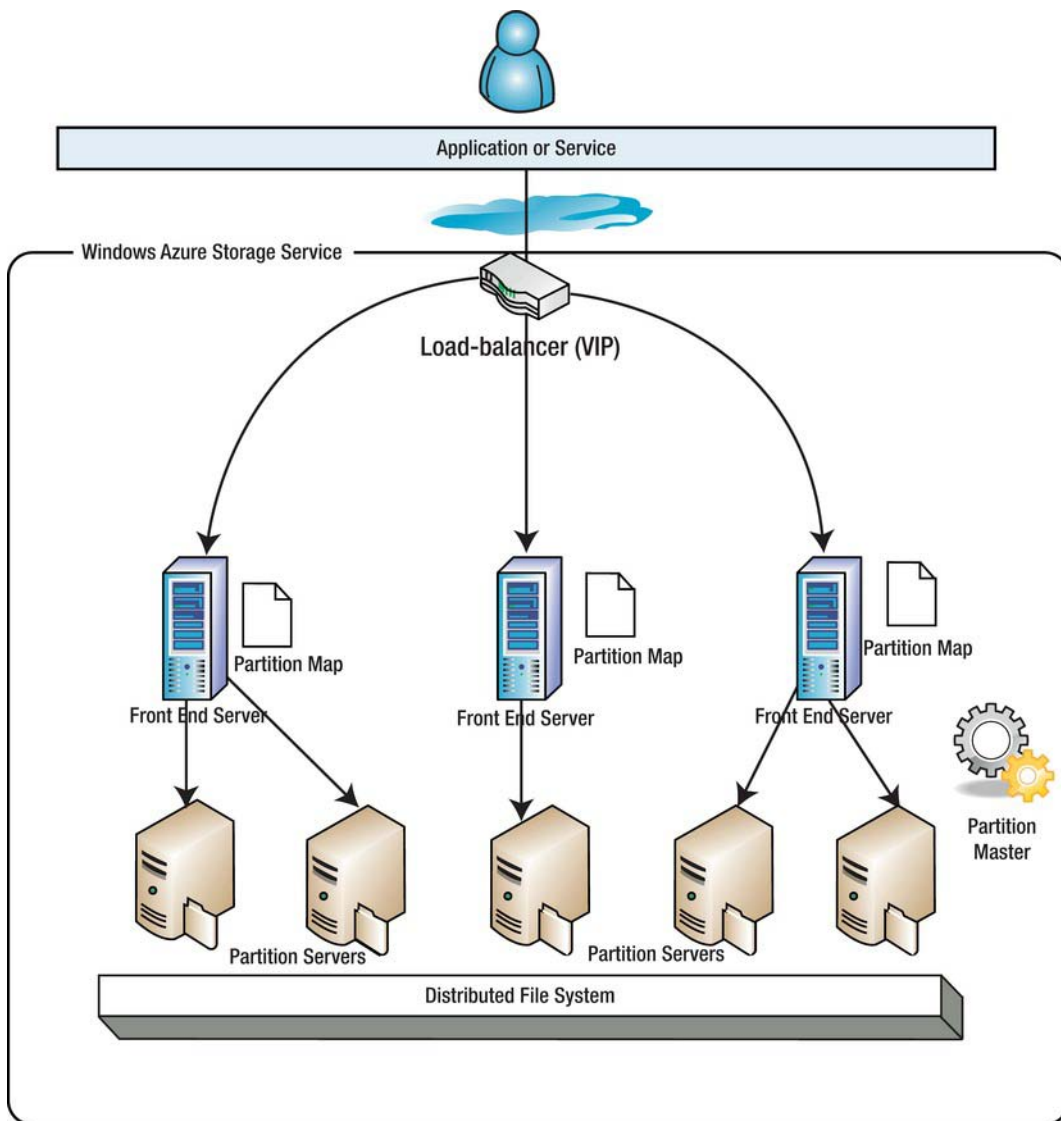


Figure 3-2. Storage service architecture

Every storage object is stored with a Partition Key that is the basis for scaling your queries against the storage service. Each object type (Blob, Table, and Queue) has a different partitioning scheme. The Partition Key determines the partition on which the object resides. The more horizontally spread your objects are, the better query scalability and performance you will get from the service. Partitioning scheme for each object type follows the following format:

- Blobs: Combination of Container Name and Blob Name

- Tables: Combination of Table Name and PartitionKey
- Queues: Queue Name (all messages in a queue are located on the same partition)

The Blob Service

The Blob service provides scalable and highly available storage for any kind of entities, such as binary files and documents. The Blob service achieves its scalability and high availability by distributing blob files across multiple servers and replicating them at least three times. The combination of Container Name and Blob name defines the Partition Key of the blob object. That means that each blob and its snapshots within the same container are stored on the same partition. The target throughput for a single blob is 60MB/sec. It provides a REST API to store named files along with their metadata. The Blob REST API provides consistency-checking features for concurrent operations.

■ **Note** *Windows Azure Blob, blob storage, Blob service, and Blob Storage service* all mean the same thing. The REST API HTTP headers call it the Blob service. Some MSDN documents refer to it as the Blob Storage service, and others call it Windows Azure blob. In this book I have tried to be consistent by calling it the Blob service. The blob object in the Blob service points to the actual file stored in the Blob service.

The Blob service is scoped at the account level. When you create an account on the Azure Services Developer Portal, you get access to the Blob service. Figure 3-3 shows the Azure Services Developer Portal page for the storage account created in the previous chapter and the URL endpoint for the Blob service.



Figure 3-3. Blob endpoint URL

The account endpoint for the Blob service is `<account name>.blob.core.windows.net`, where `<account name>` is the unique name you created for your storage account. The secret key associated with the account provides security for accessing the storage account. You can use the secret key for create an HMAC-SHA256 signature for each request. The storage server uses the signature to authenticate the request.

■ **Note** HMAC stands for Hash Message Authentication Code, which is a message-authentication code calculated from the secret key using a special cryptographic hash function like MD5, SHA-1, or SHA256. The Windows Azure Storage service expects a SHA256 hash for the request. SHA256 is a 256-bit hash for the input data.

Blob Limitations and Constraints

Even though the Blob service provides a scalable and highly available service to store large files in the cloud, it has some limitations and constraints that are important to understand before you dive deep into its architecture and programming. The storage limitations of the Blob service are as follows:

- The maximum size of each block blob is 200GB and each page blob is 1TB (per version 2009-09-19 of the storage service API).
- You can upload blobs that are less than or equal to 64MB in size using a single PUT operation. Blobs more than 64MB in size must be uploaded as a set of blocks, with each block not greater than 4MB in size.
- The development Blob service supports blob sizes only up to 2GB.

Blob Architecture

The blob architecture consists of a four-level hierarchical structure: account, containers, blobs, blocks, and pages, as shown in Figure 3-4.

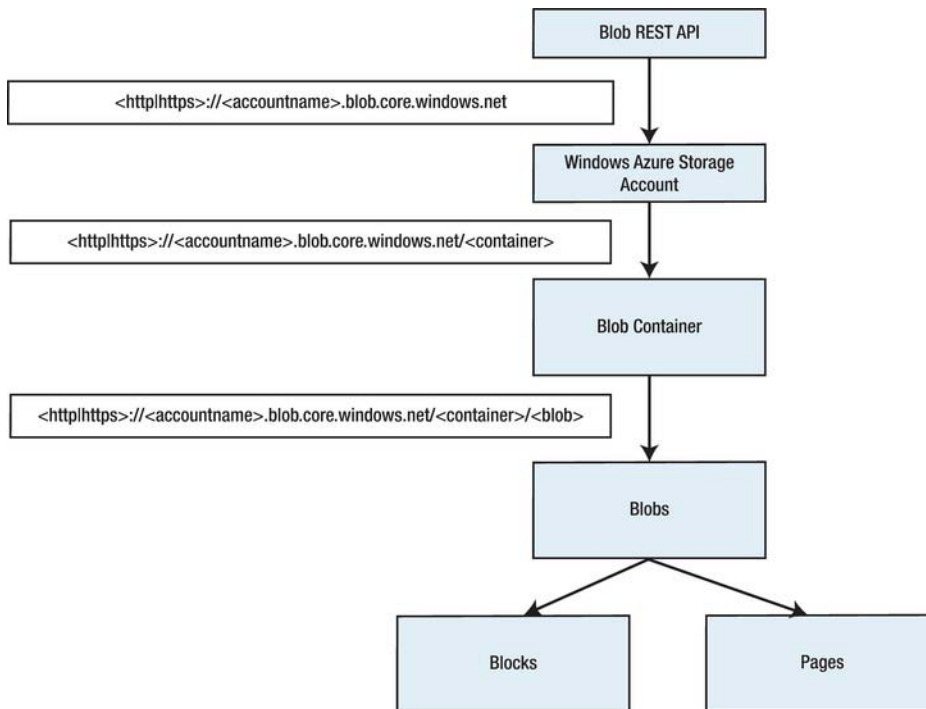


Figure 3-4. Blob service architecture

Your Windows Azure storage account is the entry point to the Blob service via the REST API.

Windows Azure Storage Account

The Windows Azure storage account encompasses the blob, queue, and table storage types. The URI scheme to access the Blob service via storage account is

```
<http/https>://<account name>.blob.core.windows.net
```

where *<account name>* is the unique name you created for your storage account. The *<account name>* must be globally unique.

For example, the Blob service for the storage account created in the previous chapter can be referenced as

```
<http/https>://proazurestorage.blob.core.windows.net
```

Containers

A *container* is a logical grouping for a set of blobs. Containers can have metadata in the form of name-value pairs. They can be created as public or private: public containers are visible to all users (anonymous) for read-only purposes without authentication, and private containers are visible only to the account owner. Blob service is the only storage type that supports public and private access; the queue and table storage types support only private access.

You can access a container the following URI

```
<http/https>://<account name>.blob.core.windows.net/<container>
```

where *<container>* is the name of the container you want to access.

For example, if you create a blob container named logs in the proazurestorage account, you can reference it using the following URI:

```
<http/https>://proazurestorage.blob.core.windows.net/logs
```

The naming constraints on a container are as follows:⁴

- Container names must be unique within an account.
- Container names must start with a letter or a number.
- Container names can't contain any special characters other than the dash (-) character.
- The dash (-) character must be immediately followed by a character or a number.
- All the characters in the container name must be lowercase.
- Container names can't be fewer than 3 or more than 63 characters in length.

If a container name or the URI violates the naming convention, an HTTP status code 400 (Bad Request) is returned by the server.

⁴Source: Windows Azure SDK documentation

■ **Note** Containers do not allow nesting. But, you can logically nest blobs in a container by adding file separators in a blob's name. For example, `tejaswiccontainer/video/Dhruv.mp4` represents a blob named `video/Dhruv.mp4` in the container `tejaswiccontainer`. But, in your application, you will need to parse the tree structure for visual representation. A Blob storage account may have a root represented by the keyword `$root`. You need to explicitly create a `$root` container. You don't need to use the container name explicitly when accessing a root container. The blob URL `http://mystorageaccount.blob.core.windows.net/$root/aaryan.html` can be represented as `http://mystorageaccount.blob.core.windows.net/aaryan.html`. One caveat in a root container is that you cannot have forward slash (/) in the blob names in a root container. Based on the `$root` keyword, the storage processing engine treats these blob addresses a bit differently than blobs in other containers.

Blobs

Blobs, which are the actual entities in the Blob service, are stored in containers. A blob name must be unique within the scope of a container. A blob can also have metadata in the form of name-value pairs. The Access Control List (ACL) is set only at the container level, so all the blobs in a public container are visible to everyone for read-only access. You can access a blob using the following URI

```
<http/https>://<accountname>.blob.core.windows.net/<container>/<blob>
```

where *<blob>* is a unique name of the blob within the specified container. For example, if you create a blob named `200912211752pm-logs` in the container named `Logs`, you can reference it by this URI:

```
<http/https>://proazurestorage.blob.core.windows.net/logs/200912211752pm-logs.txt
```

A blob name can't be more than 1,024 characters long. Blob doesn't support creation of folder hierarchies to store files; you can store files only in a flat structure. In most applications, the hierarchical organization of files is important for ease of access. To facilitate creation of a virtual folder structure, you to add a delimiter to a blob's name. For example, you can name a blob `2009/december/21/ 1752pm-logs.txt`. With this naming scheme, you can add multiple log files in the virtual folder structure `2009/december/21/`. For example, `2009/december/21/1752pm-logs.txt`, `2009/december/21/1852pm-logs.txt`, and `2009/december/21/1952pm-logs.txt` can be the log files created on December 21, 2009.

The Blob API provides filtering capabilities based on a delimiter that allows you to retrieve only the log files in a particular virtual structure. For example, you can retrieve only the log files under the virtual folder structure `2009/december/21` by specifying a delimiter when enumerating the blobs. I cover this in the programming exercises later in the chapter. To support this functionality, the blob name can contain any combination of characters. Any reserved URL characters must be appropriately escaped. Some of the well-known URL reserved characters are dollar (\$), ampersand (&), plus (+), comma (,), forward slash (/), colon (:), semicolon (;), equals (=), question mark (?), and at symbol (@).

Types of Blobs

The storage service offers two types of blobs: page blobs and block blobs.

Page Blobs

Page blobs were introduced in the 2009-09-19 version of the storage service API. They're optimized for random read/write access and provide you with the ability to copy a series of bytes into a blob. A page is represented by its start offset from the start of the blob. Writes to page blobs are immediately committed to the blob storage. You can store up to 1TB of data per page. Page blobs are ideal for applications requiring quick read/write access to binary data like images, videos, documents, and so on. Common applications for Page blobs are random file access to larger document objects, Windows Azure Drives, and so on. The Windows Azure Storage Client API provides two specific operations on page blobs: Put Page and Get Page Regions.

Block Blobs

Block blobs are optimized for streaming file access where you need to read parts of the file instead of downloading the entire file. As listed in the blob limitations and constraints earlier, if a file is more than 64MB in size, it can't be uploaded to the Blob service using the PUT blob function. You have to first break the blob file into contiguous blocks and then upload it in the form of smaller chunks of data called *blocks*. Each block can be a maximum of 4MB in size. After all the blocks are uploaded, they can be committed to a particular blob. Note that in Figure 4-3, there is no URI to access blocks in a blob: after blocks are committed to a blob, you can only retrieve that complete blob. So, you can execute the GET operation only to the blob level.

Uploading blocks and committing blocks to a blob are two separate operations. You can upload the blocks in any sequence, but the sequence in which you commit the list of blocks represents the readable blob. You may upload multiple blocks in parallel in any random sequence, but when you execute the commit operation, you must specify the correct list for the block sequence representing the readable blob. Figure 3-5 illustrates the account, container, blob, and block relationships with an example.

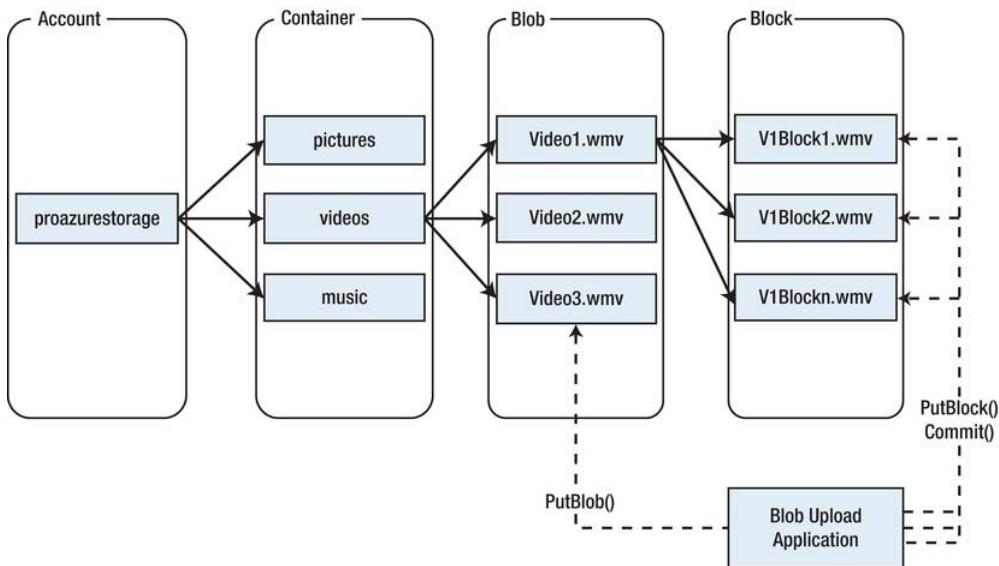


Figure 3-5. Blob service example

In Figure 3-5, you have a storage account name `proazurestorage`. The account has three containers: `pictures`, `videos`, and `music`. In the `videos` container are three video files: `Video1.wmv`, `Video2.wmv`, and `Video3.wmv`. `Video2.wmv` and `Video1.wmv` are less than 64MB in size, so the Blob Upload Application can directly upload these video files as blobs. But `Video1.wmv` is more than 64MB in size, so the Blob Upload Application has to break it into contiguous blocks and then upload each block. After all the blocks are uploaded, the application can commit the block list by giving the sequential order of the blocks that form the `Video1.wmv` blob file. Until all the blocks are committed, the blob file isn't available for reading. Any blocks that remain uncommitted due to application errors or network failures are garbage-collected after seven days of inactivity.

I have covered the uploading and committing of blocks in a programming example later in the chapter.

REST API

The REST API for the Blob service is available at the account, container, and blob levels. In this section, you will learn about the Blob service REST API with specific examples. As a result of the exercise, you will also learn to interact with the Blob service programmatically, and explore the blob methods in the available Storage Client libraries.

The REST API enables you to make HTTP calls to the Blob service and its resources. REST is an HTTP-based protocol that lets you specify the URI of the resource as well as the function you want to execute on the resource. Every REST call involves an HTTP request to the storage service and an HTTP response from the storage service.

■ **Note** Due to frequent changes to the Windows Azure Storage service API, the URL parameters may not be exactly the same as the most recent API version. But conceptually, the variation in the REST API shouldn't be significant. For the exact parameter lists, please refer to the Windows Azure SDK documentation shipped with the SDK.

Request

In the Blob service REST API, the HTTP request components include those outlined next.

HTTP Verb

The HTTP verb represents the action or operation you can execute on the resource indicated in the URI. The Blob service API supports the following verbs: GET, PUT, HEAD, and DELETE. Each verb behaves differently when executed on a different resource.

Request URI

The request URI represents the URI of a resource you're interested in accessing or executing a function on. Example resources in the Blob service API include an account, a container, and a blob. An example URI for creating a container named `proazurecontainer` in an account named `proazurestorage` is:


```
PUT http://proazurestorage.blob.core.windows.net/proazurecontainer
```

The HTTP verb PUT instructs the service to create the container, and the URI points to the resource that needs to be created.

URI Parameters

The URI parameters are the extra parameters you specify to fine-tune your operation execution. They may include operation parameters or filter parameters for the results. In the Blob service API, the URI parameters depend on the type of resource and the HTTP verb used. For example, a URI to retrieve a list of containers from an account looks like this:

```
GET http://proazurestorage.blob.core.windows.net/?comp=list
```

The HTTP verb GET instructs the Blob service to retrieve results, and the parameter ?comp=list instructs that the data requested should be a list of containers.

Request Headers

Request headers follow the standard HTTP 1.1 name-value pair format. Depending on the type of request, the header may contain security, date-time or metadata information, or instructions embedded as name-value pairs. In the Storage Service REST API, the request header must include the authorization information and a Coordinated Universal Time (UTC) timestamp for the request. The timestamp can be in the form of either an HTTP/HTTPS date header or an x-ms-Date header.

The authorization header format is as follows

```
Authorization="[SharedKey|SharedKeyLite] <Account Name>:<Signature>"
```

where SharedKey|SharedKeyLite is the authentication scheme, <Account Name> is the storage service account name, and <Signature> is a Hash-based Message Authentication Code (HMAC) of the request computed using the SHA256 algorithm and then encoded using Base64 encoding.

To create the signature, you have to follow these steps:

1. Create the signature string for signing.
2. The signature string for the Storage service request consists of the following format:

```
VERB\n
Content - MD5\n
Content - Type\n
Date\n
CanonicalizedHeaders
CanonicalizedResource
```

3. VERB is an uppercase HTTP verb such as GET, PUT, and so on. Content – MD5 is the MD5 hash of the request content. CanonicalizedHeaders is the portion of the signature string created using a series of steps described in the “Authentication Schemes” section of the Windows Azure SDK documentation: <http://msdn.microsoft.com/en-us/library/dd179428.aspx>. And CanonicalizedResource is the storage service resource in the request URI. The CanonicalizedResource string is also constructed using a series of steps

described in the “Authentication Schemes” section of the Windows Azure SDK documentation.

- 4. Use the `System.Security.Cryptography.HMACSHA256.ComputeHash()` method to compute the SHA256 HMAC-encoded string.
- 5. Use the `System.Convert.ToBase64String()` method to convert the encoded signature to Base64 format.

Listing 3-1 shows an example request header that sets the metadata values of a container.

Listing 3-1. Request Header

```
x-ms-date: Thu, 04 Jun 2009 03:58:47 GMT
x-ms-version: 2009-04-14
x-ms-meta-category: books
x-ms-meta-m1: v1
x-ms-meta-m2: v2
Authorization: SharedKey proazurestorage:88F+32ZRc+F065+wEiQ1DW/
```

The request header consists of `x-ms-date`, `x-ms-version`, `x-ms-[name]:[value]`, and `Authorization` values. The `x-ms-date` represents the UTC timestamp, and the `x-ms-version` specifies the version of the storage service API you’re using. The `x-ms-version` isn’t a required parameter, but if you don’t specify it, you have to make sure the operation you’re calling is available in the default version of the Blob service. For example, to use the Copy Blob operation, you must specify the 2009-04-14 version string because the Copy Blob operation was added in this particular version and isn’t available in the default version. The `x-ms-meta` values represent the container metadata name-value pairs that the operation wants to set. The last header value is the `Authorization SharedKey` used by the Storage service to authenticate and authorize the caller.

The Blob service REST API also supports HTTP 1.1 conditional headers. The conditional headers are used for conditional invocation of operations. For example, consider a scenario where you’re working on a document that is stored as a blob. After editing the document, you want to save it back to the Blob service, but you don’t know whether someone else on your team modified the document while you were editing it. The Blob service supports four types of conditional headers that act as preprocessing conditions for an operation to succeed. Table 3-2 lists these supported conditional headers.

Table 3-2. Conditional Headers

Conditional Header	Description
If-Modified-Since	A <code>DateTime</code> value instructing the storage service to execute the operation only if the resource has been modified since the specified time.
If-Unmodified-Since	A <code>DateTime</code> value instructing the storage service to execute the operation only if the resource has not been modified since the specified time.
If-Match	An <code>ETag</code> value instructing the storage service to execute the operation only if the <code>ETag</code> value in the header matches the <code>ETag</code> value of the resource.

If-None-Match

An ETag value instructing the storage service to execute the operation only if the ETag value in the header doesn't match the ETag value of the resource. You can use a wildcard (*) to instruct the storage service to execute the operation if the resource doesn't exist and fail if it does exists.

Different operations support different conditional headers; I cover conditional headers for specific operations in their respective sections later in the chapter.

Request Body

The response body consists of data returned by the operation. Some operations require a request body and some of them don't. For example, the Put Blob operation request body consists of the contents of the blob to be uploaded, whereas the Get Blob operation requires an empty request body.

Response

The HTTP response of the Blob service API typically includes the components described in the following sections.

Status Code

The status code is the HTTP status code that indicates the success or failure of the request. The most common status codes for the Blob service API are 200 (OK), 400 (BadRequest), 404 (NotFound), and 409 (Conflict).

Response Headers

The response headers include all the standard HTTP 1.1 headers plus any operation-specific headers returned by the Blob service. Typically, when you create or modify a container or a blob, the response header contains an ETag value and a Last-Modified value that can be used in conditional headers for future operations. The x-ms-request-id response header uniquely identifies a request. Listing 3-2 shows an example response header for a List Containers operation.

Listing 3-2. *List Containers Response Header*

```
Transfer-Encoding: chunked
Content-Type: application/xml
Server: Blob Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 53239be3-4d55-483f-90b9-fc2f2d073215
Date: Thu, 04 Jun 2009 05:28:16 GMT
```

Response Body

The response body consists of data returned by the operation. This data is specific to each operation. For example, the List Container operation returns the list of containers in an account, whereas the Get Blob operation returns the contents of the blob. Listing 3-3 shows an example of the response body for a List Container operation. The response contains three containers and a next marker pointing to the starting point of the remaining containers.

Listing 3-3. List Containers Response Body

```
<?xml version="1.0" encoding="utf-8"?>
<EnumerationResults AccountName="http://proazurestorage.blob.core.windows.net/">
  <MaxResults>3</MaxResults>
  <Containers>
    <Container>
      <Name>000000004c00f241-staging</Name>
      <Url>http://proazurestorage.blob.core.windows.net/000000004c00f241-staging</Url>
      <LastModified>Sun, 26 Apr 2009 15:05:44 GMT</LastModified>
      <Etag>0x8CB94979BAAA0F0</Etag>
    </Container>
    <Container>
      <Name>05022009-staging</Name>
      <Url>http://proazurestorage.blob.core.windows.net/05022009-staging</Url>
      <LastModified>Sun, 03 May 2009 04:50:07 GMT</LastModified>
      <Etag>0x8CB99C1C3ECE538</Etag>
    </Container>
    <Container>
      <Name>050320090743-staging</Name>
      <Url>http://proazurestorage.blob.core.windows.net/050320090743-staging</Url>
      <LastModified>Sun, 03 May 2009 14:44:28 GMT</LastModified>
      <Etag>0x8CB9A14CC091F60</Etag>
    </Container>
  </Containers>
  <NextMarker>proazurestorage/050320091143-staging</NextMarker>
</EnumerationResults>
```

■ **Tip** To test the REST API, I recommend using the Fiddler tool available at www.fiddler2.com/fiddler2/. I have used Fiddler in this book for tracing client/server communications.

Storage Client API

Even though the REST API and the operations in the REST API are easily readable, the API doesn't automatically create client stubs like the ones created by WDSL-based web services. You have to create your own client API and stubs for REST API operations. This makes the client programming more complex and increases the barriers to entry for developers. To reduce this barrier to entry, the Windows Azure SDK team has created a client helper library: `Microsoft.WindowsAzure.StorageClient` from the Windows Azure SDK. `Microsoft.WindowsAzure.StorageClient` abstracts the REST interface by providing

a client-side object model on top of the REST API. You can also build your own object model leveraging the REST API directly. For most of the applications, the `Microsoft.WindowsAzure.StorageClient` API is sufficient.

The following sections cover the `Microsoft.WindowsAzure.StorageClient` API.

■ **Note** You don't have to use the `StorageClient` library to make REST calls to the Storage service. You can create your own client library to make REST operations to the Storage service directly.

Windows Azure Storage Client Blob API

The `Microsoft.WindowsAzure.StorageClient` namespace consists of classes representing the entire Blob hierarchy. Figure 3-6 illustrates the core classes for programming Blob service applications.

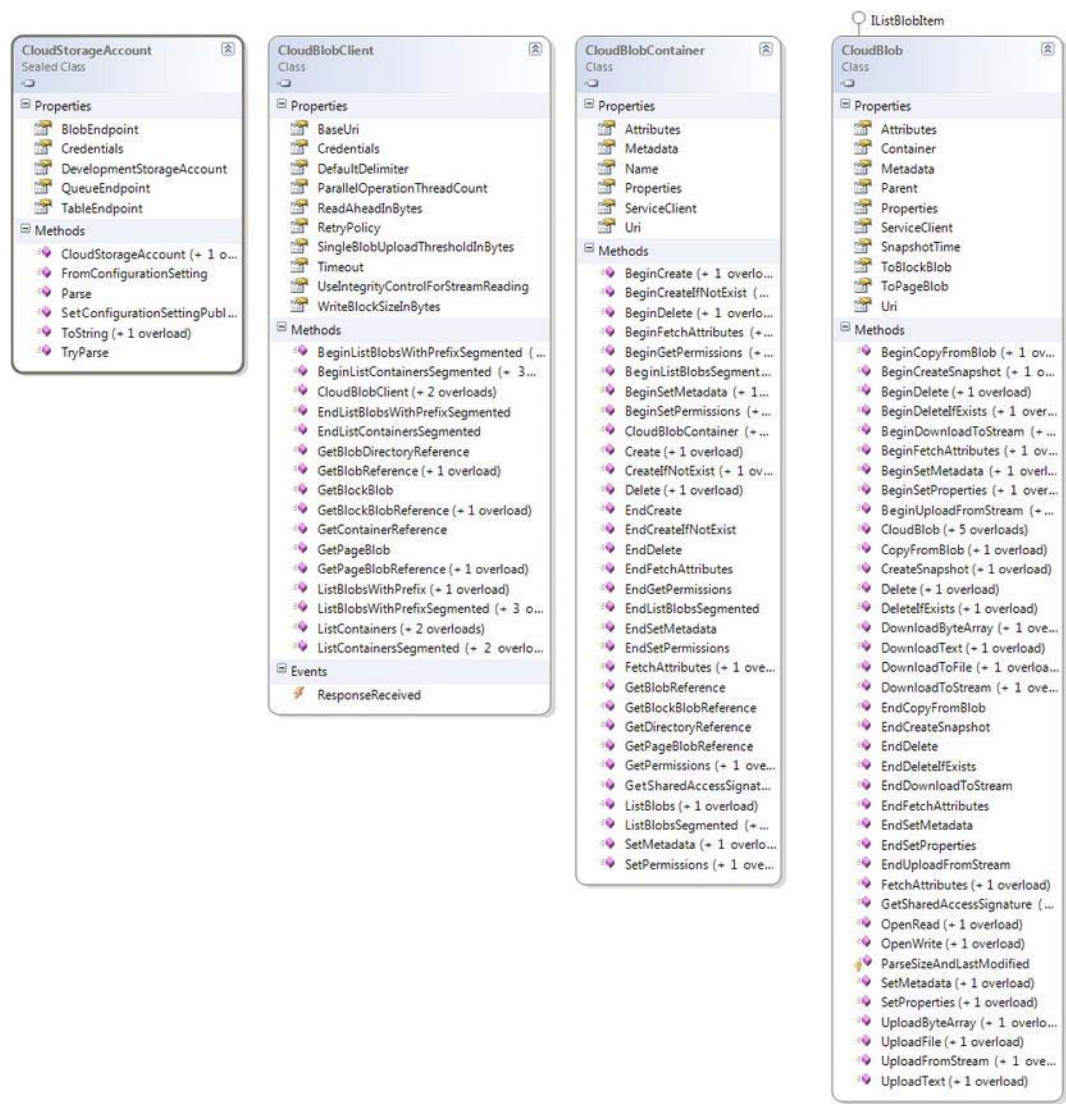


Figure 3-6. Blob class hierarchy

Table 3-3 describes these classes.

■ **Tip** The Windows Azure Storage Client API is the recommended method for programming storage service applications. The API provides synchronous as well as asynchronous methods for interacting with the Storage service REST APIs.

Table 3-3. Classes for the Blob Service

Class Name	Description
CloudStorageAccount	A helper class to retrieve account information from the configuration files or create an instance of the storage account object from account parameters.
CloudBlobClient	A wrapper class for getting references to the core blob objects. The class consists of methods like GetContainerReference() and GetBlobReference().
CloudBlobContainer	A class that consists of container operations like Create(), Delete(), ListBlobs(), and GetBlobReference().
CloudBlob	A class that consists of blob operations like Create(), Copy(), UploadFromFile(), UploadByteArray(), UploadStream(), and so on. This is the class you use the most to interact with blobs.

In addition to these core classes, classes like BlobProperties and BlobContainerProperties represent more details about the blob and the container, respectively. CloudPageBlob and CloudBlockBlob define operations for page blobs and block blobs, respectively.

The steps for programming simple blob applications with these blob classes are as follows:

1. Add the following using statement to your C# class:

```
using Microsoft.WindowsAzure.StorageClient;
```

2. Instantiate the CloudStorageAccount class from the configuration files by publishing the appropriate configuration publisher:

```
CloudStorageAccount _cloudStorageAccount
if (RoleEnvironment.IsAvailable)
{
    CloudStorageAccount.SetConfigurationSettingPublisher((configName,
configSetter) =>
    {
        configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));
    });
    cloudStorageAccount =
CloudStorageAccount.FromConfigurationSetting(configString);
```

```

    }
    else
    {
        CloudStorageAccount.SetConfigurationSettingPublisher((configName,
configSetter) =>
        {
            configSetter(ConfigurationManager.AppSettings[configName]);
        });
        cloudStorageAccount =
CloudStorageAccount.FromConfigurationSetting(configString);
    }

```

■ **Caution** When instantiating the configuration publisher in ASP.NET (IIS) application, make sure you instantiate it in `Global.asax.cs` instead of `WebRole.cs` because `WebRole.cs` (`RoleEntryPoint`) runs in a separate process than the ASP.NET application (including Web Services).

3. Or, instantiate the `CloudStorageAccount` class using account information:

```

CloudStorageAccount storageAccountInfo = new
CloudStorageAccount.Parse(storageAccountConnectionString);

```

Where `storageAccountConnectionString` is of the format

```
DefaultEndpointsProtocol=https or http;AccountName={0};AccountKey={1}
```

4. Create an instance of `CloudBlobClient`:

```
CloudBlobClient blobStorageType = storageAccountInfo.CreateCloudBlobClient();
```

When you have an instance of the `CloudBlobClient` class, you can execute operations on the Blob Storage service as follows:

List containers:

```

IList<CloudBlobContainer> containers = new
List<CloudBlobContainer>(this.blobStorageType.ListContainers(prefix,
ContainerListingDetails.All));

```

Create a container:

```
blobStorageType.GetContainerReference(containerName).CreateIfNotExist();
```

Create a container with permissions:

```

CloudBlobContainer container = blobStorageType.GetContainerReference(containerName);
BlobContainerPermissions perm = new BlobContainerPermissions();
perm.PublicAccess = accessType;
container.SetPermissions(perm);
container.Metadata.Add(new NameValueCollection());
container.CreateIfNotExist();

```


Create a blob by uploading a byte array:

```
blobStorageType.GetContainerReference(containerName).GetBlobReference(blobName).UploadByteArray(blobContents);
```

Create a blob by uploading text:

```
blobStorageType.GetContainerReference(containerName).GetBlobReference(blobName).UploadText(blobContents);
```

Create a blob by uploading a stream:

```
blobStorageType.GetContainerReference(containerName).GetBlobReference(blobName).UploadFromStream(blobContents);
```

Create a blob by uploading a file:

```
blobStorageType.GetContainerReference(containerName).GetBlobReference(blobName).UploadFile(fileName);
```

Get a blob by downloading a byte array:

```
blobStorageType.GetContainerReference(containerName).GetBlobReference(blobName).DownloadByteArray();
```

Get a blob by downloading text:

```
blobStorageType.GetContainerReference(containerName).GetBlobReference(blobName).DownloadText();
```

Get a blob by downloading a stream:

```
blobStorageType.GetContainerReference(containerName).GetBlobReference(blobName).DownloadToStream(outputStream);
```

Get a blob by downloading a file:

```
blobStorageType.GetContainerReference(containerName).GetBlobReference(blobName).DownloadToFile(outputFileName);
```

Create a temporary Shared Access Url for a blob that expires in 10 minutes

```
CloudBlob cb = new CloudBlob(absoluteBlobUri)
var readPolicy = new SharedAccessPolicy()
{
    Permissions = SharedAccessPermissions.Read,
    SharedAccessExpiryTime = DateTime.UtcNow + TimeSpan.FromMinutes(10)
};
return cb.Uri.AbsoluteUri + cb.GetSharedAccessSignature(readPolicy);
```

■ **Tip** I recommend you to only access blobs using Shared Access Signature instead of full open account access. Your storage account key is an entry to your entire storage account with administrator privileges. A hacker can wipe out your entire storage account or even store his/her own files in your account with these privileges. With

Shared Access Signature or URL, you can restrict the URL with a time window as well as add access control for the URL. I also recommend you to recycle your storage key as frequently as once a month. You can automate the process of refreshing the key and then publishing it to your applications through the Windows Azure Service Management API covered in Chapter 2.

Shared Access Signatures

Having account level security is not really useful if you want to leverage the Blob storage space within one account for multiple users. Let's say for example you want to build a file synchronization service that leverages blob storage for any number of users. You should be able to provide granular security for every user subscribed to the application. Create one storage account for every user will not be feasible for a large number of users because you will have to manage a large number of subscriptions and accounts. Shared Access Signatures allows you to specify granular level of security at the blob level by embedding time-bound signatures in the blob URL. *Shared Access Signatures* are a series of URL parameters specified in the URI of the resources for controlling access privileges to the resources. Shared Access Signatures are available for container and blob resources. In the URL, you can specify the start time when the resource becomes visible, the expiration time after which the Shared Access Signature expires, permissions that are granted to the URL, the resource that is made available to the URL, and the signature used to authenticate the request to the resource. The available permissions are read (r), write (w), delete (d), and list (l). An example of a blob PUT operation's URL with Shared Access Signatures is as follows:

```
PUT http://proazure.blob.core.windows.net/videos/myvideo.wmv?st=2009-12-21T05%3a52Z&se=2009-12-31T08%3a49Z&sr=c&sp=w&si=YWJjZGVmZw%3d%3d&sig=Rcp6gPEaNGJAI$KAM%PIR$APANG%Ca%IL%O$V%Eyou%234so$m$uch2bqEArnfJxDgE%2bKH3TCChIs%3d HTTP/1.1
Host: proazure.blob.core.windows.net
Content-Length: 19
My Name is Tejaswi..
```

In these requests, the Shared Access Signatures are as follows:

- *st (signedstart)*: (Optional) This is the start time when the resource becomes available with a valid Shared Access Signature.
- *se (signedexpiry)*: (Required) This is the end time when the Shared Access Signature becomes invalid and, as a result, the URL can't access the resource.
- *sr (signedresource)*: The parameter can have two values: b to specify access to a specific blob, and c to specify access to any blob in the container and to the list of blobs in the container.
- *sp (signedpermissions)*: (Required) This parameter specifies the type of permissions to the resource: read (r), write (w), delete (d), or list (l).
- *si (signedidentifier)*: (Optional) This is a unique string with a maximum of 64 characters that correlates to the access policy of the container, thus giving you an additional level of control over the Shared Access Signatures and the ability to revoke the signature.

- sig (signature): This is the signature used to authenticate the request. You can create a signature using the following method:

```
HMAC-SHA256(URL.Decode(UTF8.Encode(string-to-sign)))
```

HMAC-SHA256 is the algorithm used to compute the signature, and string-to-sign is of the format

```
string-to-sign = signedpermissions + "\n"
                signedstart + "\n"
                signedexpiry + "\n"
                canonicalizedresource + "\n"
                signedidentifier
```

Shared Access Signatures give you the ability to exercise fine-grained access control at the blob and container levels. Shared Access Signatures are very useful in creating time-bound temporary URLs for downloading file(s) from the Blob service.

Listing 3-4 shows the code for creating a Shared Access Signature and Signed Identifier using the Windows Azure SDK managed client API.

Listing 3-4. Shared Access Signature

```
// Get reference to your blob storage account
var storageAccount = CloudStorageAccount.DevelopmentStorageAccount;
var container = storageAccount .CreateCloudBlobClient()
    .GetContainerReference("proazurecontainer");
container.CreateIfNotExist();
var blob = container.GetBlobReference("proazuredocument.txt");
blob.Properties.ContentType = "text/plain";
blob.UploadText("Are you enjoying Blobs?");

// Create a shared access signature with a new shared access policy
var sharedAccessSignature = blob.GetSharedAccessSignature(new SharedAccessPolicy()
{
    Permissions = SharedAccessPermissions.Read
                | SharedAccessPermissions.Write,
    SharedAccessExpiryTime = DateTime.UtcNow + TimeSpan.FromMinutes(30)
});

//This link will expire after 30 minutes

// Using shared access signature for blob operations from your client application
var sharedAccessSignatureCreds = new
StorageCredentialsSharedAccessSignature(sharedAccessSignature);
var secureBlob = new CloudBlobClient(storageAccount.BlobEndpoint, sharedAccessSignatureCreds)
    .GetBlobReference("proazurecontainer/ proazuredocument.txt");
secureBlob.UploadText("Thank You!");
Console.WriteLine(secureBlob.DownloadText());

//To set the optional signed identifier
var permissions = container.GetPermissions();
permissions.SharedAccessPolicies.Add("write", new SharedAccessPolicy())
```

```

    {
        Permissions = SharedAccessPermissions.Write
    });
container.SetPermissions(permissions, new BlobRequestOptions()
{
    // This step is to check if someone modified the permissions while we were setting it
    AccessCondition = AccessCondition.IfMatch(container.Properties.ETag)
});

var sharedAccessSignatureIdentifier = blob.GetSharedAccessSignature(new SharedAccessPolicy()
{
    SharedAccessExpiryTime = DateTime.UtcNow + TimeSpan.FromHours(24)
}, "write");
```

```

Console.WriteLine("This link will expire in 24 hours");
Console.WriteLine("Full Uri " + blob.Uri.AbsoluteUri + sharedAccessSignatureIdentifier);
```

In Listing 3-8, you get reference to the blob and then add a shared access signature to the blob that lasts for 30 minutes. The code snippet also shows how to access the blob using the shared access signature from the client application. The next part of the code shows the use of shared access identifiers to apply more granular security to the blobs by giving read, write, and listing permissions or even revoking permissions.

■ **Note** You can find the latest source code companion for this book on the CodePlex site azureplatformbook.codeplex.com

Account Operations

The storage account provides an entry point to the Blob service via the Blob service endpoint URI. At the account level of the hierarchy, the Blob service supports only one operation: List Containers. The URI of a specific account is of the format `http://<account name>.blob.core.windows.net`. Table 3-4 describes the List Containers operation, and Table 3-5 lists some important characteristics of the List Containers function.

Table 3-4. Blob Account Operation

Operation	Description
List Containers	This operation gets a list of all the containers in a storage account. You can limit the number of records returned by specifying a filter on container names and the size of the dataset in the request. Table 4.6 lists all the possible URI parameters for this operation.

Table 3-5. Blob Account Operation Characteristics

Operation	HTTP	Cloud URI	Development Storage URI	HTTP Version	Permissions
List Containers	GET	<code>http://<account name>.blob.core.windows.net?comp=list</code>	<code>http://127.0.0.1:10000/<devstorageaccount>?comp=list</code>	HTTP/1.1	Only the account owner can call this operation.

`<account name>` is the storage account name, such as `proazurestorage`, and `<devstorageaccount>` is the account name for the development storage. The HTTP verb used in this operation is GET. The table lists the URI format to access Cloud Blob service as well as the development storage URI. Port 10000 is the default Blob service port in the development fabric.

The URI for the List Containers operation also supports additional optional parameters, as listed in Table 3-6.

Table 3-6. List Containers URI Parameters

Parameter	Description	Example
Prefix	A filter parameter to return containers starting with the specified prefix value.	<code>http://proazurestorage.blob.core.windows.net/?comp=list&prefix=may</code> returns containers with names starting with the prefix “may.”
Marker	Used to page container results when not all results were returned by the Storage service either due to the default maximum results allowed (the current default is 5000) or because you specify the <code>maxresults</code> parameter in the URI. The marker prefix is opaque to the client application.	<code>http://proazurestorage.blob.core.windows.net/?comp=list&prefix=may&marker=/proazurestorage/may0320091132-staging</code>
<code>maxresults</code>	The maximum number of containers the Blob service should return. The default value is 5000. The server returns an HTTP Bad Request (400) code if you specify a <code>maxresults</code> value greater than 5000.	<code>http://proazurestorage.blob.core.windows.net/?comp=list&prefix=may&maxresults=100</code>

The sample REST request for List Containers in raw format looks like Listing 3-5.

Listing 3-5. List Containers REST Request

```
GET /?comp=list&prefix=may&maxresults=6&timeout=30 HTTP/1.1
x-ms-date: Wed, 27 May 2009 04:33:00 GMT
Authorization: SharedKey proazurestorage:GCvS8cv4Em6rWMuCVix9YCsxVgssOW62S2U8zjbIa1w=
Host: proazurestorage.blob.core.windows.net
Connection: Keep-Alive
```

The characteristics of the REST request in Listing 4-5 are as follows:

- The parameter comp=list at the account level of the Blob service yields the list of all the containers.
- The prefix=may filters the results by container names starting with “may.”
- maxresults=6 returns only six containers.
- x-ms-date is the UTC timestamp of the request.
- The Authorization header contains the SharedKey of the request.
- The Host header points to the Blob service in the cloud.
- Because the request is sending a maxresults parameter, it makes sense to keep the HTTP connection alive because it’s highly likely that the user will retrieve the next set of results by making one more call to the Blob service.

Listing 3-6 shows the response for the List Containers request.

Listing 3-6. List Containers REST Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Server: Blob Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 62ae926f-fcd8-4371-90e1-bdb6d32e31e6
Date: Wed, 27 May 2009 04:34:48 GMT
Content-Length: 1571

<?xml version="1.0" encoding="utf-8"?>
<EnumerationResults AccountName="http://proazurestorage.blob.core.windows.net/">
  <Prefix>may</Prefix>
  <MaxResults>6</MaxResults>
  <Containers>
    <Container>
      <Name>may022009-01-52-staging</Name>
      <Url>http://proazurestorage.blob.core.windows.net/may022009-01-52-staging</Url>
      <LastModified>Sat, 02 May 2009 08:54:23 GMT</LastModified>
      <Etag>0x8CB991AB99A3DE8</Etag>
    </Container>
    <Container>
      <Name>may022009-01-56-staging</Name>
```

```

<Url>http://proazurestorage.blob.core.windows.net/may022009-01-56-staging</Url>
<LastModified>Sat, 02 May 2009 08:58:08 GMT</LastModified>
<Etag>0x8CB991B3F6EECF8</Etag>
</Container>
<Container>
<Name>may031119am-staging</Name>
<Url>http://proazurestorage.blob.core.windows.net/may031119am-staging</Url>
<LastModified>Sun, 03 May 2009 18:21:46 GMT</LastModified>
<Etag>0x8CB9A3326D83577</Etag></Container>
<Container><Name>may0320091132-staging</Name>
<Url>http://proazurestorage.blob.core.windows.net/may0320091132-staging</Url>
<LastModified>Sun, 03 May 2009 18:33:55 GMT</LastModified>
<Etag>0x8CB9A34D97B4CC0</Etag>
</Container>
<Container>
<Name>may0320091413pm-staging</Name>
<Url>http://proazurestorage.blob.core.windows.net/may0320091413pm-staging</Url>
<LastModified>Sun, 03 May 2009 21:14:53 GMT</LastModified>
<Etag>0x8CB9A4B5676BA40</Etag>
</Container>
<Container>
<Name>may0320091500pm-staging</Name>
<Url>http://proazurestorage.blob.core.windows.net/may0320091500pm-staging</Url>
<LastModified>Sun, 03 May 2009 22:01:55 GMT</LastModified>
<Etag>0x8CB9A51E81571B3</Etag>
</Container>
</Containers>

<NextMarker />

</EnumerationResults>

```

In Listing 3-6, the header consists of the HTTP status (200 OK) indicating the success of the operation. The response body is in XML format with `<EnumerationResults />` as the root element. The `<Containers />` element contains the retrieved containers. The `<Etag>` or the entity tag and the `<LastModified>` values are used to find changes to the content source after it was retrieved. These fields are used to detect concurrency conditions where a resource may change between retrieve and save operations. The empty `<NextMarker/>` element indicates that all the results have been retrieved.

Programming Example

To help you understand the Blob service programming model, I've created a project named Windows Azure Storage Operations in `Ch3Solution.sln`. The name of the Windows application project is Windows Azure Storage Operations. I've also created a helper class named `WASStorageHelper` in the `ProAzureCommonLib` project, to wrap the `StorageClient` methods. Figure 3-7 shows the user interface for the Windows Azure Storage Operations.exe application as it pertains to the account operations of the Blob service.

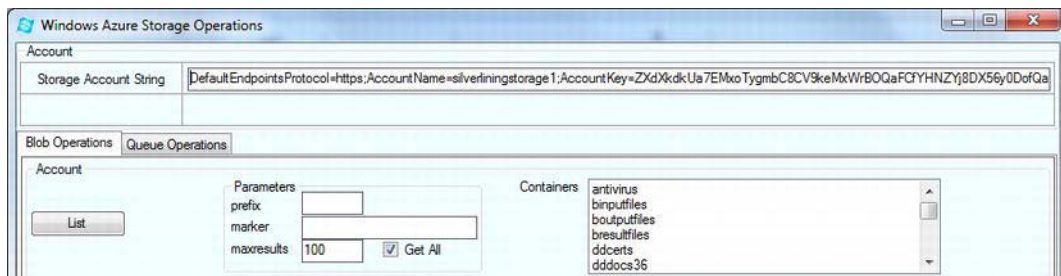


Figure 3-7. Windows Azure Storage Operations blob account operations

In Figure 3-7, the top Account section displays the Account name and SharedKey of the storage account. When the Windows Azure Storage Operations application starts, it loads the account information from the configuration file.

The AccountName and AccountSharedKey values are loaded when the application starts; the application displays these values in the Account and Key text fields, respectively. When you start the application, make sure to enter the account name and shared key of your own storage account or change then in the app.config file before building the project. The account information is used to initialize the WStorageHelper class, as shown here:

```
StorageHelper = new WStorageHelper(CloudStorageAccount.Parse(txtAccountName.Text));
```

The WStorageHelper class in the ProAzureCommonLib project has two overloaded GetContainers() methods to retrieve container names. Listing 3-7 shows the code for these two methods.

Listing 3-7. *GetContainers() Methods*

```
public IEnumerable<BlobContainer> GetContainers()
{
    return this.BlobClient.ListBlobContainers();
}
public ResultSegment<CloudBlobContainer> GetContainerSegmented(string prefix, int maxResults,
    ResultContinuation continuationToken)
{
    return BlobClient.ListContainersSegmented(prefix, ContainerListingDetails.All,
    maxResults, continuationToken);
}
```

Both methods get the list of containers in your storage account. The first ListContainers() method returns all the containers from the account; therefore it doesn't accept any filtering parameters. The second ListContainersSegmented() method accepts prefix, maxresults, and continuation token. The continuation token is used to page results. You can iterate over the container list as shown in Listing 3-8.

Listing 3-8. *Iterating over a Large List of Containers*

```
static void ListContainersSegmented(string storageAccountStr)
{
    CloudBlobClient blobClient = CloudStorageAccount.Parse(storageAccountStr);
```



```

//Return the first group of 25 containers.
ResultSegment<CloudBlobContainer> resultSegment = blobClient.ListContainersSegmented("",
ContainerListingDetails.All, 25, null);

foreach (var container in resultSegment.Results)
{
    Console.WriteLine(container.Name);
}

//Are there more results in the segment?.
if (resultSegment.HasMoreResults)
{
    resultSegment = resultSegment.GetNext();

    foreach (var container in resultSegment.Results)
    {
        Console.WriteLine(container.Name);
    }
}

//Continuation token determines whether there are more results/segments on the server.
while (resultSegment.ContinuationToken != null)
{
    resultSegment = resultSegment.GetNext();

    foreach (var container in resultSegment.Results)
    {
        Console.WriteLine(container.Name);
    }
}
}

```

■ **Note** maxresult=0 will return a maximum of 5000 containers.

The results returned from these methods are displayed in the list box in the Account section of the Windows Azure Storage Operations.exe application, as shown in the Figure 3-8.

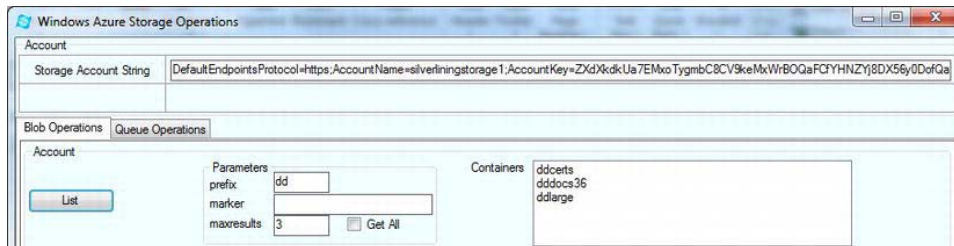


Figure 3-8. List containers

Click on the List button to retrieve containers from the Blob service in the list box. In the Parameters section, you can specify the prefix and the maxresults for filtering the result set. Click the List Containers button again to retrieve the remaining results.

Container Operations

The URI of a specific container is of the format <account name>.blob.core.windows.net/<container name>. Containers support several operations, as listed in Table 3-7.

Table 3-7. Container Operations

Operation	Description
Create Container	Creates a new container under the given account. You can specify metadata and access control for the container during creation.
Get Container Properties	Returns the user-defined metadata values and the system properties of the container. The ETag and Last-Modified values are examples of system generated container properties.
Set Container Metadata	Sets the user-defined metadata values of a container. This operation sets or overwrites all the metadata values at once. You can't change specific name-value pairs of a container. The ETag value of a container changes when this operation executes successfully.
Get Container ACL	Returns a container's access control bit value. It returns False if a container is private and True if public.
Set Container ACL	Sets a container's access control bit value. You can set the value of the header parameter x-ms-prop-publicaccess to True for a public container or False for a private container.
Delete Container	Marks a container for deletion. The delete operation doesn't delete the container instantly; it's deleted during the next garbage-collection cycle. So, if you delete a container and immediately try to create another container with the same name, you may receive an error if the container hasn't been garbage-collected. When a container is deleted, all the blobs in that container are also deleted.
List Blobs	Retrieves blobs from a particular container. Similar to the List Containers operation, you can specify maxresults and prefix parameters to filter your results. This operation also supports a delimiter parameter that you can use to group blobs in a virtual path structure. For example, if there are two blobs named mydocuments/docA.docx and mydocuments/docB.docx, then if you specify the delimiter as / in your HTTP Request, the HTTP response will contain a <BlobPrefix>mydocuments/</BlobPrefix> element as a virtual group for docA.docx and docB.docx.

Table 3-8 lists some of the important characteristics of the container operations listed in Table 3-7.

Table 3-8. *Container Operation Characteristics*

Operation	HTTP Verb	Cloud URI	Development Storage URI	HTTP Version	Permissions
Create Container	PUT	<code>http://<account name>.blob.core.windows.net/<container name></code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName></code>	HTTP/1.1	Only the account owner can call this operation.
Get Container Properties	GET/HEAD	<code>http://<account name>.blob.core.windows.net/<container name></code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName></code>	HTTP/1.1	Any client may call this operation on a public container.
Set Container Metadata	PUT	<code>http://<account name>.blob.core.windows.net/<container name>?comp=metadata</code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName>?comp=metadata</code>	HTTP/1.1	Only the account owner can call this operation.
Get Container ACL	GET/HEAD	<code>http://<account name>.blob.core.windows.net/<container name>?comp=acl</code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName>?comp=acl</code>	HTTP/1.1	Only the account owner can call this operation.
Set Container ACL	PUT	<code>http://<account name>.blob.core.windows.net/<container name>?comp=acl</code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName>?comp=acl</code>	HTTP/1.1	Only the account owner can call this operation.
Delete Container	DELETE	<code>http://<account name>.blob.core.windows.net/<container name></code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName></code>	HTTP/1.1	Only the account owner can call this operation.
List Blobs	GET	<code>http://<account name>.blob.core.windows.net/<container name>?comp=list</code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName>?comp=list</code>	HTTP/1.1	Only the account owner can call this operation.

`<account name>` is the storage account name in the cloud, and `<devstorageaccount>` is the development storage account. note that only one operation, Get Container Properties, can be called by all the users on a public container. All other operations can only be called by the owner of the container.

The following sections discuss some of the operations from Table 3-8 in detail. Even though the operations are different, the programming concepts behind them are similar. To keep the book at a

conceptual level, I discuss Create Container, Set Container Metadata, and List Blobs operations, because they cover most of the discussed concepts. By studying these three operations in detail, you can understand the programming concepts behind all the container operations. The Windows Azure Storage Operations.exe application included with this chapter's source code contains implementations of these container operations.

Create Container

The Create Container operation is used to create a container in an account. The URI for Create Container is of the format `http://<account name>.blob.core.windows.net/<container name>`. You can think of a container as a bucket for holding similar blobs, although it's not a requirement that blobs in a container be similar. For example, if you want to store all your media files as Azure blobs, you can create a container for each media type, such as Music, Video, and Pictures. Then, you can store your media blobs under each category. This gives you easy access to particular media types. The Create Container REST request looks like Listing 3-9.

Listing 3-9. Create Container REST Request

```
PUT /myfirstcontainer?timeout=30 HTTP/1.1
x-ms-date: Fri, 05 Jun 2009 02:31:10 GMT
x-ms-meta-creator: tejaswi
x-ms-meta-creation-date: 06042009
x-ms-prop-publicaccess: true
Authorization: SharedKey proazurestorage:mQfgLwFfzFdDdMU+drg5sY2LfGKMSfXQnWrxrLPtzBU=
Host: proazurestorage.blob.core.windows.net
Content-Length: 0
Connection: Keep-Alive
```

Listing 3-9 shows the request to create a container named `myfirstcontainer`. `x-ms-meta-[name]:[value]` represents the metadata values for the container. `x-ms-prop-publicaccess:true` indicates that the container has public visibility.

For the Create Container operation, the Blob service responds with a status code of HTTP/1.1 201 Created or HTTP/1.1 409 Conflict if a container with the same name already exists. The Create Container response is shown in Listing 3-10.

Listing 3-10. Create Container REST Response

```
HTTP/1.1 201 Created
Last-Modified: Fri, 05 Jun 2009 02:32:43 GMT
ETag: 0x8CBB39D0A486280
Server: Blob Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: a0ea17df-5528-4ad3-985c-20664b425c7b
Date: Fri, 05 Jun 2009 02:32:43 GMT
Content-Length: 0
```

In Listing 3-10, the first line represents the status code of the operation. The ETag and the Last-Modified values can be used in conditional headers while modifying or deleting the container. The Create Container operation doesn't support any conditional headers, but the Set Container Metadata and Delete Container operations, discussed later, do support conditional headers. `x-ms-request-id` represents a unique request identifier that you can use for debugging or tracing.

Figure 3-9 shows the working of the Create Container operation in the Windows Azure Storage Operations application.

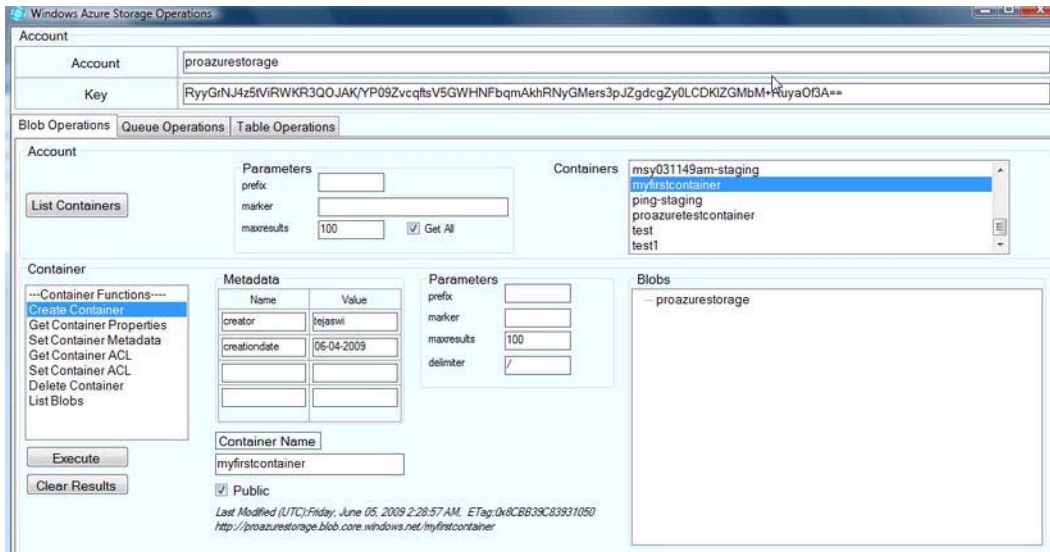


Figure 3-9. Create Container from the Windows Azure Storage Operations application

As shown in Figure 3-9, you follow these steps to create a container:

1. Enter a container name (such as myfirstcontainer) in the Container Name text field.
2. Check the Public check box if the container is public (accessible to everyone).
3. Select Create Container Function in the Container Functions list box.
4. Click the Execute button.

After the container is created, the Containers list box in the Account section is refreshed and displays the newly created container's name. To better understand the programming model of the Create Container operation, open the Visual Studio Solution Chapter3.sln from the Chapter 3 source directory. The WASTorageHelper class in ProAzureCommonLib contains helper functions for creating containers, as shown in Listing 3-11.

Listing 3-11. CreateContainer Method in the WASTorageHelper Class

```
public IList<CloudBlobContainer> GetContainers(string prefix)
{
    return BlobClient.ListContainers(prefix,
ContainerListingDetails.All).ToList<CloudBlobContainer>();
}

public bool CreateContainer(string containerName)
{
    CloudBlobContainer container = GetBlobContainer(containerName);
```

```

        return container.CreateIfNotExist();
    }

    public bool CreateContainer(string containerName, BlobContainerPermissions
permissions, NameValueCollection metadata)
    {
        CloudBlobContainer container = GetBlobContainer(containerName);
        bool result = container.CreateIfNotExist();
        if (result)
        {
            container.SetPermissions(permissions);
            container.Metadata.Add(metadata);
            container.SetMetadata();
        }
        return result;
    }
}

```

The first `CreateContainer()` method creates a container with default properties (private access). The second method accepts a permission object and metadata for the container. If you observe, the function abstracts the calls to `SetPermissions()` and `SetMetadata()` within a single call.

Set Container Metadata

Containers contain name-value pairs of metadata values. You can store values like time of creation, creator, last modified by user, and so on in a container's metadata fields. The size of the metadata can be 8KB per container. The Set Container Metadata operation sets the metadata of a container independently. The URI for the Set Container Metadata operation is of the format `http://<account name>.blob.core.windows.net/<container name>?comp=metadata`. The Set Container Metadata REST request looks like Listing 3-12.

Listing 3-12. Set Container Metadata REST Request

```

PUT /myfirstcontainer?comp=metadata&timeout=30 HTTP/1.1
x-ms-date: Fri, 05 Jun 2009 05:44:21 GMT
x-ms-meta-creator: tejaswi
x-ms-meta-creation-date: 06042009
x-ms-meta-last-updated-by: arohi
Authorization: SharedKey proazurestorage:hC5t3Qsc09kIN0zRCRN2vcgTIyPR97ay7WZRzwbKBI=
Host: proazurestorage.blob.core.windows.net
Content-Length: 0
Connection: Keep-Alive

```

In Listing 3-12, the HTTP verb used is PUT. Note the URI parameter `?comp=metadata`; it instructs the Blob service to set the container metadata instead of creating the container. The Create Container operation doesn't have this parameter. The `x-ms-meta.[name]:[value]` entries represent the metadata name-value pairs you want to set on the container.

■ **Caution** The Set Container Metadata operation replaces all the existing metadata of the container. It doesn't update individual metadata entries. For example, if a container has two metadata values Creator and Creation-Time, and you call Set Container Metadata with only one metadata value LastUpdatedBy, then the Creator and Creation-Time values will be deleted and the container will have only one metadata value LastUpdatedBy. To avoid this side effect, always set all the metadata values again along with any new values you want to add to the container's metadata.

The Set Container Metadata operation also supports the conditional header If-Modified-Since, which isn't shown in Listing 3-12. The If-Modified-Since header carries a date-time value instructing the Blob service to set the metadata values only if they have been modified since the supplied date in the request header.

The response from the Blob service consists of one the following HTTP status codes:

- HTTP/1.1 200 OK if the operation is successful
- HTTP/1.1 412 PreconditionFailed if the precondition If-Modified-Since fails
- HTTP/1.1 304 NotModified if the condition specified in the header isn't met

Figure 3-10 illustrates the execution of the Set Container Metadata operation in the Windows Azure Storage Operations application.

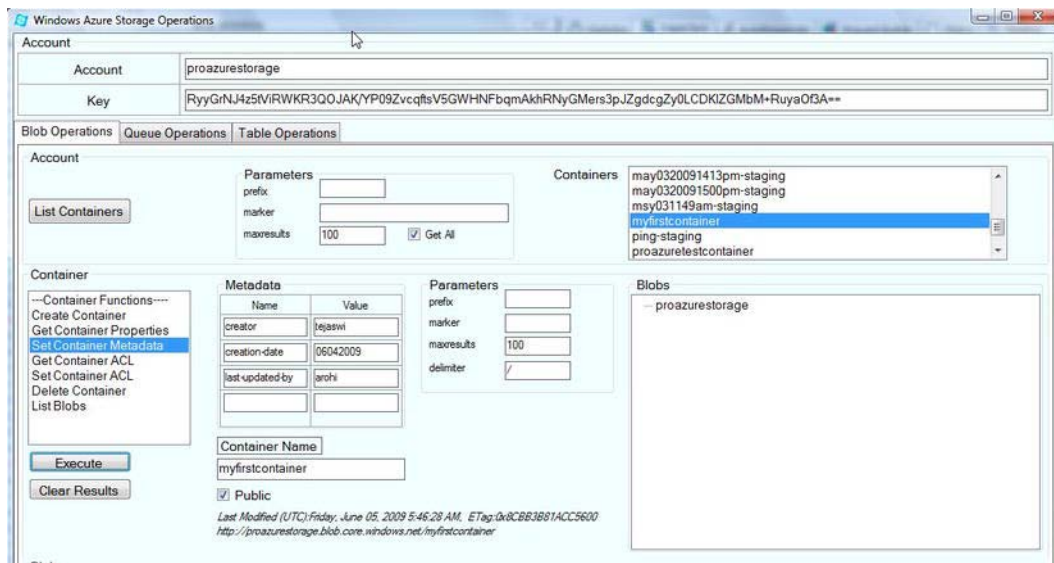


Figure 3-10. Set Container Metadata in the Windows Azure Storage Operations application

As shown in Figure 3-10, please follow these steps to execute the Set Container Metadata operation:

1. In the Account section, click the List Containers button to get a list of containers in your account.
2. Select one of the containers from the list (such as myfirstcontainer).
3. Make sure the Container Name text box in the Container section displays the name of the selected container.
4. In the Container section, select the Set Container Metadata operation from the list of container operations.
5. In the Containers section, enter metadata name-value pairs in the Metadata section.
6. Click the Execute button to execute the operation.
7. To verify the success of the operation, click the Clear Results button in the Containers section, and re-select the container from the Containers list in the Account section to the newly set metadata values.

To better understand the programming model of the Set Container Metadata operation, open the Visual Studio Solution Chapter3.sln from the Chapter 3 source directory. The WASTorageHelper class in ProAzureCommonLib contains a helper function called SetContainerMetadata(), as shown in Listing 3-13.

Listing 3-13. *SetContainerMetadata Method in the WASTorageHelper Class*

```
public void SetContainerMetadata(string containerName, NameValueCollection metadata)
{
    CloudBlobContainer container = GetBlobContainer(containerName);
    container.CreateIfNotExist();
    container.Metadata.Clear();
    container.Metadata.Add(metadata);
    container.SetMetadata();
}
```

In Listing 3-13, the SetContainerMetadata method accepts the container name and a System.Collection.Specialized.NameValueCollection object populated with metadata name-value pairs. The container name is used to create a local instance of the CloudBlobContainer object. The code then calls the SetContainerMetadata() method on the CloudBlobContainer object to set the metadata values for the container. If the metadata is set successfully, a Boolean value of true is returned to the caller; otherwise, a false value is returned.

List Blobs

Containers are typically used to logically group and store blobs. The URI for the List Blobs operation is of the format `http://<account name>.blob.core.windows.net/<container name>?comp=list`. At the container level of the Blob service hierarchy, you can get a list of blobs in a container by calling the List Blobs operation. The List Blobs operation also provides paging capabilities with maxresults and prefix parameters, similar to the List Containers operation discussed earlier. The container and blob hierarchy is a single-level hierarchy, but in real-world applications you want to create deeper hierarchies with folder structures to store blob files. For example, there may be a scenario where you would want to create a multilevel folder structure for your music files in the music container, as shown in Figure 3-11.

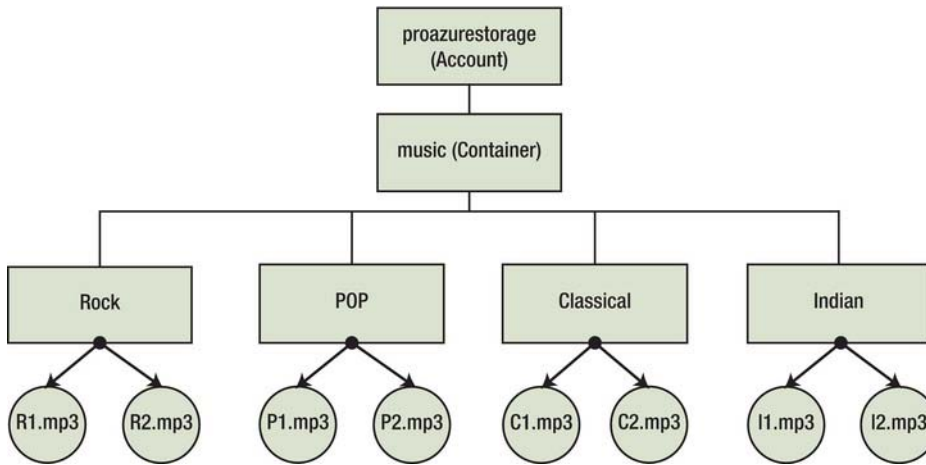


Figure 3-11. Music container hierarchy

The *.mp3 files represent the blob files you want to store in the music container, but you want an intermediate folder structure to organize the blob files by the genre of the music file. The Blob service hierarchy doesn't allow you to create folders in containers to create folder structures, but the blob naming convention is relaxed so that you can add a delimiter in the name of the blob file to create a virtual folder structure within the name of the blobs. To create a virtual folder structure as shown in Figure 3-11, you can name the blobs as follows:

Rock/R1.mp3
 Rock/R2.mp3
 POP/P1.mp3
 POP/P2.mp3
 Classical/C1.mp3
 Classical/C2.mp3
 Indian/I1.mp3
 Indian/I2.mp3

When you store a blob to the container, you specify the folder structure in the file name. When you retrieve the blob structure, you specify the delimiter character (/ in this example) as the parameter in the URI of the List blob operation. The Blob service sends you a BlobPrefix XML element specifying the folder structure that groups blobs with similar names together. You see an example of this in a few paragraphs.

The URI for the List Blobs operation also supports additional optional parameters, as listed in Table 3-9.

Table 3-9. List Blobs URI Parameters

Parameter	Description	Example
Prefix	A filter parameter to return files starting with the specified prefix value.	<code>http://proazurestorage.blob.core.windows.net/music?comp=list&prefix=kishore</code> returns containers with names starting with the prefix “kishore.”
delimiter	A character or a string that represents the separation of different tokens present in the name of a blob.	If the name of a blob is <code>rock/R1.mp3</code> , you can specify <code>/</code> as a delimiter to separate the string tokens <code>rock</code> and <code>R1.mp3</code> .
marker	Pages blob results when all results weren’t returned by the Storage service either due to the default maximum results allowed (the current default is 5000) or because you specify the <code>maxresults</code> parameter in the URI. The marker prefix is opaque to the client application.	<code>http://proazurestorage.blob.core.windows.net/music/?comp=list&prefix=kishore&marker=/proazurestorage/kishore0320091132.mp3</code> .
maxresults	The maximum number of blobs the Blob service should return. The default value is 5000. The Server returns an HTTP Bad Request (400) code if you specify a <code>maxresults</code> value greater than 5000.	<code>http://proazurestorage.blob.core.windows.net/music/?comp=list&prefix=kishore&maxresults=100</code> .

Assume that the blob hierarchy from Figure 3-11 exists in the Blob service. To retrieve all the blobs in the music container, you have to execute two REST requests for each blob file. In the first request, you pass the delimiter (such as `/`) as one of the URI parameters. The Blob service response gets the first token of the blob name separated by the delimiter (for example, `Classical/`) as a `BlobPrefix` element in the response body. The first token doesn’t represent the end of the blob name, so you have to make one more request to the Blob service by passing the first token as a prefix parameter to the Blob service URI (for example, `prefix=Classical/`). Now, because the next token represents the end of the blob name, the Blob service sends the blob properties in the response. If the blob name has more tokens, you must keep on querying the Blob service until you reach the end of blob name to retrieve the blob properties you’re interested in.

■ **Note** If the blob name doesn’t contain a delimiter, or if you want to retrieve the blob name along with the delimiter, then you don’t have to pass a delimiter—the first response retrieves all the blob properties in the specified container.

In the first REST request, you specify the container name, maxresults (optional), and delimiter (/ in the music example), as shown in Listing 3-14.

Listing 3-14. List Blobs First REST request

```
GET /music?comp=list&delimiter=%2f&maxresults=100&timeout=30 HTTP/1.1
x-ms-date: Sun, 07 Jun 2009 05:53:37 GMT
Authorization: SharedKey proazurestorage:7euawYh5wNOGFJZGnvRn9vyR4y
Host: proazurestorage.blob.core.windows.net
```

The Blob service responds to this request with the list of <BlobPrefix> values tokenized by the delimiter at the next level of the folder hierarchy. In the music example, the next level of the folder hierarchy consists of Genres values like Classical, Indian, POP, and Rock. The response from the Blob service is shown in Listing 3-15.

Listing 3-15. List Blobs First Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Server: Blob Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 7c490b17-8c99-43fa-ab8b-bde4cef032d7
Date: Sun, 07 Jun 2009 05:54:41 GMT
Content-Length: 408

<?xml version="1.0" encoding="utf-8"?>
<EnumerationResults
ContainerName="http://proazurestorage.blob.core.windows.net/music">
  <MaxResults>100</MaxResults>
  <Delimiter>/</Delimiter>
  <Blobs>
    <BlobPrefix>
      <Name>Classical</Name>
    </BlobPrefix>
    <BlobPrefix>
      <Name>Indian</Name>
    </BlobPrefix>
    <BlobPrefix>
      <Name>POP</Name>
    </BlobPrefix>
    <BlobPrefix>
      <Name>Rock</Name>
    </BlobPrefix>
  </Blobs>
  <NextMarker />
</EnumerationResults>
```

Next, for each <BlobPrefix> value, you send a request to the Blob service to get the next string token separated by the delimiter at the next level of hierarchy. In this request, the prefix URI parameter must contain the <BlobPrefix> value, e.g. prefix=Classical, prefix=Indian, prefix=POP, or prefix=Rock. In the music hierarchy, the next token is the last token representing the file name of the music file. For example, under the Classical folder are C1.mp3 and C2.mp3 files. The response from the Blob service

contains the properties of the C1.mp3 and C2.mp3 files. The sample request for the List Blobs operation at the Genre folder structure level looks like Listing 3-16.

Listing 3-16. *List Blob Second REST Request*

```
GET /music?comp=list&
prefix=Classical%2f&delimiter=%2f&
maxresults=100&timeout=30 HTTP/1.1
x-ms-date: Sun, 07 Jun 2009 05:53:38 GMT
Authorization: SharedKey proazurestorage:E0V9XEPvs9J5zejM0HD+d3+3Lc2+B816HS9Vu2NwkaE=
Host: proazurestorage.blob.core.windows.net
```

In Listing 3-16, the prefix parameter is set to the <BlobPrefix> value sent in the response for the first REST request. Listing 3-17 represents the response from the Blob service to get the next level of hierarchy elements. The <Prefix> element contains the value of the prefix parameter passed in the URI.

Listing 3-17. *List Blob Second REST Response*

```
HTTP/1.1 200 OK
Content-Type: application/xml
Server: Blob Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 6ad95e46-652d-4e4c-a50b-68c14dd2bd74
Date: Sun, 07 Jun 2009 05:54:41 GMT
Content-Length: 863

<?xml version="1.0" encoding="utf-8"?>
<EnumerationResults ContainerName="http://proazurestorage.blob.core.windows.net/music">
  <Prefix>Classical/</Prefix>
  <MaxResults>100</MaxResults>
  <Delimiter></Delimiter>
  <Blobs>
    <Blob><Name>Classical/C1.mp3</Name>
    <Url>http://proazurestorage.blob.core.windows.net/music/Classical/C1.mp3</Url>
    <LastModified>Sun, 07 Jun 2009 05:47:18 GMT</LastModified>
    <Etag>0x8CBB54A8D83F750</Etag>
    <Size>4055168</Size>
    <ContentType>audio/mpeg</ContentType>
    <ContentEncoding /><ContentLanguage />
  </Blob>
  <Blob><Name>Classical/C2.mp3</Name>
  <Url>http://proazurestorage.blob.core.windows.net/music/Classical/C2.mp3</Url>
  <LastModified>Sun, 07 Jun 2009 05:47:38 GMT</LastModified>
  <Etag>0x8CBB54A99E42600</Etag>
  <Size>4055168</Size><ContentType>audio/mpeg</ContentType>
  <ContentEncoding />
  <ContentLanguage />
  </Blob>
</Blobs>
<NextMarker />
</EnumerationResults>
```

Repeat the same procedure for the other three genres—Indian, POP, and Rock—to get the blobs in those containers.

The Windows Azure Storage Operations application supports the retrieving of blobs with delimiters. In the Container section of the application, the TreeView control on the right side displays the results from the List Blobs operation. The Parameters group box contains text boxes for prefix, marker, maxresults, and delimiter. Figure 3-12 illustrates the List Blobs operation executed on a music container. Note the delimiter and the tree structure that are created in the TreeView control.

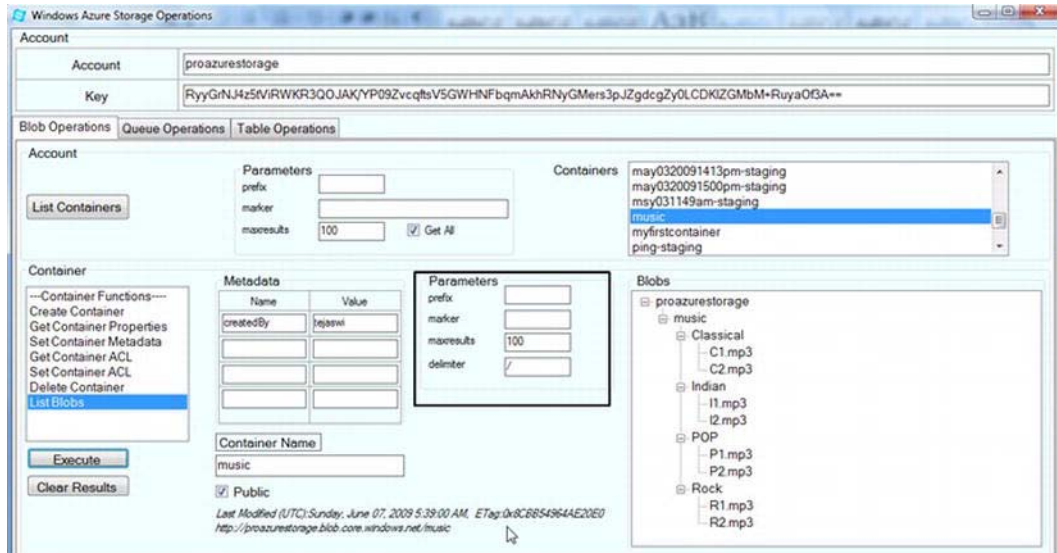


Figure 3-12. List Blobs in Windows Azure Storage Operations.exe

The List Blobs operation is called on the music container. The Delimiter text field contains the / delimiter character. The TreeView shows the virtual folder hierarchy of the blobs in the music container.

You can also use the CloudBlobDirectory class to get a reference to the logical directory in a container as shown in the Listing 3-18.

Listing 3-18. CloudBlobDirectory Usage

```
static void ListBlobs(string storageAccountStr, string containerName, string directoryName)
{
    CloudBlobClient blobClient = CloudStorageAccount.Parse(storageAccountStr);

    //Get a reference to a blob directory in the specified container.
    CloudBlobDirectory blobDir = blobClient.GetBlobDirectoryReference(string.Format("{0}/{1}",
        containerName, directoryName));

    //List blobs and directories.
    foreach (var blobItem in blobDir.ListBlobs())
    {

```

```
        Console.WriteLine(blobItem.Uri);
    }
    Console.WriteLine("-----");

    //using flat listing.
    BlobRequestOptions options = new BlobRequestOptions();
    options.UseFlatBlobListing = true;
    foreach (var blobItem in blobDir.ListBlobs(options))
    {
        Console.WriteLine(blobItem.Uri);
    }
}
```

Blob Operations

The URI of a specific blob is of the format `http://<account name>.blob.core.windows.net/<container name>/<blob name>`. Blobs support several operations, as listed in Table 3-10.

Table 3-10. Blob Operations

Operation	Description
Put Blob	Creates a new blob under the given container or updates an existing blob. Updates complete overwrite a blob’s contents and metadata. You can upload a blob up to 64MB in size using the Put Blob operation. If it’s bigger than 64MB, see the Put Block operation.
Get Blob	Retrieves the blob, its metadata, and its properties from the blob service. The operation times out if the download takes more than two minutes per megabyte.
Get Blob Properties	Retrieves the blob’s system properties, HTTP properties, and user-defined metadata.
Get Blob Metadata	Retrieves only the user-defined metadata of the specified Blob.
Set Blob Metadata	Sets the user-defined metadata of the specified blob.
Put Block (Block Blob)	Used to upload blobs larger than 64MB. Split the file into multiple blocks of 4MB each, and upload multiple blocks using this operation.
Get Block List (Block Blob)	Gets the list of blocks uploaded by the Put Block operation. The operation supports the listing of committed as well as uncommitted blocks.
Put Block List (Block Blob)	Commits a list of uploaded blocks to a blob. The operation accepts a list of block IDs of successfully uploaded blocks. Uncommitted blocks are garbage-collected.
Copy Blob	Copies a blob from a source to a destination within the Blob service.

Delete Blob	Marks a specified blob for deletion. The actual deletion takes place during the garbage-collection cycle.
Lease Blob	The Lease Blob method creates a one minute lock on the blob for write operations. This is helpful when you are writing concurrent applications supporting concurrent users. You can Acquire, Renew, Release or Break a lease. When you create a lease, the blob storage service returns you a leaseId which you can use for write operations on the blob.
Snapshot Blob	The Snapshot Blob method creates a readonly snapshot of the blob. This method is commonly used for backup and archiving scenarios. This method is also commonly used for taking snapshot of the Windows Azure drives and then attaching these drives to multiple role instances for readonly data access.
Put Page (Page Blob)	The Put Page operation writes page ranges to the page blob. A Page Blob must exist before you write pages to it. You can use Update and Clear options in the Put Page method to update and clear the specified page ranges.
Get Page Regions (Page Blob)	The Get Page Regions method gets the page regions for the specified blob or snapshot of a blob. You can optionally specify the regions you want returned or the blob service will return all the page regions.

Table 3-11 lists some of the important characteristics of the blob operations. Table 3-12. Blob Operation Characteristics

Operation	HTTP Verb	Cloud URI	Development Storage URI	HTTP Version	Permissions
Put Blob	PUT	<code>http://<account name>.blob.core.windows.net/<container name>/<blob name></code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name></code>	HTTP/1.1	Only the account owner can call this operation.
Get Blob	GET	<code>http://<account name>.blob.core.windows.net/<container name>/<blob name></code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name></code>	HTTP/1.1	Any client may call this operation on a blob in the public container.
Get Blob Properties	HEAD	<code>http://<account name>.blob.core.windows.net/<container name>/<blob name></code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name></code>	HTTP/1.1	Any client may call this operation on a blob in the public container.
Get Blob	GET/	<code>http://<account name>.blob.core.wind</code>	<code>http://127.0.0.1:10000/<devstorageac</code>	HTTP/1.1	Only the account owner can call this

Metadata	HEAD	ows.net/<container name>/<blob name>?comp=metadata	count>/<containerName>/<blob name>?comp=metadata		operation.
Set Blob Metadata	PUT	http://<account name>.blob.core.windows.net/<container name>/<blob name>?comp=metadata	http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name>?comp=metadata	HTTP/1.1	Only the account owner can call this operation.
Put Block (Block Blob)	PUT	http://<account name>.blob.core.windows.net/<container name>/<blob name>?comp=block&blockid=id	http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name>?comp=block&blockid=id	HTTP/1.1	Only the account owner can call this operation.
Get Block List (Block Blob)	GET	http://<account name>.blob.core.windows.net/<container name>/<blob name>?comp=blocklist&blocklisttype=[committed uncommitted all]	http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name>?comp=blocklist&blocklisttype=[committed uncommitted all]	HTTP/1.1	Any client may call this operation on a blob in the public container.
Put Block List (Block Blob)	PUT	http://<account name>.blob.core.windows.net/<container name>/<blob name>?comp=blocklist	http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name>?comp=blocklist	HTTP/1.1	Only the account owner can call this operation.
Copy Blob	PUT	http://<account name>.blob.core.windows.net/<container name>/<blob name>	http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name>	HTTP/1.1	Only the account owner can call this operation.
Delete Blob	DELETE	http://<account name>.blob.core.windows.net/<container name>/<blob name>	http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name>	HTTP/1.1	Only the account owner can call this operation.
Lease Blob	PUT	http://<account name>.blob.core.windows.net/<container name>/<blob name>?comp=lease	http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name>?comp=lease	HTTP/1.1	Only the account owner can call this operation.

Snapshot Blob	PUT	<code>http://<account name>.blob.core.windows.net/<container name>/<blob name>?comp=snapshot</code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name>?comp=snapshot</code>	HTTP/1.1	Only the account owner can call this operation.
Put Page (Page Blob)		<code>http://<account name>.blob.core.windows.net/<container name>/<blob name>?comp=page</code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name>?comp=page</code>	HTTP/1.1	Only the account owner can call this operation.
Get Page Regions (Page Blob)		<code>http://<account name>.blob.core.windows.net/<container name>/<blob name>?comp=pagelist</code> <code>http://<account name>.blob.core.windows.net/<container name>/<blob name>?comp=pagelist&snapshot=<DateTime></code>	<code>http://127.0.0.1:10000/<devstorageaccount>/<containerName>/<blob name>?comp=pagelist</code>	HTTP/1.1	Any client may call this operation on a blob in the public container.

<account name> is the storage account name in the cloud, and *<devstorageaccount>* is the development storage account. *<container name>* is the name of the container in which the blob is stored, and *<blob name>* is the name of the blob object.

The following sections discuss some of the operations from Table 3-11 in detail. Even though the operations are different, the programming concepts behind them are similar. To keep the book at a conceptual level, I discuss the Put Blob, Get Blob, and Copy Blob operations because they cover most of the discussed concepts. By studying these three operations in detail, you will understand the programming concepts behind all the blob operations. The Windows Azure Storage Operations application included with this chapter's source code contains implementations of most of the blob operations.

Put Blob

The Put Blob operation is used to upload blob objects to the Blob service. A blob must be stored in a container, so the URI is of the format `http://<account name>.blob.core.windows.net/<container name>/<blob name>`, where the *<container name>* must be referenced before a *<blob name>*. You can upload a blob file up to 64MB using a single Put Blob operation. The Put Blob REST request looks like Listing 3-19.

Listing 3-19. Put Blob REST Request

```

PUT /pictures/toucan.jpg?timeout=30 HTTP/1.1
x-ms-date: Wed, 10 Jun 2009 05:32:42 GMT
Content-Type: image/jpeg
If-None-Match: *
Authorization: SharedKey proazurestorage:GvjnS02oBj8nS1Fjh0DonOwDhvG6ak32VlPHZNp6qc8=
Host: proazurestorage.blob.core.windows.net
Content-Length: 33624
Expect: 100-continue

```

In Listing 3-19, a toucan.jpg file is uploaded to the pictures container in the Blob service. Note that the conditional header If-None-Match has a * value associated with it. This conditional header instructs the Blob service to upload the file only if the ETag value of the destination is different to the ETag value of the source. Because this file is a fresh upload, the conditional header doesn't matter. Also note that the Content-Length of the HTTP request body is only 33,624 bytes. Because this is less than 64MB, a single Put Blob operation can upload this file to the Blob service. Listing 3-20 shows the response from the Blob service.

Listing 3-20. Put Blob REST Response

```

HTTP/1.1 201 Created
Content-MD5: df6MtpHeFTI4oChTKxil1A==
Last-Modified: Wed, 10 Jun 2009 05:34:43 GMT
ETag: 0x8CBB7A44AEB70B0
Server: Blob Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 7a898dd6-4458-439e-8895-003584810d7c
Date: Wed, 10 Jun 2009 05:34:19 GMT
Content-Length: 0

```

The Blob service responds with an HTTP/1.1 201 Created status code for a successful blob upload. Figure 3-13 shows the working of the Put Blob operation in the Windows Azure Storage Operations application.

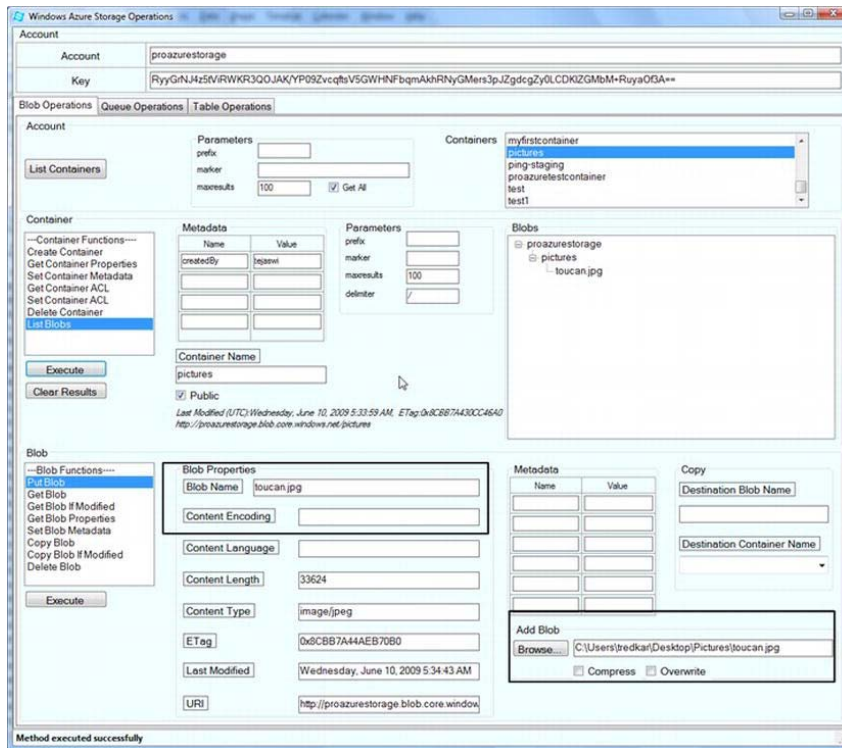


Figure 3-13. Put Blob in Windows Azure Storage Operations.exe

As illustrated in Figure 3-13, you can upload a blob to the Blob service from the Windows Azure Storage Operations application. The steps to upload are as follows:

1. Create a new container (called pictures).
2. Select the new container from the list box.
3. In the Blob section on the right side, under Add Blob, select an image from your file system.
4. If you wish, rename the file in the Blob Name text field.
5. You can also create a virtual folder structure (such as pictures/toucan.jpg) in the path name in the Blob Name text field.
6. Select the Put Blob function from the Blob functions list box.
7. Execute the function to upload the blob to the pictures container.
8. Execute the List Blobs function to refresh the blobs list and display the newly added blob.

To help you understand the programming model of the Put Blob operation, open the Visual Studio Solution Chapter3Solution.sln from the Chapter 3 source directory. The WASTorageHelper.cs class in the Windows Azure Storage Operations project consists of several overloaded PutBlob() methods, as shown in Listing 3-21.

Listing 3-21. *PutBlob() Method in WindowsAzureStorage.cs*

```
public void PutBlob(CloudBlob blob, Byte[] contents)
{
    //Upload byte array
    blob.UploadByteArray(contents);
}

public void PutBlobFromFile(CloudBlob blob, string fileName)
{
    //Upload file
    blob.Properties.ContentType =
WASTorageHelper.GetContentTypeFromExtension(Path.GetExtension(fileName));
    blob.UploadFile(fileName);
}

public void PutBlob(CloudBlob blob, Stream contents)
{
    //Upload Stream
    blob.UploadFromStream(contents);
}

public void PutBlob(CloudBlob blob, string contents)
{
    //Upload Text
    blob.UploadText(contents);
}
```

Each PutBlob() method represents the type of datasource used for uploading contents to the blob. You can upload a byte array, stream, file from a local file system and even text content directly to a blob.

Put Block and Put Block List Operations

To upload files larger than 64MB, break the blob into smaller contiguous files (Blocks) that are maximum of 4MB each, and then upload these blocks using the Put Block operation. You can commit uploaded blocks to a blob using the Put Block List operation. Before uploading a block, you have to assign a blockid that is unique within a blob. Blocks in different blobs can have the same blockid because blockids are unique only within the scope of a blob. You can upload blocks in parallel or in any order, as long as the Put Block List operation commits all the blocks to a blob in the correct contiguous order. An uploaded block doesn't become part of a blob unless it's committed using the Put Block operation. Uncommitted blocks are stored for seven days before they're garbage-collected by the system.

■ **Note** The `UploadFromStream()` method uses the Put Block method from the REST API. The `CloudBlobClient.WriteBlockSizeInBytes` property determines each block size that will be uploaded to the blob storage. If you uploading larger files (maximum Block blob size is 200GB), you can set `CloudBlobClient.WriteBlockSizeInBytes = 4 MB` which is the maximum size per block. Behind the scenes, the `UploadFromStream()` method will upload the file in multiple blocks of 4MB.

Ideally, from an end-user perspective, the Put Block operation should be transparent. The application should upload the blob as contiguous blocks and commit the Blocklist transparently for the end user. The end user should only be given the status of the blob upload. The `StorageClient` API abstracts the Put Block operation from the end user but you can use the REST API directly for uploading blocks and then committing the blocks to a blob. To test the Put Block and Put Block List operations, upload an image file larger than 64MB from the Windows Azure Storage Operations application. Listing 3-22 shows four separate REST requests of the Put Block operations to upload a music file `R3.mp3` to the music container.

Listing 3-22. REST Requests for Put Block Operations

PUT /music/Rock/R3.mp3?comp=block&blockid=AAAAA%3d%3d&timeout=30 HTTP/1.1

```
x-ms-date: Thu, 11 Jun 2009 04:12:14 GMT
Content-Type: audio/mpeg
If-None-Match: *
Authorization: SharedKey proazurestorage:UE3slooBGXZewrAHTXj7efzdA33ozPoElVs/5NWNoy8=
Host: proazurestorage.blob.core.windows.net
Content-Length: 1048576
Expect: 100-continue
```

PUT /music/Rock/R3.mp3?comp=block&blockid=AQAAAA%3d%3d&timeout=30 HTTP/1.1

```
x-ms-date: Thu, 11 Jun 2009 04:12:17 GMT
Content-Type: audio/mpeg
If-None-Match: *
Authorization: SharedKey proazurestorage:BOHjZPkvSN1IZWNJ7VGhrOppe7DAXSvjKv516xgGOWQ=
Host: proazurestorage.blob.core.windows.net
Content-Length: 1048576
Expect: 100-continue
```

PUT /music/Rock/R3.mp3?comp=block&blockid=AgAAAA%3d%3d&timeout=30 HTTP/1.1

```
x-ms-date: Thu, 11 Jun 2009 04:12:18 GMT
Content-Type: audio/mpeg
If-None-Match: *
Authorization: SharedKey proazurestorage:Fo+V+kdv6cEbBsOCLIMIdQ+lZLfHX7Dit8lqAEkwqeI=
Host: proazurestorage.blob.core.windows.net
Content-Length: 1048576
Expect: 100-continue
```

PUT /music/Rock/R3.mp3?comp=block&blockid=AwAAAA%3d%3d&timeout=30 HTTP/1.1

```
x-ms-date: Thu, 11 Jun 2009 04:12:20 GMT
Content-Type: audio/mpeg
If-None-Match: *
Authorization: SharedKey proazurestorage:uwPvdQyf6RMZ0i6fYtVmz1RRZxXu3L1SLvXoTGLpCY8=
Host: proazurestorage.blob.core.windows.net
Content-Length: 909440
Expect: 100-continue
```

As shown in Listing 3-22, the blob is uploaded in four contiguous blocks. Note the unique blockid of each REST request.

After uploading blocks to the Blob service, you need to commit them to the blob using the Put Block List operation. Listing 3-23 shows the REST request for the Put Block List operation to commit the uploaded blocks to the Rock/R3.mp3 blob.

Listing 3-23. REST Request for the Put Block Operation

```
PUT /music/Rock/R3.mp3?comp=blocklist&timeout=30 HTTP/1.1
x-ms-date: Thu, 11 Jun 2009 04:12:21 GMT
Content-Type: audio/mpeg
If-None-Match: *
Authorization: SharedKey proazurestorage:0X01XUegzNFcyuwBicRSoon/CgB8jA0wrEQaMDFGGlk=
Host: proazurestorage.blob.core.windows.net
Content-Length: 156
Expect: 100-continue
```

```
<?xml version="1.0" encoding="utf-8"?>
<BlockList>
<Block>AAAAAA==</Block>
<Block>AQAAAA==</Block>
<Block>AgAAAA==</Block>
<Block>AwAAAA==</Block>
</BlockList>
```

The blob Rock/R3.mp3 is created when the four blocks in Listing 3-22 are committed by the Put Block List operation in Listing 3-23. The name of the blob is part of the operation URI, even though the blob doesn't exist before Put Block List is executed. The blob is created only after the Put Block List operation is successfully executed.

Get Blob

The Get Blob operation is used to download the blob contents, its properties, and metadata from the Blob service. The URI for the Get Blob operation is of the format `http://<account name>.blob.core.windows.net/<container name>/<blob name>`. Listing 3-24 shows the REST API request for the Get Blob operation.

Listing 3-24. Get Blob REST Request

```
GET /pictures/birds/toucan.jpg?timeout=30 HTTP/1.1
x-ms-date: Thu, 11 Jun 2009 05:14:10 GMT
If-Match: 0x8CBB8550DF72BC0
```

```
x-ms-range: bytes=0-51086
Authorization: SharedKey proazurestorage:EVXgpmvaiEtyJlmBgupxLi2VebXK4XQk6/HsPF903EI=
Host: proazurestorage.blob.core.windows.net
```

In Listing 3-24, the URI points to the blob `birds/toucan.jpg`. The `If-Match` conditional header instructs the Blob service to check the specified ETag before downloading the blob. The `x-ms-range` value represents the range of bytes to be retrieved. This value is usually transparent to the end user; you can use it to download the blobs in batches of bytes. Listing 3-25 shows the REST API response from the Blob service for the Get Blob operation.

Listing 3-25. Get Blob REST Response

```
HTTP/1.1 206 Partial Content
Content-Length: 33624
Content-Type: image/jpeg
Content-Range: bytes 0-51086/51087
Last-Modified: Thu, 11 Jun 2009 02:40:02 GMT
ETag: 0x8CBB8550DF72BC0
Server: Blob Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 374e2072-106d-4841-b51c-45f25e9e6596
x-ms-meta-createdBy: tejaswi
Date: Thu, 11 Jun 2009 05:15:21 GMT
```

Listing 3-25 shows the HTTP header of the Get Blob operation. The HTTP response body consists of the contents of the blob. Figure 3-14 shows the working of the Get Blob operation in the Windows Azure Storage Operations application.

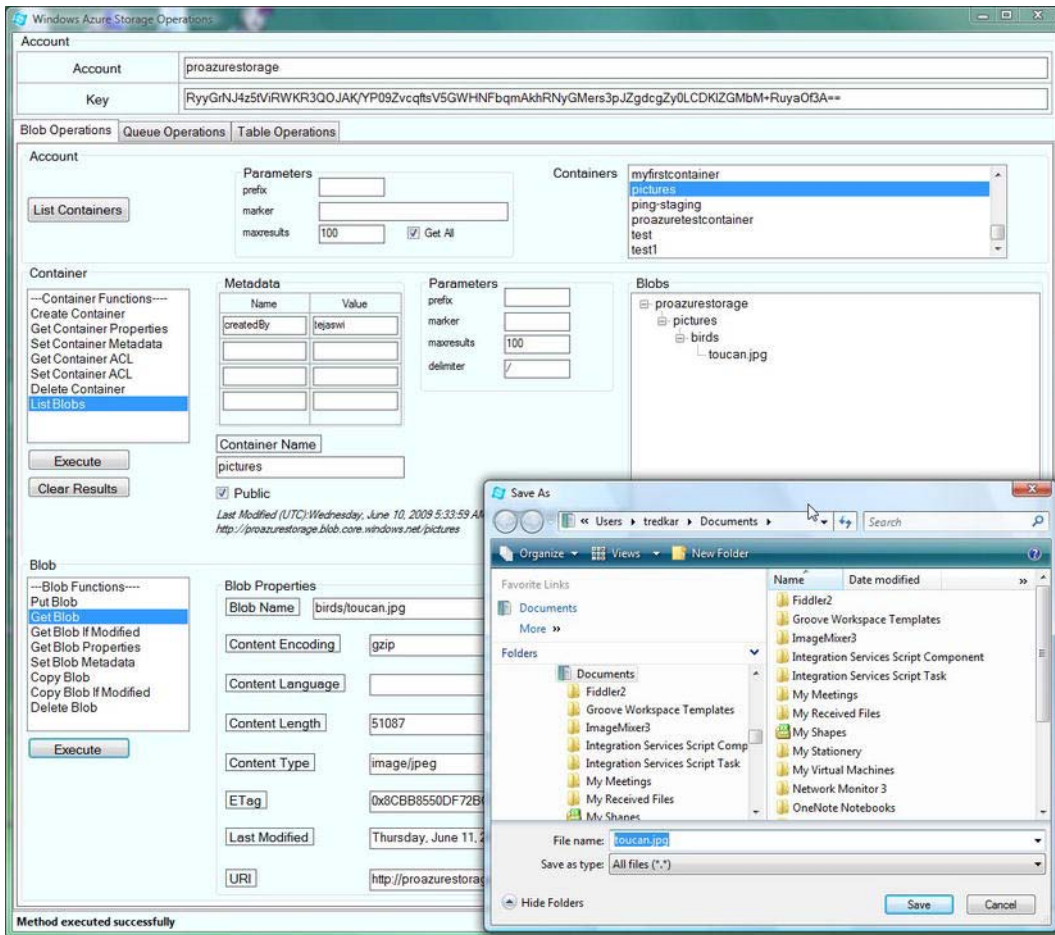


Figure 3-14. Get Blob in Windows Azure Storage Operations.exe

As illustrated in Figure 3-14, you can download a blob from the blob service using Windows Azure Storage Operations. The steps for downloading are as follows:

1. In the Containers list box, select a container (such as pictures) that has blobs.
2. In the Containers section, execute the List Blobs operation to get a list of blobs in the container (for example, birds/toucan.jpg).
3. Select a blob (such as birds/toucan.jpg) from the Blobs TreeView control.
4. In the Blobs section, execute the Get Blob operation.
5. A Save As dialog box pops up, where you can choose the local folder in which to store the blob.

6. When you click Save, the blob is stored on your local machine in the specified folder.

To help you understand the programming model of the Get Blob operation, open the Visual Studio Solution Chapter3Solution.sln from the Chapter 3 source directory. The WASTorageHelper.cs file in the Windows Azure Storage Operations project consists of several overloads of the `GetBlob()` method, some are shown in Listing 3-26.

Listing 3-26. *GetBlob() Method in WindowsAzureStorage.cs*

```
public void GetBlobContentsAsFileIfModified(string containerName, string blobName, string
fileName)
{
    CloudBlob blob = GetBlob(containerName, blobName);
    BlobRequestOptions options = CreateIfModifiedOption(blob);
    blob.FetchAttributes();
    try
    {
        blob.DownloadToFile(fileName, options);
    }
    catch (StorageClientException ex)
    {
        if (ex.ErrorCode == StorageErrorCode.BadRequest)
            throw new InvalidOperationException(string.Format("{0} was not downloaded,
since the blob has not been modified.", blobName));
        else
            throw ex;
    }
}

public void GetBlobContentsAsStream(string containerName, string blobName, Stream
stream)
{
    CloudBlob blob = GetBlob(containerName, blobName);
    blob.DownloadToStream(stream);
}
```

The `GetBlobContentsAsFileIfModified()` method downloads the contents of the blob to a file on the local file system only if the contents of the blob are modified in the Blob storage. The `GetBlobContentsAsStream()` downloads the contents of the blob to a stream you can use for further processing.

■ **Note** The Blob service supports blob concurrency on Get and Put operations via snapshot isolation. A Get Blob operation sees only a single version of the blob. If the blob is updated during the Get Blob operation, you receive a “connection closed” error. You can then follow up the error with an If-Modified conditional Get Blob operation.

Copy Blob

The Copy Blob operation is used to copy a source blob and its properties and metadata to a destination blob within a storage account. If you don't specify metadata values for the destination blob, then the source blob metadata values will be copied by default. The URI for the Copy Blob operation is of the format `http://<account name>.blob.core.windows.net/<destination container name>/<destination blob name>`. Listing 3-27 shows the REST API request for the Copy Blob operation.

Listing 3-27. Copy Blob REST Request

PUT /test/birds/toucan-copy.jpg?timeout=30 HTTP/1.1

x-ms-date: Mon, 15 Jun 2009 15:49:56 GMT

x-ms-version: 2009-04-14

x-ms-meta-createdBy: tejaswi

x-ms-copy-source: /proazurestorage/pictures/birds/toucan.jpg

Authorization: SharedKey proazurestorage:FqssEZkcIUj1VrQhH0aLdt+rtEmvgjN0tu9XZ06iRKw=

Host: proazurestorage.blob.core.windows.net

Content-Length: 0

In Listing 3-27, the URI points to the destination blob `birds/toucan.jpg` in the `test` container. The `x-ms-version` value specifies the version of the Storage REST API to use. The Copy method wasn't available in the earlier CTP versions of the Storage REST API; you can use it beginning with version 2009-04-14. The `x-ms-copy-source` value specifies the source blob for the copy operation. Listing 3-28 shows the REST API response from the Blob service for the Copy Blob operation.

Listing 3-28. Copy Blob REST Response

HTTP/1.1 201 Created

Last-Modified: Mon, 15 Jun 2009 15:52:27 GMT

ETag: 0x8CBBBE86B023C10

Server: Blob Service Version 1.0 Microsoft-HTTPAPI/2.0

x-ms-request-id: ee93e063-9256-443b-b6cb-536dd4012863

Date: Mon, 15 Jun 2009 15:51:28 GMT

Content-Length: 0

Listing 3-28 shows the HTTP header of the Copy Blob operation. The HTTP response body is similar to the Put Blob operation response body you saw earlier in the chapter. Figure 3-15 shows the Copy Blob operation in the Windows Azure Storage Operations.exe application.

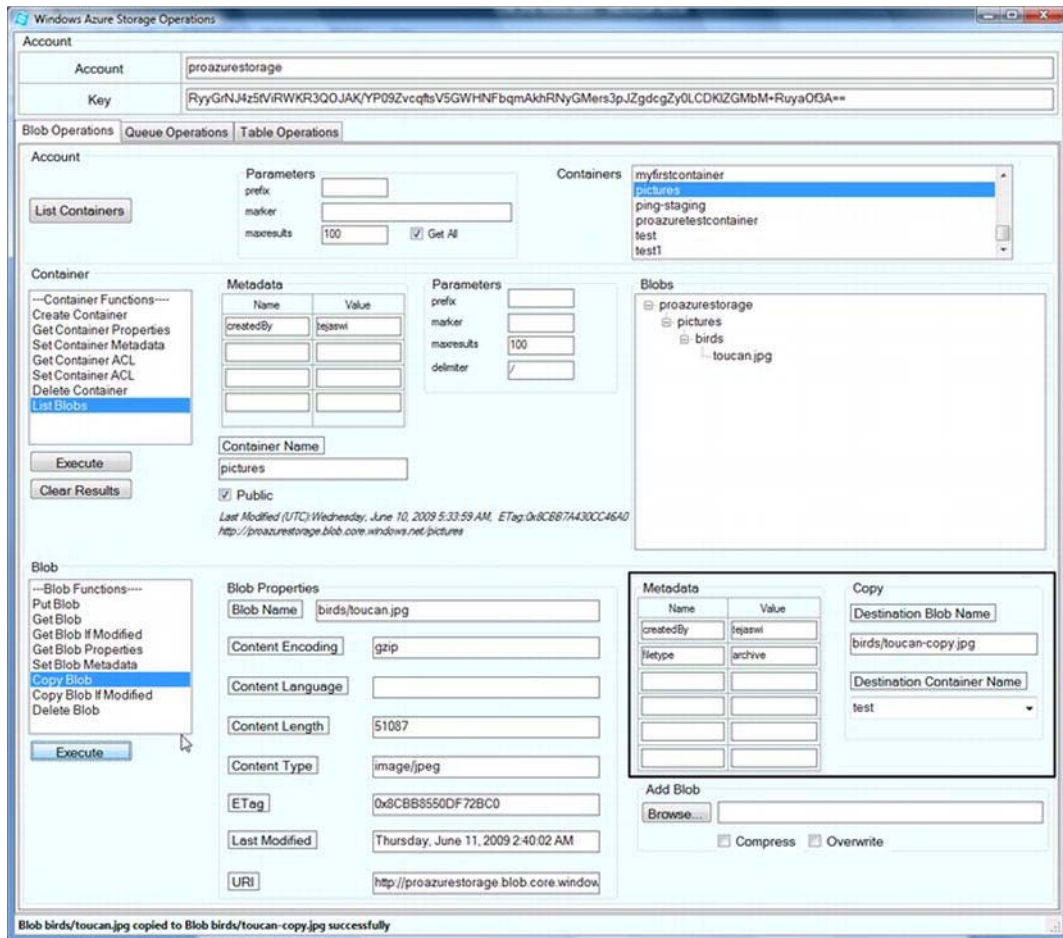


Figure 3-15. Copy Blob in Windows Azure Storage Operations.exe

As illustrated in Figure 3-15, you can copy a blob from a source blob to a destination blob within the same storage account from Windows Azure Storage Operations. The steps for copying a blob are as follows:

1. In the Containers list box, select a container (such as pictures) that has blobs.
2. In the Containers section, execute the List Blobs operation to get a list of blobs in the container (for example, birds/toucan.jpg).
3. Select a blob (such as birds/toucan.jpg) from the Blobs TreeView control.
4. In the Blobs section, enter a name for the destination blob in the Destination Blob Name text box.

5. Also select a destination container (such as test) for the blob from the Destination Container Name drop-down list.
6. A Save As dialog box pops up where you can choose the local folder in which to store the blob.
7. Enter metadata, if any, in the Metadata text fields in the Blob section.
8. Select the Copy Blob operation in the Blob operations list box, and click the Execute button to execute the Copy Blob operation.
9. The status bar message indicates the success or failure of the operation.
10. You can execute the List Blobs operation on the destination container to see the copied blob.

To help you understand the programming model of the Copy Blob operation, open the Visual Studio Solution Chapter3Solution.sln from the Chapter 3 source directory. The WASTorageHelper.cs file in the Windows Azure Storage Operations project consists of several CopyBlob() overloaded methods. See Listing 3-29.

Listing 3-29. *CopyBlob() Method in WindowsAzureStorage.cs*

```
public CloudBlob CopyBlob(CloudBlob blob, string destinationContainerName, string
destinationBlobName)
{
    CloudBlobContainer copyContainer = GetBlobContainer(destinationContainerName);
    CloudBlob copyBlob = copyContainer.GetBlobReference(destinationBlobName);
    copyBlob.CopyFromBlob(blob);
    return copyBlob;
}

public CloudBlob CopyBlob(CloudBlob blob, string destinationContainerName, string
destinationBlobName, NameValueCollection additionalMetadata)
{
    //Get a reference to the destination container object
    CloudBlobContainer copyContainer = GetBlobContainer(destinationContainerName);
    CloudBlob copyBlob = copyContainer.GetBlobReference(destinationBlobName);
    copyBlob.CopyFromBlob(blob);
    // we have to do this for now, there is a bug in the SDK where the additional
metadata is not copied.
    copyBlob.Metadata.Add(additionalMetadata);
    copyBlob.SetMetadata();
    return copyBlob;
}
```

The first CopyBlob() method copies a blob from source to destination whereas the second method copies the blob and then sets the metadata of the copied blob.

Now that you have understood the Blob storage concepts in detail, let's look at how we can set use the CDN to cache blobs.

Content Delivery Network (CDN)

Content Delivery Network (CDN) is a caching service that caches your blobs and static content from compute instances at strategic locations closer to the blob consumers. For example, if your media-heavy web site has media files centrally located in the United States, whereas your users are from all the continents, then there will be performance degradation for the users in distant locations. Windows Azure CDN pushes content closer to the users at several data center locations in Asia, Australia, Europe, South America, and the United States. You can find the current list of CDN locations here: <http://msdn.microsoft.com/en-us/library/gg680302.aspx>. So, if you enable your media files on the Windows blob storage with CDN, they will be automatically available across these locations locally, thus improving the performance for the users. Currently, the only restriction on enabling CDN is the blob containers must be public. This makes CDN extremely useful for e-commerce, news media, social networking, and interactive media web sites.

When you enable a storage account with CDN, the portal creates a unique URL with the following format for CDN access to the blobs in that storage account: `http://<guid>.vo.msecnd.net/`.

This URL is different from the blob storage URL format, `http://<storageaccountname>.blob.core.windows.net/`, because, the blob storage URL is not designed to resolve to CDN locations. Therefore, to get the benefit of CDN, you must use the URL generated by CDN for the blob storage. You can also register a custom domain name for the CDN URL from Windows Azure Developer Portal.

To enable CDN on a storage account, follow these steps:

1. Go to your Windows Azure Developer Portal storage account.
2. Click on CDN on the left hand side menu.
3. On the top CDN menu, click on New Endpoint.
4. In the New Endpoint window, you can select a hosted service or a storage account for caching content.
5. Click OK to cache the content in CDN.
6. The portal provides a CDN endpoint to the storage by creating a CDN URL of the format `http://<guid>.vo.msecnd.net/`.

You can use the CDN endpoint URL for accessing your public containers. To create a custom domain name, you can click on the Add Domain button from the top menu and enter the domain name. The portal will instruct you to create a CNAME record pointing to `verify.azure.com`.

■ **Note** CDN content can be made available only for public containers and blobs over HTTP and HTTPS. Be careful while choosing content for caching. If you cache constantly changing data, you may not be able to reap the benefits of the CDN and will also cost you a lot. Also, you don't have control over the cache endpoints, means, based on the user access, the CDN decides which edge cache machine to cache your content on. This may have cost implications if you have worldwide user base.

You can also enable CDN on hosted services with the following constraints:

- CDN can only cache static content from the Windows Azure instances.
- The cloud service must be deployed to the production slot and not the staging slot.
- The cloud service must provide content on port 80 over the HTTP protocol.
- The cloud service must place the content to the /cdn folder on the web application.

Listing 3-30 shows the code for creating a blob and modifying its cache-control property so that it is cached in the CDN. The cache-control property decides the time to live for the cached blobs.

Listing 3-30. Creating CDN Blob

```
//Create storage credentials.

StorageCredentialsAccountAndKey credentials = new
StorageCredentialsAccountAndKey("silverliningsstorage1",

"m4AHAKXjfhlt2rE2BN/hcUR4U2lkGdCmj2/1ISutZK1+OqlrZN98Mhzq/U2AHYJT992tLmrkFW+mQmz9loIVCg==");

//Create a storage account instance
CloudStorageAccount storageAccount = new CloudStorageAccount(credentials, true);

//Create a new blob client instance
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

//Create a new container instance
CloudBlobContainer container = blobClient.GetContainerReference("mycdn");
//Create the container if it does not exist
container.CreateIfNotExist();

//Specify that the container is publicly accessible. This is a requirement for CDN
BlobContainerPermissions containerPermissions = new BlobContainerPermissions();
containerPermissions.PublicAccess = BlobContainerPublicAccessType.Container;
container.SetPermissions(containerPermissions);

//Create a new blob
CloudBlob blob = blobClient.GetBlobReference("mycdn/mytestblob.txt");
blob.UploadText("My first CDN Blob.");

//Set the Cache-Control header property of the blob and specify your desired
refresh interval (in seconds).

blob.Properties.CacheControl = "public, max-age=30036000";
blob.SetProperties();
```

After you have created the Blob, you can access it from your browser using the public URL `http://[your CDN GUID].vo.msecnd.net/mycdn/mytestblob.txt`
A few points to note about the CDN are:

- Your content is pushed to the edge cache only on first request.

- You will be charged for content cached on all the nodes.
- You cannot choose specific edge cache nodes to cache the content to, your content will be cached based on end-user requests.

Make sure you do due diligence on your end-user access points and perform a pricing exercise in early stages of development.

■ **Note** You can also stream blob contents to a video or music play. I recommend the following articles:

Smooth Streaming Video from Blob Storage msdn.microsoft.com/en-us/realdevelopment/hh285879

Adaptive Streaming with Windows Azure Blobs and CDN – By Steve Marx blog.smarx.com/posts/smooth-streaming-with-windows-azure-blobs-and-cdn

Windows Azure Drives

Windows Azure Drive provides durable NTFS volumes for your applications running in Windows Azure. Windows Azure Drives can only be mounted in role instances running in Windows Azure or in the Windows Azure local simulation environment. You cannot attach a Windows Azure Drive to your local machine and share it with Windows Azure instances. Windows Azure Drives, once mounted on to Windows Azure role instances, can be accessed as regular NTFS volumes from the cloud applications running on those role instances. In this section, I will cover Windows Azure Drives in detail and show you some examples for leveraging them in your code.

Overview

With mounted Windows Azure Drives, you can access the files system on the drive by referencing drive letters like Z:\ in your applications. You can perform most of the read and write file system operations like creating and reading files and folders. Windows Azure Drives are actually NTFS formatted VHD files stored in the Blob storage as Page Blobs. Page Blob has a maximum size limit of 1TB, and therefore the maximum size of a Windows Azure Drive cannot exceed 1TB. The Windows Azure team has written drivers that are installed on the role instances. These drivers expose the VHDs from Blob storage as drives when mounted using the Microsoft.WindowsAzure.CloudDrive API. When you perform write operations on a mounted Windows Azure Drive, the non-buffered data is written to the drive and is persistent across system reboots. This means that even if your role instance crashes, the data on the drive remains persisted to the VHD in Blob storage. You can access that data back after mounting the drive again to the new role instance. Windows Azure Drives also support caching of data locally on the role instance for improving reads. The cache size is specified through configuration and API while mounting the drive and it takes up the local drive space allocated to the role instance based on its size. Therefore, the drive cache cannot exceed the size of the local drive space. Each size of the virtual machine instance has a different local storage limit, therefore while designing your application; you should choose the size of your VM instance that fits your caching needs. You can also reduce the Blob service transaction cost by caching the data on local drive. One thing to be aware of is the cache does not

proactively cache data on the local drive. It caches the data when you first access it. Therefore, the first call to the data may be much slower than any subsequent calls.

Drive Operations

The API for Windows Drives operations are available in the `Microsoft.WindowsAzure.CloudDrive.dll` in the class `CloudDrive`. All the drive specific operations can be performed using the `CloudDrive` class. But, because the drive itself is a Page Blob, you can modify some of the Page blob properties using the `CloudBlob` class. Figure 3-16 shows the class diagram for classes and enumerations from the `Microsoft.WindowsAzure.Cloud.dll` assembly.

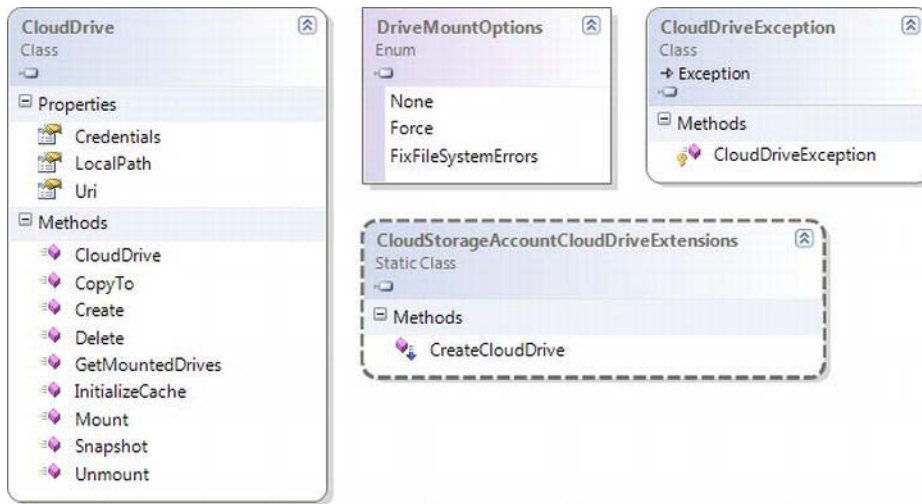


Figure 3-16. *CloudDrive class*

The `CloudDrive` class supports `Create`, `Delete`, `Mount`, `Snapshot`, `Copy`, and `Unmount` methods. You can call these methods only from a Windows Azure instance running in the local development fabric or the Windows Azure cloud. The `InitializeCache()` function initializes the cache on the local machine. To leverage the local cache, you need to first create a `LocalStorage` entry in the configuration with sufficient size. The cache consumes space from the total disk space available for the instance. Multiple cached drives will consume space from the total disk space available for the instance. Therefore, it is important to choose the right role instance type (i.e., Extra Small, Small, Medium, Large, and Extra Large) when you consider caching the drive data on local disk. Figure 3-17 illustrates the typical life cycle of a Windows Azure Drive.

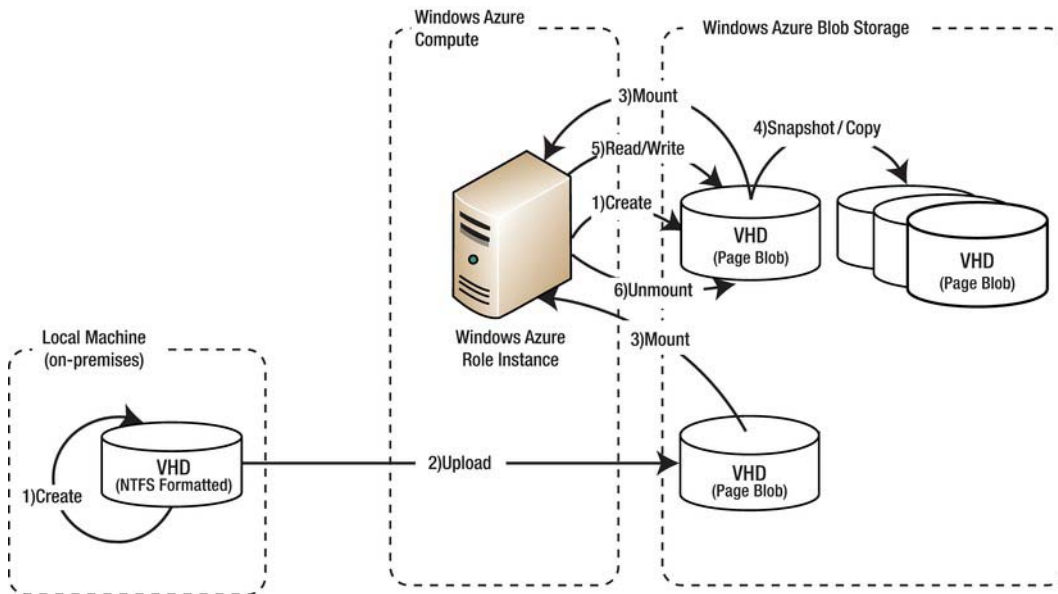


Figure 3-17. Life cycle of a Windows Azure Drive

The typical life cycle of a Windows Azure Drive is comprised of the following steps.

1. **Creating a Drive:** During this step, you either create a VHD locally or create a VHD directly from your Windows Azure role instance by calling the `CloudBlob.Create()` method from your role instance.
2. **Uploading a Drive:** If you created a drive locally, it needs to be uploaded to the Blob storage as a Page Blob to be visible to the Windows Azure role instances.
3. **Mounting a Drive:** Once a drive is available in the Blob storage, you can mount the drive from any Windows Azure role instance by calling the `CloudDrive.Mount()` method. The cloud drive operating system driver tries to acquire an exclusive access lease on the Page Blob representing the drive. If the lease succeeds, the drive will be mounted to a drive letter. One Page Blob drive can be mounted once and only once in a role instance for write access. A role instance can mount up to 16 drives.
4. **Working with a Drive:** After a drive is mounted, you will receive a drive letter that you can use to read and write data to the drive. It is recommended to geo-locate the drive's Page Blob in the same Windows Azure datacenter as the role instance mounting it. This will improve performance and you don't have to pay data transfer costs.
5. **Snapshotting a Drive:** You can take snapshot of a drive by calling the `CloudDrive.Snapshot()` method on an already mounted drive. Snapshot creates a readonly copy of the mounted drive. You can mount a snapshot to any number of role instances for readonly access. You cannot write to a drive

snapshot. I have used this functionality in high-scale compute architectures where a large amount of same input data is needed by multiple role instances running high performance calculations. These instances do not modify the input data but create a new set of output data which can be stored separately in Blob storage. You can create a writeable drive from a snapshot by calling the `CloudDrive.Copy()` method on the snapshot.

6. **Copying a Drive:** The `CloudDrive.Copy()` method allows you to create a writable copy of a snapshot or an unmounted drive. You can then mount the copy to another role instance. You cannot create a copy of a mounted drive, therefore, common pattern is to create a snapshot of a mounted drive, then create a copy and then delete the snapshot.
7. **Unmounting a Drive:** You can Unmount a drive from a role instance by calling the `CloudDrive.Unmount()` method. This will free the drive from the role instance and release the exclusive lock it had acquired during the `Mount()` operation.

Next, I will go into the details of each step of the drive's life cycle with an example.

Creating a Drive Locally

If you have Windows 7, you can use the Disk Management utility “`diskmgmt.msc`” utility for creating Fixed Size VHDs as shown in Figure 3-18. The following steps show you how:

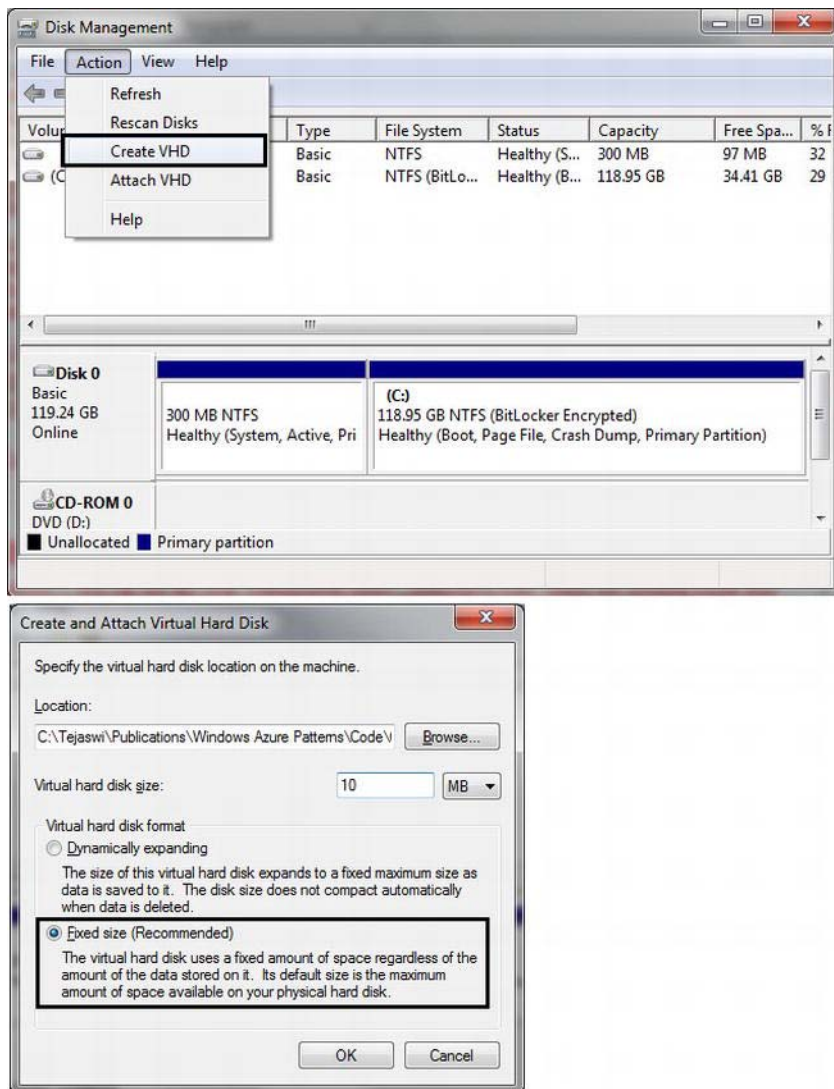


Figure 3-18. Creating a Fixed Size VHD in Windows 7

1. **Initialize the disk:** Once you create a disk, you can right-click on the disk and click select “Initialize Disk.” This will mount the drive Figure 3-19.

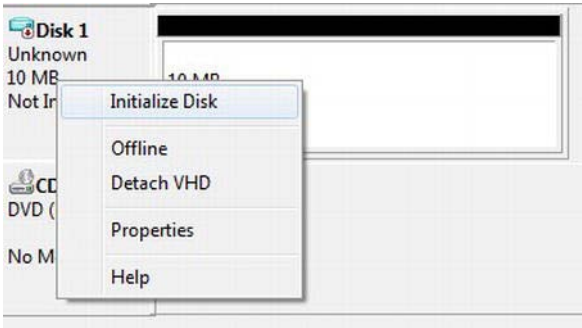


Figure 3-19. Initialize Disk

- 2. Next, create a new simple volume and format the disk in NTFS.
- 3. The drive gets mounted and you should see the volume created in the Disk Management console and also in Windows Explorer, as shown in the Figure 3-20.

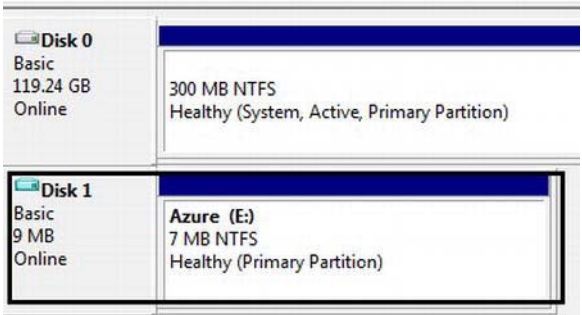
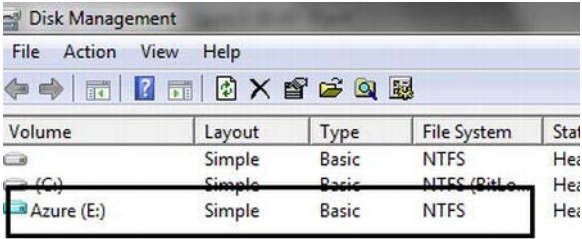


Figure 3-20. Drive mounted locally

- 4. You can then copy files to the drive, as shown in Figure 3-21.

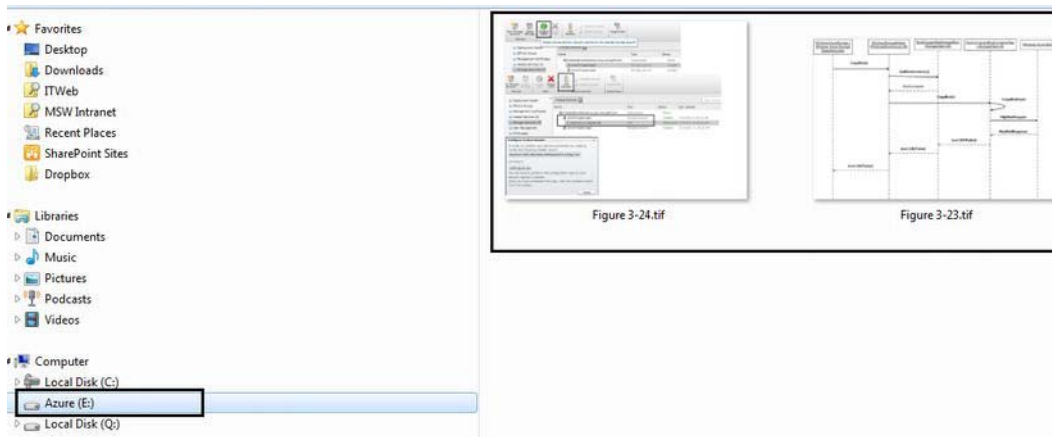


Figure 3-21. Copy files to local drive

5. After you are done copying files to the drive, detach the drive from the Disk Management. See Figure 3-22.

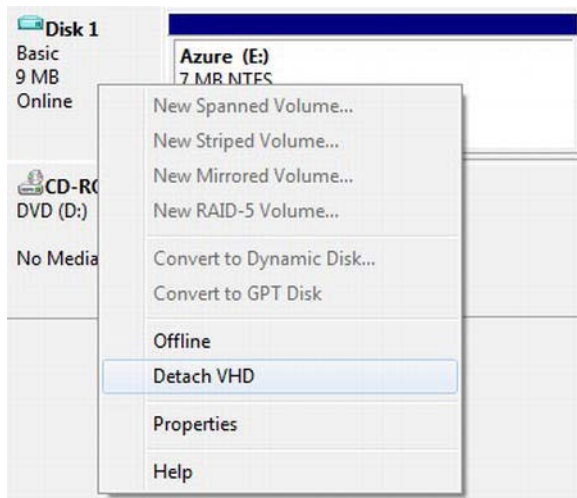


Figure 3-22. Detach the drive

■ **Note** Only Fixed Size VHDs are supported in Windows Azure Drives.

Uploading a Drive

A VHD disk can be uploaded to the Blob storage as a Page Blob. In this example, I have used Andy Edward's⁵ `vhduupload.exe` program for uploading the drive to Blob storage. The source code for the `vhduupload.exe` is available in the `Ch3Solution.sln` Visual Studio solution. The usage for `vhduupload.exe` is `Vhduupload.exe <local file> < blob url> <keyfile>`

Create a key file named `storagekey.txt` and copy your storage key in the file. Next, use the command as shown in the following:

```
vhduupload.exe myfirstdrive.vhd
http://silverliningstorage1.blob.core.windows.net/drives/myfirstdrive.vhd storagekey.txt
```

In this example, I have a public container named `drives` and I am uploading the drive `myfirstdrive.vhd` to that container. `Silverliningstorage1` is the name of my storage account.

Mounting a Drive

After the drive completes uploading, you can then mount the drive using the `Mount()` function, as shown in Listing 3-31.

Listing 3-31. Mount Drive

```
StorageCredentialsAccountAndKey credentials =
    new StorageCredentialsAccountAndKey("silverliningstorage1", storageAccountKey);
CloudDrive drive = new CloudDrive(blobURI, credentials);
drive.Mount(100000, DriveMountOptions.None);
```

■ **Note** You must run the Mounting operations in a Windows Azure role because the functions are not applicable on your local machine because the drives will be mounted on your Windows Azure instances and not on your local machine.

Creating and Mounting a Drive from a Role Instance

You can also create a drive and then mount a drive from the role instance. In this scenario, you don't have to upload a drive from your local machine. The `CloudDrive.Create()` function creates the drive as a Page Blob in the specified Blob storage account. Listing 3-32 shows the code for creating and mounting the drive in the same code segment.

⁵ Using Windows Azure Page Blobs and How to Efficiently Upload and Download Page Blobs, <http://blogs.msdn.com/b/windowsazurestorage/archive/2010/04/11/using-windows-azure-page-blobs-and-how-to-efficiently-upload-and-download-page-blobs.aspx>

Listing 3-32. Creating and Mounting a Drive from a Role Instance

```
// Create a storage account object
StorageCredentialsAccountAndKey credentials =
    new StorageCredentialsAccountAndKey("silverliningsstorage1", storageAccountKey);

try
{
    CloudDrive.InitializeCache(localCache.RootPath,
        10000);
    CloudDrive drive = new CloudDrive(BlobURI, credentials);
    drive.Create(sizeofDrive);
    string driveLetter = string.Empty;
    driveLetter = drive.Mount(cacheSize, DriveMountOptions.None);
}
catch (CloudDriveException ex)
{
    throw ex;
}
```

In Listing 3-24, the `InitializeCache()` function initializes the local cache for caching data from the drive. A new Page Blob is created for the drive in the specified URI. The URI may include a container that already exists in your Blob storage. Note that the size of the cache is specified during mounting of the drive.

■ **Note** When creating a drive using Blob URI, you can specify Shared Access Signature within the URI and pass null as credentials. You can create a granular security scheme at the Blob level.

Snapshotting a Drive

Snapshotting a drive creates a read-only copy of the drive. Listing 3-33 shows the listing of a function named `SnapshotAzureDrive()` available in `ProAzureCommonLib` Visual Studio project.

Listing 3-33. Snapshotting a Drive

```
public static string SnapshotAzureDrive(string accountName, string accountKey, string
azureDriveContainerName, string azureDrivePageBlobName)
{
    try
    {
        CloudStorageAccount csa = WStorageHelper.GetCloudStorageAccount(accountName,
accountKey, false);
```

```

        // Create the blob client

        CloudBlobClient client = csa.CreateCloudBlobClient();
        // Create the blob container which will contain the pageblob corresponding to
the azure drive.
        CloudBlobContainer container =
client.GetContainerReference(azureDriveContainerName);
        container.CreateIfNotExist();

        // Get the page blob reference which will be used by the azure drive.
        CloudPageBlob blob = container.GetPageBlobReference(azureDrivePageBlobName);
        CloudDrive drive = new CloudDrive(blob.Uri, csa.Credentials);
        if (drive != null)
        {
            return drive.Snapshot().ToString();
        }
    }
    catch (Exception ex)
    {
        WindowsAzureSystemHelper.LogError(String.Format("Error in snapshot drive {0}
- {1}", azureDrivePageBlobName, ex.Message));
    }

    return string.Empty;
}

```

Similar to the `Mount()` and `Snapshot()` methods, you can call the `Unmount()` method to unmount the drive from the compute instance. The `CopyTo()` method copies the data from a Windows Azure drive to a page blob. Next, let's look at some of the Windows Azure Drive scenarios.

■ **Note** The Windows Azure Storage team has written a blog post on sharing Windows Azure drives via SMB with multiple role instances. I recommend you to run this sample for understanding the possibilities of using Windows Azure Drives in your solution. blogs.msdn.com/b/windowsazurestorage/archive/2011/04/16/using-smb-to-share-a-windows-azure-drive-among-multiple-role-instances.aspx

Windows Azure Drives Scenarios

Windows Azure Drives can be created, mounted, copied and unmounted at run time. This give you the ability to leverage drives in several different scenarios as described here:

- **Data storage for third-party applications:** The new startup tasks with elevated features enable you to deploy even third-party applications in the cloud. Some of these applications have a pre-defined file storage structure or requires storage capacity in the cloud. Data stored on the local drive may get lost in case of instance failure. Storing this data on a drive will ensure that the data is persisted in the Blob storage and can be accessed anytime by a role instance by mounting it.
- **Readonly data storage for high-scale compute scenarios:** In typical high-scale compute scenarios, a large input dataset is made available to compute nodes for processing and generating results which are then persisted to file-base storage or a database. Usually, the input data is made available using file storage. High-scale compute scenarios are a sweet spot for Windows Azure, because you can dynamically scale up and scale down your system based on the input data load. In this scenario, you can upload the large input dataset in a Windows Azure Drive, then take a snapshot of that drive and publish the snapshot to multiple worker role instances for processing the data. Figure 3-23 illustrates the steps involved in publishing snapshot drives to multiple worker role instances.

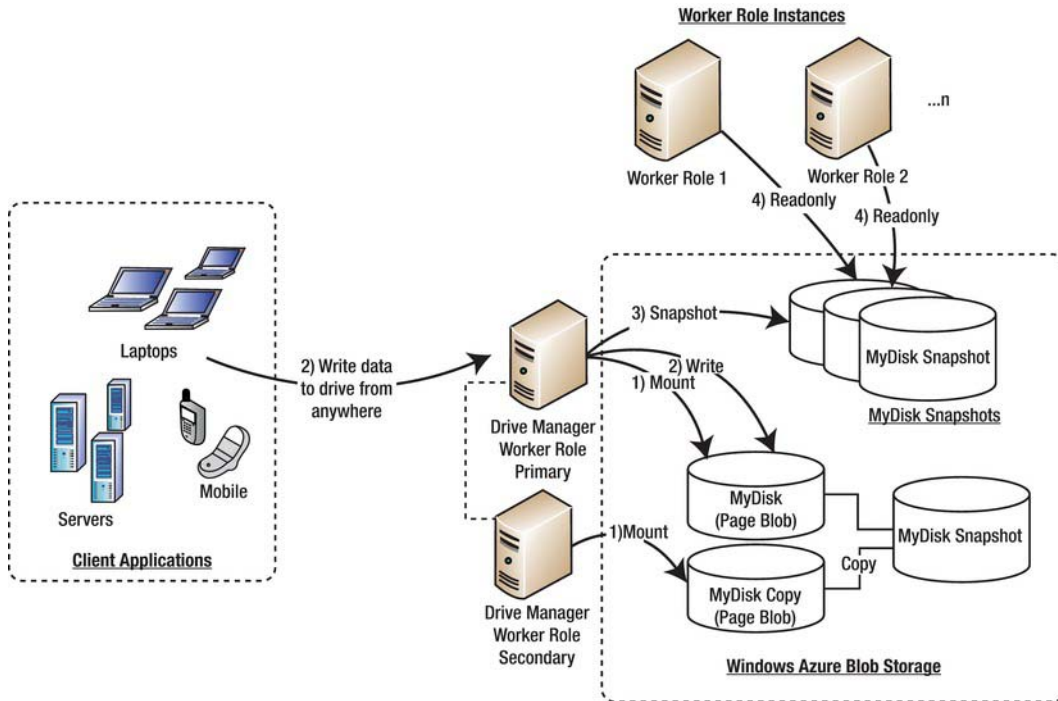


Figure 3-23. Using drive in Windows

In Figure 3-23, the Master Drive Manager is a Worker Role with external web service endpoints for writing data to mounted drives. Client applications call these web service methods for writing data to the MyDisk mounted drive. Whenever desired, the client applications can request a snapshot of the drive. The snapshot of the drive is then mounted on high-scale compute worker role instances for processing

the data uploaded to the drive. In this architecture, you can only upload the data differences to MyDisk and control the publishing of the data to high-scale compute nodes. This architecture can also work in data back-up scenarios where data is continuously pushed to the disk in the cloud and periodic snapshots are taken by an automated process. The risk in this architecture is the exposure of Master Drive Manager as a single point of failure. This can be mitigated by designing the following:

- Writing a monitoring package to restart the Master Drive Manager in case of a failure
- Creating a near real time copy of MyDisk by first taking a snapshot, then copying the snapshot to a writable drive and mounting the drive to another worker role instance for redundancy

You can then route the load-balanced requests from secondary to primary because sticky sessions are not available in Windows Azure load-balancers.

In case the primary node fails, you can promote secondary to primary. Of course, you have to maintain heart-beat between the primary and secondary instances for detecting these failures.

■ **Note** In one of the real-world applications, I have used a similar pattern for storing data for Intex software (http://www.intex.com/main/solutions_software.php). The Intex software runs on one of the Windows Azure VM Roles and receives data updates directly from Intex. The Intex software is unaware that it is writing to a Windows Azure drive instead of a locally mounted physical drive. After the data is updated, the snapshot of the drive is published to multiple role instances for financial data processing.

Blob Storage Scenarios

In this section, I have listed some commonly seen Blob Storage scenarios.

Massive Data Uploads

When you are migrating applications to Windows Azure, one important recommendation I have for you is to architect the data migration to early on during the planning phase of the project. Applications are easier to migrate, but massive data upload takes time. This is specifically true for media applications. The data uploads are limited by the bandwidth you experience from your datacenter to Windows Azure datacenters. Listed in the following are some recommendations for uploading massive data to Windows Azure Blob storage:

- Calculate the throughput you receive from your local machines to Windows Azure datacenters using the Azure Throughput Analyzer from Microsoft Research (<http://research.microsoft.com/en-us/downloads/5c8189b9-53aa-4d6a-a086-013d927e15a7/default.aspx>).
- Architect appropriate parallelism in your data upload application.
- Split the data into sets and then upload each set independently via parallel threads or processes.

- Keep a log of uploaded data and backlog to track progress of the data upload across all the parallel operations.
- Leverage CDNs for uploading data.
- I also recommend approaching your nearest Microsoft Technology Center (MTC)⁶ for uploading your data, because they have better network connections to Microsoft datacenters.

Figure 3-24 shows the result of one test I performed using the Azure Throughput Analyzer.

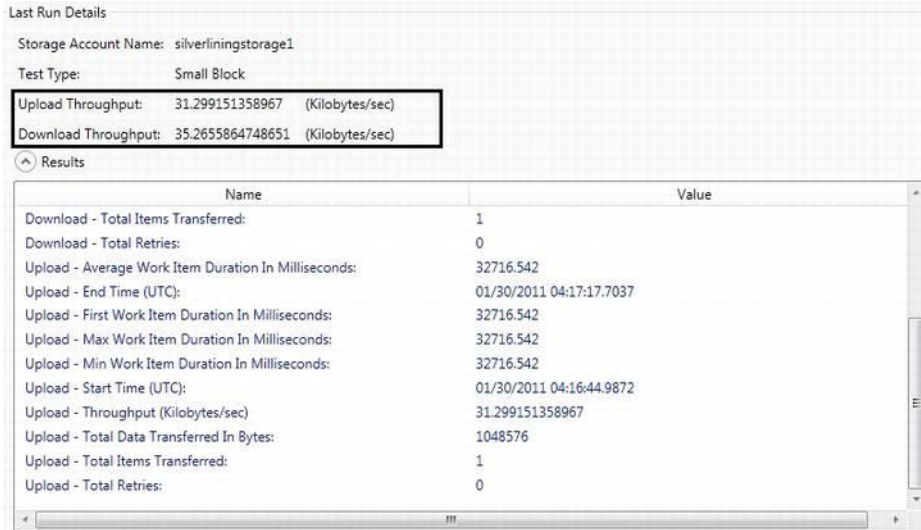


Figure 3-24. Azure Throughput Analyzer

Storage as a Service in the Cloud

Storage as a Service stores enterprise data in the cloud. Depending on the maturity of the cloud service provider and the type of enterprise data, several services like data backup, application backup and file synchronization are possible. For example, recently one of the larger Windows Azure customers uploaded their Windows 7 operating system images as blobs over CDN for deploying Windows 7 to all the enterprise desktops quickly and efficiently across the world. Now you may wonder, CDN does not offer blob security, wasn't the images open over the internet? The security requirement of a Windows 7 image file is very low because the images were deployed without activation keys. So, they were like downloading trial versions of Windows 7 from Microsoft's web site.

For designing enterprise storage as a service in the cloud, I recommend the prerequisites covered in the following sections.

⁶ Microsoft Technology Centers: <http://www.microsoft.com/mtc/default.mspx>

Integration with Your Enterprise Domain Accounts

A single shared key access to the storage gives too much power to the end-user. Therefore you need to have a Web or a Worker role façade managing role-based security by federating with your identity management system. The façade role is the only application that has access to the shared storage key.

■ **Note** I have covered federation with identity providers in Chapter 7.

Figure 3-25 illustrates the role façade pattern for managing enterprise user access to the blob storage.

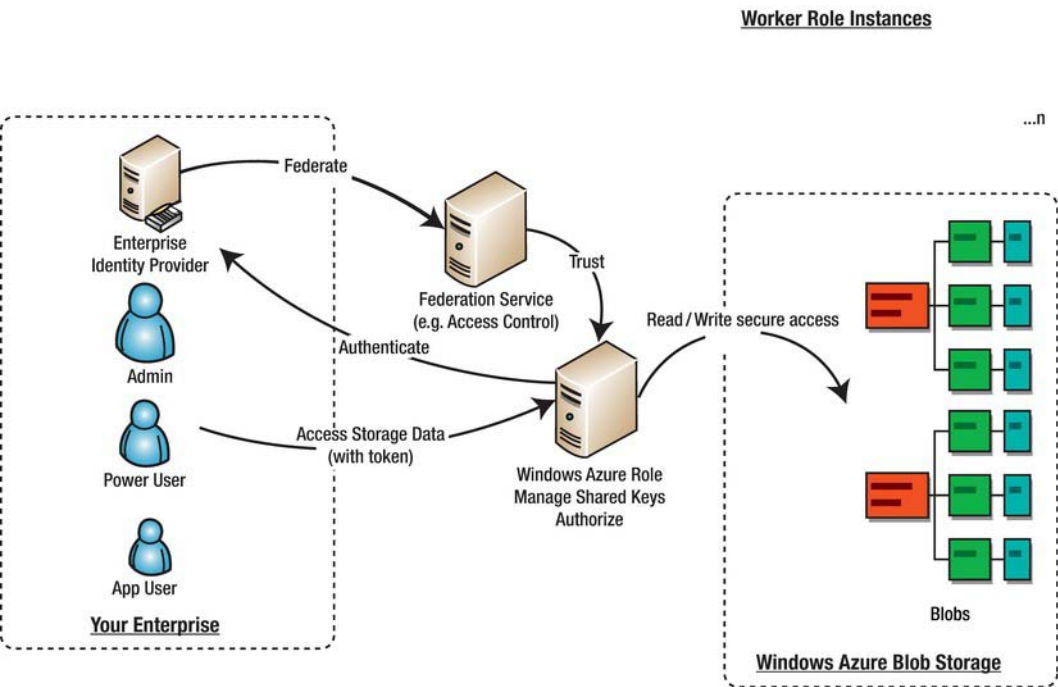


Figure 3-25. Role Façade over Blob storage

Storage Taxonomy Design

Designing the storage service for randomly storing enterprise data into the blob storage is going to turn into a maintenance nightmare. Have appropriate governance and management of the data in an organized manner is extremely important for optimizing your Blob storage usage. The first step towards a well-managed storage service is designing an extensible taxonomy of the data that will last for years to come. Remember that eventually the storage service will replace your storage area network (SAN). As a

recommendation, closely align the Blob storage taxonomy to your organization structure. This means you can allocate Blob storage by functional units and isolate them for maintenance. This will also help you create localized storage accounts in Asian datacenters for offices in Asia and align them back to the headquarter storage accounts.

Encryption and Decryption

For some of the sensitive data, encryption is required by enterprise IT. Windows Azure Blob storage does not provide you with encryption capabilities within the platform, therefore you have to encrypt and decrypt the data using your own encryption keys. One of the drawbacks of encrypting data is that it cannot be indexed because the search crawlers do not decrypt the data. For storing encrypted data in Blob storage, you can use any popular encryption technique: symmetric or asymmetric. In encryption architecture, the key management is the most important piece of the puzzle. The owner of the key is liable for the security of the key itself.

Enterprise File Sync

If you have ever used Windows Live Sync or Dropbox, you know the convenience these tools offer to consumers for synchronizing data not only between your machine and the cloud, but also from your machine to another group of machines. Offering a similar service within an enterprise will provide significant benefits in productivity. Imagine enterprise users do not have to worry about backing up their files because the enterprise file sync will automatically synchronize their files into Windows Azure Blob storage and also take periodic backups of the files right into the cloud. For designing Enterprise File Sync architecture, some of the software components that you may want to consider are the following:

- Microsoft Sync Framework 4.0 (<http://msdn.microsoft.com/en-us/sync/default.aspx>) for developing an end-to-end file synchronization application between your enterprise and Windows Azure.
- SQLAzure Data Sync (<http://www.microsoft.com/en-us/sqlazure/datasync.aspx>) for maintaining the relational data for the Enterprise Sync application.
- Windows Azure AppFabric Access Control (<http://msdn.microsoft.com/en-us/library/ee732536.aspx>).

Figure 3-26 illustrates a high-level diagram of a sample Enterprise File Sync service running in Windows Azure.

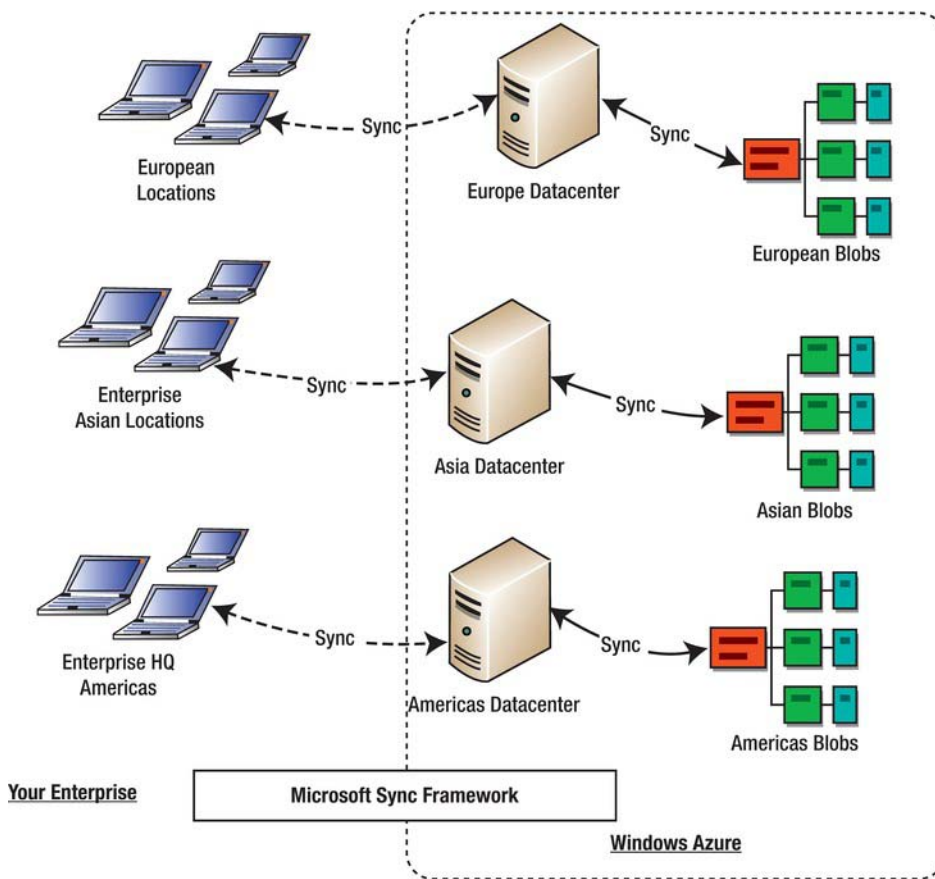


Figure 3-26. Enterprise File Sync

There are several interesting Blob scenarios that I would have loved to cover in this book, but had to stop here in the interest of space. These scenarios include Media Streaming, Disaster Recovery Data Repository, Application Store Repository, and Storing mobile applications and data.

■ **Note** In Windows Azure 1.5 SDK release, cross-region data replication was announced for the Windows Azure Storage service. This replication occurs automatically behind the scenes without any user intervention. You can find more information about 1.5 SDK release here:

blogs.msdn.com/b/windowsazurestorage/archive/2011/09/16/windows-azure-storage-at-build-2011-geo-replication-and-new-blob-table-and-queue-features.aspx

Summary

The Blob service is a scalable and highly available cloud storage designed to store any kind of file. It has a REST API that you can use to program applications against. You can use Blob storage from any kind of client that supports REST interactions. For Windows Azure compute applications, the Blob storage offers a great unstructured storage for storing just about any kind of file or even serialized objects. The Windows Azure SDK also provides a local Blob service emulator on top of local SQL Server Express. You can use the Windows Azure Storage Operations.exe application for testing Blob service scenarios. By reading this chapter, I hope you have gained enough knowledge for building your own Blobs and Drive applications. In the next chapter, I cover Windows Azure queues.

Bibliography

MSDN. (n.d.). *ADO.NET Data Services Specification*. Retrieved from MSDN Developer's Network:
<http://msdn.microsoft.com/en-us/library/cc668808.aspx>.

MSDN. (2009, May). *Windows Azure Blob — Programming Blob Storage*. Retrieved from MSDN:
<http://go.microsoft.com/fwlink/?LinkId=153400>.

MSDN. (2009, May). *Windows Azure Queue — Programming Queue Storage*. Retrieved from MSDN:
<http://go.microsoft.com/fwlink/?LinkId=153402>.

MSDN. (2009, May). *Windows Azure SDK*. Retrieved from MSDN: <http://msdn.microsoft.com/en-us/library/dd179367.aspx>.

MSDN. (2009, May). *Windows Azure Table — Programming Table Storage*. Retrieved from MSDN:
<http://go.microsoft.com/fwlink/?LinkId=153401>.

Windows Azure Storage

Part II – Queues

The Windows Azure Queue service is an Internet-scale message queuing system for cross-service communications. Even though the service is called a queue, the messages aren't guaranteed to follow the First In First Out (FIFO) pattern. The design focus of the Queue service is on providing a highly scalable and available asynchronous message communication system that's accessible anywhere, anytime. The Queue service is not a replacement for your on-premises Microsoft Message Queuing (MSMQ), because it lacks some of the features like transactional messaging, distributed transactions, and integration with domain security. But, most of the applications that do not use these MSMQ features should be able to replace it by the Queue service with minor modifications. The Queue service provides a REST API for applications to use the large-scale Queue service infrastructure. If you want to build an application that is agnostic to the type of queuing system it uses, you should build an abstraction layer using interface contracts and then let the implementation decide the type of queue. In real-world programming, you should always start with interfaces whether you have multiple implementations or not.

The Queue service is scoped at the account level. So, when you create an account on the Azure Services Developer Portal, you get access to the Queue service. Figure 4-1 illustrates the Management Portal page for the storage account that I created in the previous chapter and URL endpoint for the Queue service.

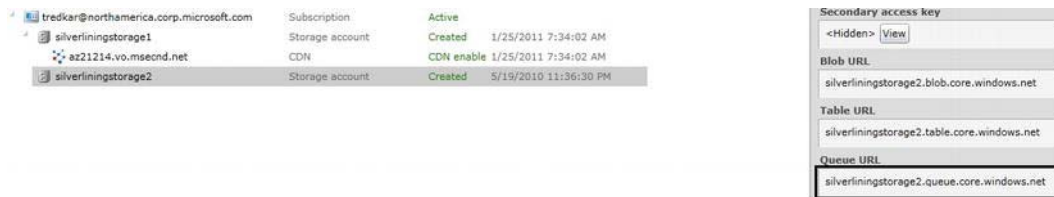


Figure 4-1. Queue endpoint URL

The account endpoint for the Queue service is `<account name>.queue.core.windows.net`, where `<account name>` is the unique name you created for your storage account. The secret key associated with the account provides security for accessing the storage account. You can use the secret key to create a Hash-based Message Authentication Code (HMAC) SHA256 signature for each request. The storage server uses the signature to authenticate the request.

■ **Note** HMAC is a message-authentication code calculated from the secret key using a special cryptographic hash function like MD5, SHA-1, or SHA256. The Windows Azure Storage service expects the SHA256 hash for the request. SHA256 is a 256-bit hash for the input data.

Queue Limitations and Constraints

Even though the Queue service provides a scalable and highly available infrastructure for asynchronous message communications in the cloud, it has some limitations and constraints that are important to understand before diving deep into architecture and programming. The limitations of the Queue service are as follows:

- The Queue service supports an unlimited number of messages, but individual messages in the Queue service can't be more than 64KB in size. If your object is larger than 64KB, you can store the actual object in Blob or Table storage and just send the link as a queue message.
- The FIFO behavior of the messages sent to the Queue service isn't guaranteed.
- Messages can be received in any order.
- The Queue service doesn't offer *guaranteed-once delivery*. This means a message may be received more than once.
- Messages sent to the Queue service can be in either text or binary format, but received messages are always in the Base64 encoded format.
- The expiration time for messages stored in the Queue service is seven days. After seven days, the messages are garbage-collected. Therefore, don't use Queue service as a permanent storage solution. Use Blob or Table storage for permanent storage.

■ **Caution** Do not expect Windows Azure Queues to deliver same performance as MSMQ or any other on-premises queuing system because the access protocol is still HTTP. HTTP(S) is a text-based protocol and not optimized for high-performance remote method invocations. The Windows Azure AppFabric may provide you with better performance options than Windows Azure queues. Windows Azure queues do offer highly available queues for light-weight communications.

Queue Service Architecture

The Queue service architecture consists of a three-level hierarchy: accounts, queues, and messages, as shown in Figure 4-2.



Figure 4-2. Queue service architecture

Your Windows Azure storage account is the entry point to the Queue service via the REST API.

■ **Note** In Windows Azure 1.5 SDK, some new features were announced to the Queue service like, larger message sizes (64KB instead of 8KB), message lease extension, and message update. You can find more information on the new features here:

blogs.msdn.com/b/windowsazurestorage/archive/2011/09/15/windows-azure-queues-improved-leases-progress-tracking-and-scheduling-of-future-work.aspx

Windows Azure Storage Account

The URI scheme for accessing the Queue service via your storage account is

`<http|https>://<account name>.queue.core.windows.net`

where `<account name>` is the unique name you created for your storage account. The `<account name>` must be globally unique.

For example, the Queue service for the storage account that I created in the previous chapter can be referenced as

`<http|https>://proazurestorage.queue.core.windows.net`

Queues

A *queue* is a logical destination for sending messages. There can be any number of queues in an account in the Queue service. A queue stores messages and makes them available to applications via the REST API. Queues can have metadata in the form of name-value pairs up to 8KB in size per queue. The Queue service support only private access; that means you need to have account privileges in order to access queues in a Queue service.

You can access a queue using the URI

```
<http|https>://<account name>.queue.core.windows.net/<queue name>
```

where *<queue name>* is the name of the queue you want to access.

For example, if you create a queue named logsqueue in the proazurestorage account, you can reference it using the following URI:

```
<http|https>://proazurestorage.queue.core.windows.net/logsqueue
```

The naming constraints for a queue are as follows:¹

- The queue name must be a valid DNS name.
- Queue names must be unique within an account.
- Queue names must start with a letter or a number.
- Container names can't contain any special characters other than the dash (-) character.
- The dash (-) character must be immediately followed by a character or a number.
- All the characters in the queue name must be lowercase.
- Queue names can't be less than 3 or more than 63 characters in length.

If a queue name or the URI violates the naming convention, an HTTP status code 400 (Bad Request) is returned by the server.

Messages

Messages are stored in queues. There is no limit to the number of messages that can be stored in a queue, but the size of each individual message can't exceed 8KB. To communicate large object messages, you can put the large object in a Blob and then send the URI of that object as a message to a queue.

When you send a message, it can be in either text or binary format; but when you receive a message from the queue, it's always in Base64-encoded format. A GUID MessageID assigned by the Queue service uniquely identifies a message in the queue.

A message has the following attributes:

- **MessageID** – Uniquely identifies a message in a queue and is created by the Queue service when you send the message to the Queue service.

¹Source: Windows Azure SDK documentation

- **PopReceipt** – An attribute of the message used for deleting or popping the message out from the queue.
- **Visibilitytimeout** – An integer value representing the visibility of the message in seconds after it's received by a receiving application. The default visibilitytimeout value is 30 seconds, which means that after a message is received, it remains invisible to other applications for 30 seconds (unless it's deleted by the receiving application). The maximum visibilitytimeout value is two hours. The visibilitytimeout value is passed to the Queue service while a message is being retrieved from a queue. From the message receiver, you can control the visibility timeout in such a way that processing of the message is completed within the visibility timeout period, which includes deleting the message from the queue. If the processing is not complete and message not explicitly deleted within the visibility timeout period, the system will assume that the processing failed and the message will be visible again in the queue for other receivers to process.
- **Messagettl** – An integer value representing the time-to-live value in seconds. When you send a message to a queue, you can specify the messagettl, which instructs the Queue service to keep the message only for the specified number of seconds. The default value for messagettl is seven days. That means if you don't specify a messagettl value, the Queue service keeps the message for seven days before it's garbage-collected.

You can access messages in a queue using this URI

```
<http|https>://<account name>.queue.core.windows.net/<queue name>/messages
```

where *<queue name>* is the unique name of the queue within the scope of the account specified in the URI, and messages is a constant string representing all the messages in the specified queue. For example, if you create a queue named logsqueue, you can get messages from it by calling the following URI:

```
<http|https>://proazurestorage.queue.core.windows.net/logsqueue/messages
```

REST API

The REST API for the Queue service is available at the account, queue, and message levels. In this section, you learn about the Queue service REST API with specific examples. You also learn to interact with the Queue service programmatically, and you explore the queue methods in the available storage client libraries.

The REST API enables you to make HTTP calls to the Queue service and its resources. REST is an HTTP-based protocol that lets you specify the URI of the resource as well as the function you want to execute on the resource. Every REST call involves an HTTP request to the storage service and an HTTP response from the storage service.

Request

The Queue service REST API's HTTP request components are described in the following sections.

HTTP Verb

The HTTP verb represents the action or operation you can execute on the resource indicated in the URI. The Queue service REST API supports the following verbs: GET, PUT, POST, HEAD, and DELETE. Each verb behaves differently when executed on a different resource.

Request URI

The request URI represents the URI of a resource you're interested in accessing or executing a function on. Example resources in the Queue service include accounts, queues, and messages. An example URI for creating a queue named logsqueue in an account named proazurestorage is

```
PUT http://proazurestorage.queue.core.windows.net/logsqueue
```

The HTTP verb PUT instructs the service to create the queue, and the URI points to the resource that needs to be created.

URI Parameters

The URI parameters are the extra parameters you specify to fine-tune your operation execution. They may include operation parameters or filter parameters for the results. In the Queue service API, the URI parameters depend on the type of resource and the HTTP verb used. For example, a URI for retrieving a list of queues from an account looks like this:

```
GET http://proazurestorage.queue.core.windows.net/?comp=list
```

The HTTP verb GET instructs the Queue service to retrieve results, and the parameter ?comp=list specifies that the data requested is a list of queues.

Request Headers

Request headers follow the standard HTTP 1.1 name-value pair format. Depending on the type of request, the header may contain security, date/time, metadata, or instructions embedded as name-value pairs. In the Storage service REST API, the request header must include the authorization information and a Coordinated Universal Time (UTC) timestamp for the request. The timestamp can be in the form of either an HTTP/HTTPS Date header or the x-ms-Date header.

The authorization header format is as follows:

```
Authorization="[SharedKey|SharedKeyLite] <Account Name>:<Signature>"
```

Where SharedKey|SharedKeyLite is the authentication scheme, <Account Name> is the storage service account name, and <Signature> is an HMAC of the request computed using the SHA256 algorithm and then encoded by using Base64 encoding.

To create the signature, follow these steps:

1. Create the signature string for signing. The signature string for the Storage service request consists of the following format:

```
VERB\n
Content - MD5\n
Content - Type\n
```

Date\n
CanonicalizedHeaders
CanonicalizedResource

where VERB is the uppercase HTTP verb such as GET, PUT, and so on; *Content* — MD5 is the MD5 hash of the request content; *CanonicalizedHeaders* is the portion of the signature string created using a series of steps described in the “Authentication Schemes” section of the Windows Azure SDK documentation (<http://msdn.microsoft.com/en-us/library/dd179428.aspx>); and *CanonicalizedResource* is the storage service resource in the request URI. The *CanonicalizedResource* string is also constructed using a series of steps described in the “Authentication Schemes” section of the Windows Azure SDK documentation.

2. Use the `System.Security.Cryptography.HMACSHA256.ComputeHash()` method to compute the SHA256 HMAC-encoded string.
3. Use the `System.Convert.ToBase64String()` method to convert the encoded signature to Base64 format.

Listing 4-1 shows an example request header that sets the metadata values of a queue.

Listing 4-1. Request Header

```
PUT /myfirstazurequeue?comp=metadata&timeout=30 HTTP/1.1
x-ms-date: Wed, 17 Jun 2009 04:33:45 GMT
x-ms-meta-createdBy: tejaswi
x-ms-meta-creationDate: 6/16/2009
Authorization: SharedKey proazurestorage:
    spPPnadPYnH6AJguuYT9wP1GLXmCjn0I1S6W2+hzyMc=
Host: proazurestorage.queue.core.windows.net
Content-Length: 0
```

In Listing 4-1, the request header consists of `x-ms-date`, `x-ms-version`, `x-ms-[name]:[value]`, and `Authorization` values. `x-ms-date` represents the UTC timestamp, and `x-ms-version` specifies the version of the storage service API you’re using. `x-ms-version` isn’t a required parameter, but if you don’t specify, you have to make sure the operation you’re calling is available in the default version of the Queue service. Before making the REST call, be sure you match the operation you’re calling with the API version it’s supported in. It’s always safe to match the operation with the version to get the expected results. The `x-ms-meta` values represent the queue metadata name-value pairs the operation should set. The last header value is the `Authorization SharedKey` used by the Storage service to authenticate and authorize the caller.

■ **Note** Unlike the Blob service REST API, the Queue service REST API doesn’t support HTTP 1.1 conditional headers.

Request Body

The request body consists of the contents of the request operation. Some operations require a request body and some don't. For example, the Put Message operation request body consists of the message data in XML format, whereas the Get Messages operation requires an empty request body.

Response

The HTTP response of the Queue service API typically includes the following components.

Status Code

The status code is the HTTP status code that indicates the success or failure of the request. The most common status codes for the Queue service API are 200 (OK), 201 (Created), 204 (No Content), 400 (BadRequest), 404 (NotFound), and 409 (Conflict).

Response Headers

The response headers include all the standard HTTP 1.1 headers plus any operation-specific headers returned by the Queue service. The x-ms-request-id response header uniquely identifies a request. Listing 4-2 shows an example response header for a List Queues operation.

Listing 4-2. List Queues Response Header

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/xml
Server: Queue Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: ccf3c21c-7cca-4386-a636-7f0087002970
Date: Tue, 16 Jun 2009 04:47:54 GMT
```

Response Body

The response body consists of data returned by the operation. This data is specific to each operation. For example, the List Queues operation returns the list of queues in an account, whereas the Get Messages operation returns the messages in a queue. Listing 4-3 shows an example of the response body for a List Queues operation. The response contains four queues.

Listing 4-3. List Queues Response Body

```
<?xml version="1.0" encoding="utf-8"?>
<EnumerationResults AccountName="http://proazurestorage.queue.core.windows.net/">
  <MaxResults>50</MaxResults>
  <Queues>
    <Queue>
      <QueueName>testq</QueueName>
```

```

<Url>http://proazurestorage.queue.core.windows.net/testq</Url>
</Queue>
<Queue>

<QueueName>testq1</QueueName>
<Url>http://proazurestorage.queue.core.windows.net/testq1</Url>
</Queue>

<Queue>

<QueueName>testq2</QueueName>

<Url>http://proazurestorage.queue.core.windows.net/testq2</Url>
</Queue>
<Queue>

<QueueName>testq3</QueueName>

<Url>http://proazurestorage.queue.core.windows.net/testq3</Url>
</Queue>
</Queues>
<NextMarker />
</EnumerationResults>

```

■ **Tip** To test the REST API, I recommend using the Fiddler Tool available at www.fiddler2.com/fiddler2/. In this book, I have used this tool to trace client/server communications.

Storage Client API

Even though the REST API and the operations in the REST API are easily readable, the API doesn't automatically create client stubs like the ones created by WDSL-based web services. You have to create your own client API and stubs for REST API operations. The Windows Azure SDK team has created a client helper managed code library: `Microsoft.WindowsAzure.StorageClient` from Windows Azure SDK. Behind the scenes, the client API invokes the REST APIs of the Windows Azure Queue Storage service. The `Microsoft.WindowsAzure.StorageClient` library abstracts this by providing a closed-source interface and therefore is easier for developers to extend it and use it directly from your code.

In the following sections, I will cover the class structure and calling mechanisms from the `Microsoft.WindowsAzure.StorageClient` assembly.

Windows Azure Storage Client Queue API

The `Microsoft.WindowsAzure.StorageClient` namespace consists of classes representing the entire queue hierarchy. Figure 4-3 illustrates the core classes for programming Queue service applications.

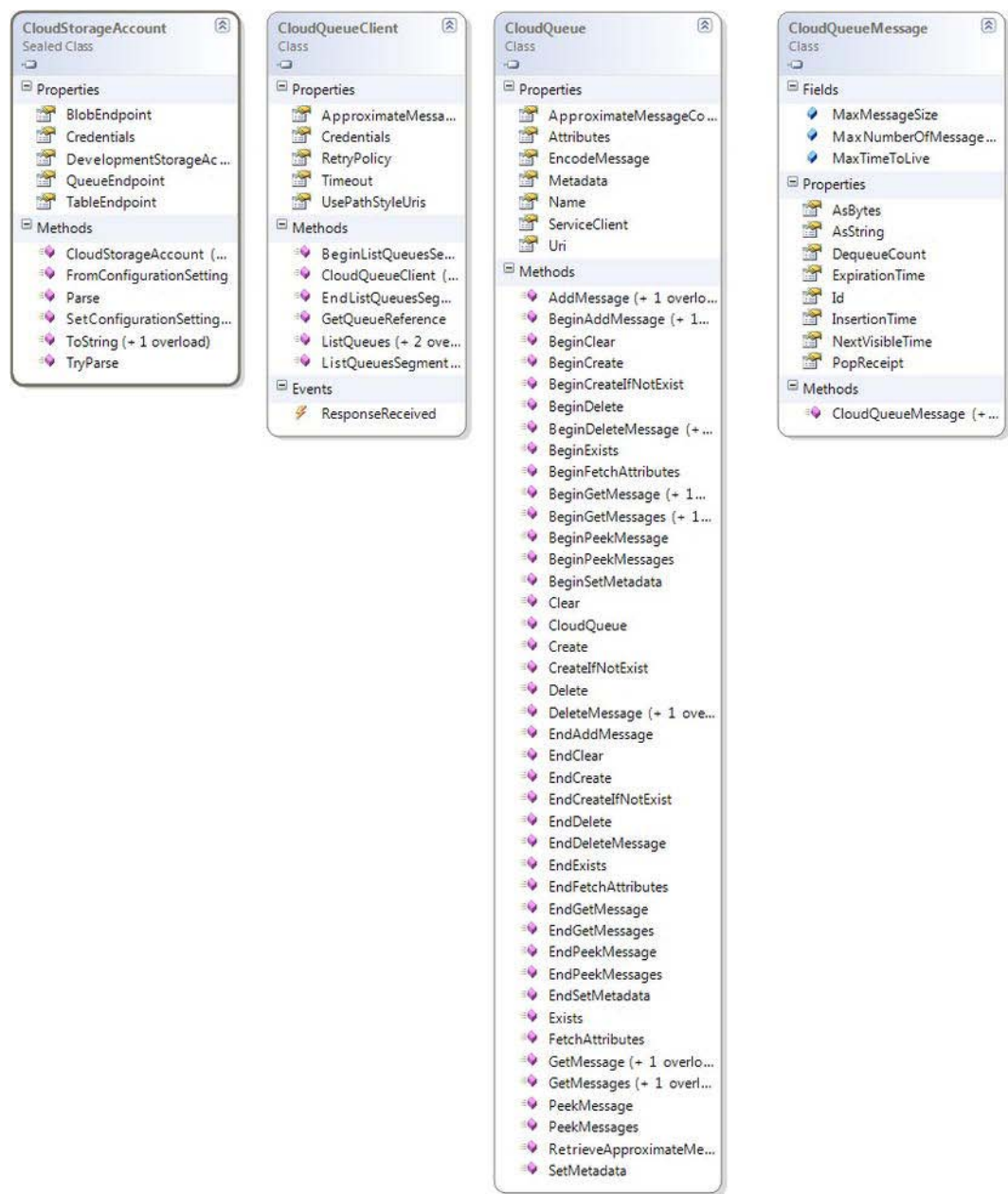


Figure 4-3. Queue class hierarchy

As shown in Figure 4-3, four core classes are required for queue operations. Table 4-1 provides a short description of each of them.

■ **Tip** The Windows Azure Storage Client API is the recommended method for programming Storage service applications. The API provides synchronous as well as asynchronous methods for interacting with the Storage service REST API.

Table 4-1. Classes for the Queue Service

Class Name	Description
CloudStorageAccount	A helper class for retrieving account information from the configuration file or creating an instance of the storage account object from account parameters.
CloudQueueClient	A wrapper class for getting references to the core queue objects. The class consists of methods like <code>GetQueueReference()</code> and <code>ListQueues()</code> .
CloudQueue	Consists of queue operations like <code>Create()</code> , <code>Delete()</code> , <code>AddMessage()</code> , and <code>GetMessage()</code> .
CloudQueueMessage	Represents a queue message with properties like <code>InsertionTime</code> , <code>ExpirationTime</code> , <code>NextVisibleTime</code> , <code>ID</code> , and <code>PopReceipt</code> .

In addition to these core classes, classes like `QueueAttributes` and `QueueErrorCodeStrings` represent more details about the queue.

The steps for programming simple queue applications with the queue classes listed in Table 4-1 are as follows:

1. Add the following using statement to your C# class:

```
using Microsoft.WindowsAzure.StorageClient;
```

2. Instantiate the `CloudStorageAccount` class from the configuration file:

```
CloudStorageAccount storageAccountInfo =  
CloudStorageAccount.FromConfigurationSetting(configurationSettingName);
```

3. Or, instantiate the `CloudStorageAccount` class using account information:

```
CloudStorageAccount storageAccountInfo = new CloudStorageAccount(new  
StorageCredentialsAccountAndKey(accountName, accountKey), new Uri(blobEndpointURI), new  
Uri(queueEndpointURI), new Uri(tableEndpointURI));
```

4. Create an instance of `CloudQueueClient`:

```
CloudQueueClient queueStorageType = storageAccountInfo.CreateCloudQueueClient ();
```

When you have an instance of the `CloudQueueClient` class, you can execute operations on the queue storage service as follows:

List queues:

```
IEnumerable<CloudQueue> queues = queueStorageType.ListQueues();
Create Queue
queueStorageType.GetQueueReference(queueName).Create();
Delete Queue
queueStorageType.GetQueueReference(queueName).Delete();
```

Add a message:

```
public void AddMessage(string queueName, CloudQueueMessage queueMessage)
{
    queueStorageType.GetQueueReference(queueName).AddMessage(queueMessage);
}
```

Get messages:

```
queueStorageType.GetQueueReference(queueName).GetMessages(numberofMessages,
    TimeSpan.FromSeconds(visibilityTimeoutInSecs));
```

Peek messages:

```
queueStorageType.GetQueueReference(queueName).PeekMessages(numberofMessages);
```

Delete a message:

```
public void DeleteMessage(string queueName, CloudQueueMessage queueMessage)
{
    queueStorageType.GetQueueReference(queueName).DeleteMessage(queueMessage);
}
```

Set queue metadata:

```
public void SetQueueMetadata(string queueName, NameValueCollection queueProps)
{
    CloudQueue queue = queueStorageType.GetQueueReference(queueName);
    queue.Attributes.Metadata = queueProps;
    queue.SetMetadata();
}
```

The call to `SetMetadata()` method calls the method on the queue service API in the cloud.

In the next few sections, you learn how to call some of these functions at every level of the Queue service hierarchy.

Account Operations

The storage account provides an entry point to the Queue service via the Queue service endpoint URI. At the account level of the hierarchy, the Queue service supports only one operation: List Queues. The URI of a specific account is of the format <account name>.queue.core.windows.net. Table 4-2 describes the List Queues operation, and Table 4-3 lists some important characteristics of the List Queues function.

Table 4-2. Queue Account Operation

Operation	Description
List Queues	This operation gets the list of all the queues in a storage account. You can limit the number of records returned by specifying a filter on queue names and the size of the data set in the request. Table 4-4 lists all the possible URI parameters for this operation.

Table 4-3. Queue Account Operations Characteristics

Operation	HTTP Verb	Cloud URI	Development Storage URI	HTTP Version	Permissions
List Queues	GET	account name>.queue.core.windows.net?comp=list	http://127.0.0.1:10001/<devstorageaccount>?comp=list	HTTP/1.1	Only the account owner can call this operation.

<account name> is the storage account name, such as proazurestorage; and <devstorageaccount> is the account name for the development storage. The HTTP verb used in this operation is GET. The table lists the URI format for accessing the cloud Queue service as well as the development storage URI. Port 10001 is the default Queue service port in the development fabric.

The URI for the List Queues operation supports additional optional parameters, as listed in Table 4-4.

Table 4-4. List Queues URI Parameters

Parameter	Description	Example
prefix	A filter parameter for returning queues starting with the specified prefix value.	http://proazurestorage.queue.core.windows.net/?comp=list&prefix=may returns queues with names starting with the prefix “may.”
marker	Used for paging queue results when all results aren't returned by the Storage service either due to the default maximum results allowed (the current default is 5000), or because you specify the maxresults parameter in the URI. The marker prefix is opaque to the client application.	http://proazurestorage.queue.core.windows.net/?comp=list&prefix=may&marker=/proazurestorage/testq
maxresults	The maximum number of queues the Queue service should return. The default value is 5000. The	http://proazurestorage.queue.core.windows.net/?comp=list&prefix=may&

server returns HTTP Bad Request (400) code if you specify a maxresults value greater than 5000.

The sample REST request for List Queues in raw format looks like Listing 4-4.

Listing 4-4. List Queues REST Request

```
GET /?comp=list&prefix=test&maxresults=50&timeout=30 HTTP/1.1
x-ms-date: Wed, 27 May 2009 04:33:00 GMT
Authorization: SharedKey proazurestorage:GCvS8cv4Em6rWMuCVix9YCsxVgssOW62S2U8zjbIa1w=
Host: proazurestorage.queue.core.windows.net
Connection: Keep-Alive
```

The characteristics of the REST request in Listing 4-4 are as follows:

- The parameter comp=list at the account level of the Queue service yields the list of all the queues.
- The prefix=test filters the results by queue names starting with “test.”
- The maxresults=50 returns 50 queues or less.
- The x-ms-date is the UTC timestamp of the request.
- The Authorization header contains the SharedKey of the request.
- The Host header points to the Queue service in the cloud.

Because the request is sending a maxresults parameter, it makes sense to keep the HTTP connection alive, because it’s highly likely that the user will retrieve the next set of results by making another call to the Queue service.

Listing 4-5 shows the response for the List Queues request.

Listing 4-5. List Queues REST Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Server: Queue Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: dde8c8bd-121d-4692-a578-d8fac08e4525
Date: Wed, 17 Jun 2009 01:24:45 GMT
Content-Length: 648

<?xml version="1.0" encoding="utf-8"?>
<EnumerationResults AccountName="http://proazurestorage.queue.core.windows.net/">
  <Prefix>test</Prefix>
  <MaxResults>50</MaxResults>
  <Queues>
    <Queue>
      <QueueName>testq</QueueName>
      <Url>http://proazurestorage.queue.core.windows.net/testq</Url>
    </Queue>
```

```

<Queue>
<QueueName>testq1</QueueName>
<Url>http://proazurestorage.queue.core.windows.net/testq1</Url>
</Queue>
<Queue>
<QueueName>testq2</QueueName>
<Url>http://proazurestorage.queue.core.windows.net/testq2</Url>
</Queue>
<Queue>
<QueueName>testq3</QueueName>
<Url>http://proazurestorage.queue.core.windows.net/testq3</Url>
</Queue>
</Queues>
<NextMarker />
</EnumerationResults>

```

In Listing 4-5, the header consists of the HTTP status (200 OK) indicating the success of the operation. The response body is in XML format with `<EnumerationResults />` as the root element. The `<Queues />` element contains the retrieved queues. The Queue element encompasses queue attributes like the queue name and the queue URI. An empty `<NextMarker />` element indicates that all the results have been retrieved.

To help you understand the Queue service programming model, open the Windows Azure Storage Operations project from `Ch4Solution.sln`. The project consists of a Windows form and uses the `StorageClient` project from the same solution for making calls to all the Windows Azure storage. The `StorageClient` project is shipped with the Windows Azure SDK. I also created a helper class named `WASStorageHelper` in the `ProAzureCommonLib` project for wrapping the `StorageClient` methods. Figure 4-4 shows the user interface for the Windows Azure Storage Operations application as it pertains to the Operations account of the Queue service.

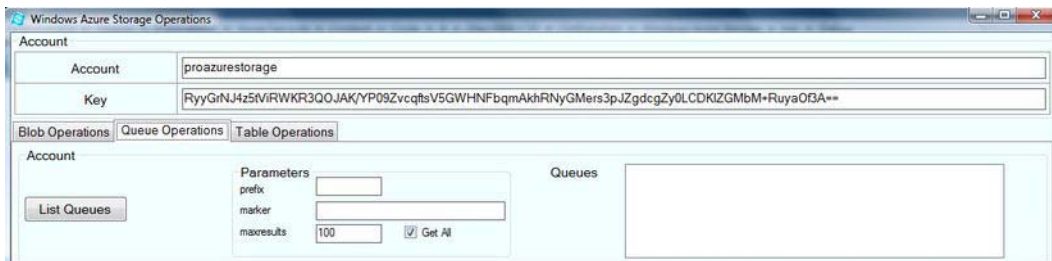


Figure 4-4. Windows Azure storage Queue service account operations

In Figure 4-4, the top Account section displays the account name and SharedKey of the storage account. When the Windows Azure Storage Operations.exe application starts, it loads the account information from the configuration file. Listing 4-6 shows the account configuration in the project's `app.config` file.

Listing 4-6. *App.config*

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="StorageAccountConnectionString" value="UseDevelopmentStorage=true"/>
  </appSettings>
</configuration>
```

The `StorageAccountConnectionString` is loaded when the application starts and displays in the textbox. Before starting the application, make sure you modify this connection string to point to your storage account, except when you are using development storage. The `QueueStorageEndpoint` is the URI of the Queue service.

The `WASStorageHelper` class in `ProAzureCommonLib` project has a `ListQueue()` method for retrieving queue names. Listing 4-7 shows the code for `ListQueues()` method.

Listing 4-7. *ListQueues() Method*

```
public IEnumerable<CloudQueue> ListQueues(string prefix)
{
    if (string.IsNullOrEmpty(prefix))
    {
        return this.QueueClient.ListQueues();
    }
    else
    {
        return this.QueueClient.ListQueues(prefix);
    }
}
```

In Listing 4-7, the `ListQueues()` method calls the `ListQueues()` method on the `QueueClient` object, which is of type `CloudQueueClient` from the `StorageClient` assembly. The first method returns all the queues in an account or filtered based on the prefix. Figure 4-5 illustrate the execution of the `ListQueues` operation in the Windows Azure Storage Operations application.



Figure 4-5. *List Queues operation*

When you click the `List Queues` button, the application retrieves queues from the Queue service and displays the names of queues in the Queues `ListBox`. In the parameter section, you can specify the prefix.

Queue Operations

Queues support several operations, as listed in Table 4-5.

Table 4-5. Queue Operations

Operation	Description
Create Queue	Creates a new queue under the given account. You can specify metadata for the queue during creation.
Delete Queue	Marks the specified queue for deletion. The garbage collector deletes marked queues on a periodic basis. So, if you delete a queue and try to create it immediately, the Queue service complains that the queue already exists.
Get Queue Metadata	Gets the user-defined queue metadata and other queue properties. The metadata is retrieved in the form of name-value pairs.
Set Queue Metadata	Sets the metadata values of the specified queue. Set Queue Metadata replaces all the metadata of the specified queue with new values.

Table 4-6 lists some of the important characteristics of the queue operations listed in Table 4-5.

Table 4-6. Queue Operations Characteristics

Operation	HTTP Verb	Cloud URI	Development Storage URI	HTTP Version	Permissions
Create Queue	PUT	<code>http://<account name>.queue.core.windows.net/<queue name></code>	<code>http://127.0.0.1:10001/<devstorageaccount>/<queue name></code>	HTTP/1.1	Only the account owner can call this operation.
Delete Queue	DELETE	<code>http://<account name>.queue.core.windows.net/<queue name></code>	<code>http://127.0.0.1:10001/<devstorageaccount>/<queue name></code>	HTTP/1.1	Only the account owner can call this operation.
Get Queue Metadata	GET/HEAD	<code>http://<account name>.queue.core.windows.net/<queue name>?comp=metadata</code>	<code>http://127.0.0.1:10001/<devstorageaccount>/<queue name>?comp=metadata</code>	HTTP/1.1	Only the account owner can call this operation.
Set Queue Metadata	PUT	<code>http://<account name>.queue.core.windows.net/<queue name></code>	<code>http://127.0.0.1:10001/<devstorageaccount>/<queue name></code>	HTTP/1.1	Only the account owner can call this operation.

<code>name>?comp=metadata</code>	<code>name>?comp=</code> <code>metadata</code>
-------------------------------------	--

Table 4-6 lists the HTTP verb, cloud URI, development storage URI, HTTP version, and access control for the queues. The `<account name>` is the storage account name in the cloud, and the `<devstorageaccount>` is the development storage account. Observe that unlike blob containers, all the operations can be called only with the account owner privileges.

The following sections discuss some of the operations from Table 4-7 in detail. Even though the operations are different, the programming concepts behind them are similar. To keep the book at a conceptual level, I discuss just the Create Queue and Set Queue Metadata operations. By studying these operations in detail, you can understand the programming concepts behind all the queue operations. The Windows Azure Storage Operations application included with this chapter's source code contains an implementation of all the queue operations.

Create Queue

The Create Queue operation creates a queue in a storage account. The URI for the Create Queue operation is of the format `account name>.queue.core.windows.net/<queue name>`. You can think of Queue as a message queuing system in the cloud. For example, if you want to send and receive messages across diverse applications in different domains, Windows Azure Queue may fit your requirement. Because of its standard REST interface and Internet scale, you can send and receive queue messages anywhere, anytime, and in any programming language that supports Internet programming. The Create Queue REST request looks like Listing 4-8.

Listing 4-8. Create Queue REST Request

```
PUT /myfirstazurequeue?timeout=30 HTTP/1.1
x-ms-date: Wed, 17 Jun 2009 03:16:12 GMT
Authorization: SharedKey proazurestorage:a0EQSlfMdxFrP/wdfCUVqMYiv4PjXesF0Jp4d71DA=
Host: proazurestorage.queue.core.windows.net
Content-Length: 0
```

Listing 4-8 shows the request for creating a queue named `myfirstazurequeue`. The PUT HTTP verb instructs the Queue service to create a queue. There is no metadata information for the queue, so the queue is created without any metadata. You can add `x-ms-meta-[name]:[value]` to the header to create metadata values. For the Create Queue operation, the Queue service responds with a status code of HTTP/1.1 201 Created, or HTTP/1.1 409 Conflict if a queue with the same name already exists. The Create Queue response is shown in Listing 4-9.

Listing 4-9. Create Queue REST Response

```
HTTP/1.1 201 Created
Server: Queue Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 8b4d45c8-2b5d-46b8-8e14-90b0d902db80
Date: Wed, 17 Jun 2009 03:17:57 GMT
Content-Length: 0
```

In Listing 4-9, the first line represents the status code of the operation. The `x-ms-request-id` represents a unique request identifier that can be used for debugging or tracing.

Figure 4-6 shows the working of the Create Queue operation in the Windows Azure Storage Operations application.

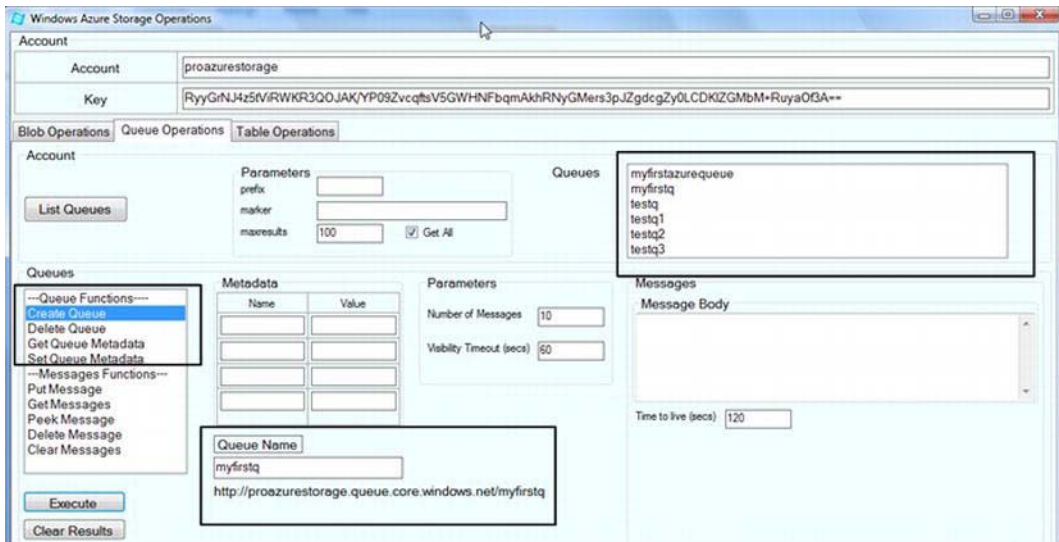


Figure 4-6. Create Queue from Windows Azure Storage Operations.exe

As shown in Figure 4-6, to create a queue, you need to do the following:

1. Go to the Queue Operations tab.
2. Enter a queue name (such as myfirstazurequeue) in the Queue Name text field.
3. Select the Create Queue operation from the Operations list box.
4. Click the Execute button. After the queue is created, the queues list box in the Account section is refreshed with the newly created queue name in it.

To help you understand the programming model of the Create Queue operation, open the Visual Studio Solution Chapter4.sln from the Chapter 4 source directory. The WASTorageHelper class in the ProAzureCommonLib contains a helper function called CreateQueue, as shown in Listing 4-10.

Listing 4-10. Create Queue Method in the WASTorageHelper Class

```
public bool CreateQueue(string queueName)
{
    CloudQueue q = QueueClient.GetQueueReference(queueName);
    return q.CreateIfNotExist();
}
```

The CreateQueue() method calls the GetQueueReference() method to get a reference to the CloudQueue object. The CloudQueue object is a local instance of the Queue object and may not represent a queue that already exists. This instance doesn't create a queue when you instantiate it. To create a queue, you have to call the CreateQueue() method on the CloudQueue object explicitly. The CloudQueue object creates the accurate URI and metadata headers for calling the Queue service.

Under the hood, the API uses `System.Net.HttpWebRequest` to send the REST message over HTTP. Upon success or failure of the operation, the Queue service returns an HTTP status code: HTTP/1.1 201 for success or HTTP/1.1 409 for conflict or failure. The `CreateQueue()` method translates the HTTP status code into true for success and false for failure or conflict. The Boolean value is passed all the way to the Windows Azure Storage Operations application as a return parameter of the `CreateQueue()` method.

Set Queue Metadata

Queues can contain name-value pairs of metadata values. You can store values like the time of creation, creator, last modified by user, and so on in the metadata fields of a queue. The size of the metadata can be 8KB per queue. The Set Queue Metadata operation sets the metadata of a queue independently. The URI for the Set Queue Metadata operation is of the format `account name>.queue.core.windows.net/<queue name>?comp=metadata`. The Set Queue Metadata REST request looks like Listing 4-11.

Listing 4-11. Set Queue Metadata REST Request

```
PUT /myfirstazurequeue?comp=metadata&timeout=30 HTTP/1.1
x-ms-date: Wed, 17 Jun 2009 04:33:45 GMT
x-ms-meta-createdBy: tejaswi
x-ms-meta-creationDate: 6/16/2009
Authorization: SharedKey proazurestorage:spPPnadPYnH6AJguuYT9wP1GLXmCjn0I1S6W2+hzyMc=
Host: proazurestorage.queue.core.windows.net
Content-Length: 0
```

In Listing 4-11, the HTTP verb used is PUT, and the URI parameter is `?comp=metadata`. This parameter instructs the Queue service to set the queue metadata instead of creating the queue. The Create Queue operation doesn't have this parameter. The `x-ms-meta.[name]:[value]` entries represent the metadata name-value pairs you want to set on the queue.

■ **Caution** Set Queue Metadata operation replaces all the existing metadata of the queue. It doesn't update individual metadata entries. For example, if a queue has two metadata values `Creator` and `Creation-Time`, and you call Set Queue Metadata with only one metadata value `LastUpdatedBy`, then the `Creator` and `Creation-Time` values will be deleted and the queue will have only one metadata value: `LastUpdatedBy`. To avoid this side effect, always set all the metadata values again along with any new values you want to add to the queue's metadata.

Figure 4-7 illustrates how to execute the Set Queue Metadata operation in Windows Azure Storage Operations application.

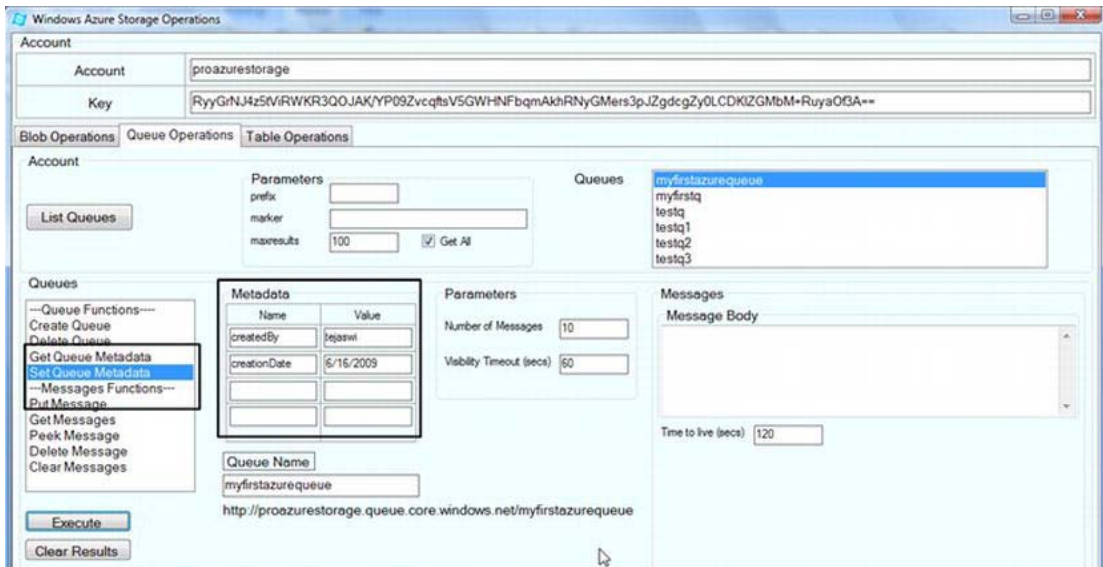


Figure 4-7. Set Queue Metadata in the Windows Azure Storage Operations application

As shown in Figure 4-7, to execute the Set Queue Metadata operation, you do the following:

1. Go to the Queue Operations tab.
2. In the Account section, click the List Queues button to get a list of queues in your account.
3. Select one of the queues from the list (such as myfirstazurequeue).
4. Make sure the Queue Name text box in the Queues section displays the name of the selected queue.
5. In the Queues section, select the Set Queue Metadata operation from list of queue operations.
6. In the Queues section, enter metadata name-value pairs in the Metadata section.
7. Click the Execute button to execute the operation.
8. To verify the success of the operation, click the Clear Results button in the Queues section, and re-select the queue from the Queues list in the Account section to retrieve the newly set metadata values.

To help you understand the programming model of the Set Queue Metadata operation, open the Visual Studio Solution Chapter4.sln from the Chapter 4 source directory. The WASTorageHelper class in the ProAzureCommonLib contains a helper function called SetQueueMetadata(), as shown in Listing 4-12.

Listing 4-12. SetQueueMetadata Method in the WASTorageHelper Class

```
public void SetQueueMetadata(string queueName, NameValueCollection metadata)
{
    CloudQueue queue = GetQueue(queueName);

    queue.Metadata.Clear();
    queue.Metadata.Add(metadata);
    queue.SetMetadata();
}
```

In Listing 4-12, the System.Collections.Specialized.NameValueCollection represents the metadata name-value pairs. The queue name is used to create a local instance of the CloudQueue object. The code then clears all the metadata and adds new metadata to the queue. Note that you need to call SetMetadata() operation to actually commit the metadata changes.

The SetMetadata() method creates the REST message and sends it synchronously to the Windows Azure Queue service to set the queue metadata values. It uses the System.Net.HttpWebRequest to send the REST message over HTTP. Upon success or failure of the operation, the Windows Azure Queue service returns an HTTP status code: HTTP/1.1 200 for success or HTTP/1.1 204 (No content)

Message Operations

Messages support several operations, as listed in Table 4-7.

Table 4-7. Messages Operations

Operation	Description
Put Message	En-queues a message at the end of the specified queue. The message can't be more than 8KB in size.
Get Messages	Dequeues one or more messages from the front of the specified queue. The maximum number of messages that can be retrieved in a single call is 32. The messages received are marked invisible until the visibilitytimeout property of the message expires. The default visibilitytimeout is 30 seconds, and the maximum value is two hours.
Peek Messages	Reads one or more messages from the front of the specified queue. This method doesn't alter the visibilitytimeout property of a message, so the messages are visible to other applications at the same time.
Delete Message	Deletes the specified message from the queue. The Queue service marks the message for deletion, and the message is deleted during the next garbage-collection cycle. The delete operation requires the MessageId and PopReceipt of the message to be passed to the

Queue service.

Clear Messages Deletes all messages from the specified queue.

Table 4-8 lists some of the important characteristics of the message operations.

Table 4-8. *Message Operations Characteristics*

Operation	HTTP Verb	Cloud URI	Development Storage URI	HTTP Version	Permissions
Put Message	POST	http://<account name>.queue.core.windows.net/<queue name>/messages	http://127.0.0.1:10001/<devstorageaccount>/<queue name>/messages	HTTP/1.1	Only the account owner can call this operation.
Get Messages	GET	http://<account name>.queue.core.windows.net/<queue name>/messages	http://127.0.0.1:10001/<devstorageaccount>/<queue name>/messages	HTTP/1.1	Only the account owner can call this operation.
Peek Messages	GET	http://<account name>.queue.core.windows.net/<queue name>/messages?peekonly=true	http://127.0.0.1:10001/<devstorageaccount>/<queue name>/messages?peekonly=true	HTTP/1.1	Only the account owner can call this operation.
Delete Message	DELETE	http://<account name>.queue.core.windows.net/<queue name>/<messageid>?popreceipt=[pop receipt value]	http://127.0.0.1:10001/<devstorageaccount>/<queue name>/<messageid>?popreceipt=[pop receipt value]	HTTP/1.1	Only the account owner can call this operation.
Clear Messages	DELETE	http://<account name>.queue.core.windows.net/<queue name>/messages	http://127.0.0.1:10000/<devstorageaccount>/<queue name>/messages	HTTP/1.1	Only the account owner can call this operation.

The <account name> is the storage account name in the cloud, and the <devstorageaccount> is the development storage account. The <queue name> is the name of the queue in which messages are stored. The following sections discuss some of the operations from Table 4-8 in detail. Even though the operations are different, the programming concepts behind them are similar. To keep the book at a conceptual level, I discuss just the Put Message and Get Messages operations. By studying these two operations in detail, you can understand the programming concepts behind all the message operations. The Windows Azure Storage Operations application included with this chapter's source code contains implementations of most of the message operations.

Put Message

The Put Message operation en-queues (puts) a message at the end of the queue. The URI of a Put Message operation is of the format `account name>.queue.core.windows.net/<queue name>/messages`. You can send a message with size up to 8KB. To send larger files, you can save the message as a blob and send the URI of the blob to the queue. The body of the message while sending can be text or binary, but it should support inclusion in an XML body with UTF-8 encoding. This is because a message received from the queue is always returned in Base64-encoded format within an XML response body. You see this in the Get Messages operation. The URI for the Put Message operation supports an additional optional parameter, listed in Table 4-9.

Table 4-9. Put Message URI Parameter

Parameter	Description	Example
messagettl	This is an integer value of seconds representing the time-to-live for a message in the queue before it's retrieved or deleted. The default and the maximum value for messagettl is seven days, after which the message is garbage-collected.	<code>account name>.queue.core.windows.net/<queue name>/messages?messagettl=60</code>

The Put Message REST request looks like Listing 4-13.

Listing 4-13. Put Message REST Request

```
POST /myfirstazurequeue/messages?messagettl=120&timeout=30 HTTP/1.1
x-ms-date: Thu, 18 Jun 2009 05:52:00 GMT
Authorization: SharedKey proazurestorage:Ahv5yhR9x0rHiMTnq3fBcaBKL8KeUFQ3r
Host: proazurestorage.queue.core.windows.net
Content-Length: 84
Expect: 100-continue
```

```
<QueueMessage>
<MessageText>bXlmaXJzdGF6dXJlbWVzc2FnZQ==</MessageText>
</QueueMessage>
```

In Listing 4-13, a string message “myfirstazuremessage” is sent to the queue named myfirstazurequeue. The time-to-live seconds for the message is 120, which means if the message isn’t received or deleted by an application within 120 seconds in the queue, the message will be marked for deletion and won’t be visible to any applications. The request body consists of the message content wrapped in the `<QueueMessage>` element. Note that the content of the message within the `<MessageText />` element is in Base64-encoded format. Listing 4-14 shows the response from the Queue service.

Listing 4-14. Put Message REST Response

```

HTTP/1.1 201 Created
Server: Queue Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: e724cc82-3d21-4253-9317-3b3964374be7
Date: Thu, 18 Jun 2009 05:53:32 GMT
Content-Length: 0

```

As shown in Listing 4-14, the Queue service responds with an HTTP/1.1 201 Created status code for a successful Put Message operation. Figure 4-8 shows the working of the Put Message operation in the Windows Azure Storage Operations application.

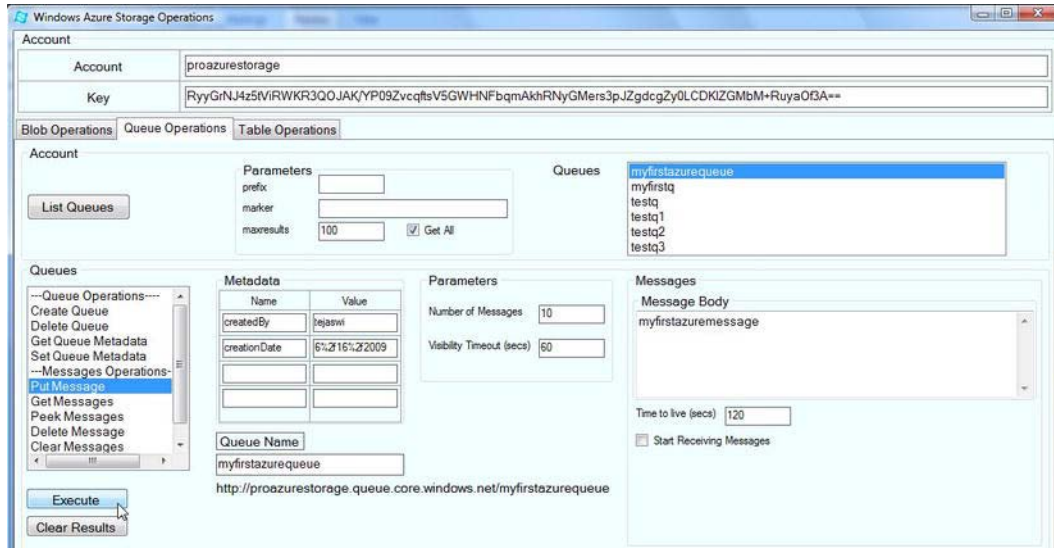


Figure 4-8. Put Message in Windows Azure Storage Operations.exe

As illustrated in Figure 4-8, you can send a text message using the Windows Azure Storage Operations application. The steps for sending a message to a queue are as follows:

1. Create a new queue (called myfirstazurequeue).
2. Select the new queue from the Queues List Box in the Accounts section.
3. Add some text to the Message Body text box in the Queues section.
4. Select the Put Message operation from the Operations text box.
5. Make sure the Queue Name text box is populated with the selected queue name.
6. Optionally, you can specify the time-to-live in the “Time to live (secs)” text box.

- 7. Click the Execute button to execute the Put Message operation.

To help you understand the programming model of the Put Message operation, open the Visual Studio Solution Chapter4.sln from the Chapter 4 source directory. The WindowsAzureStorage.cs file in the Windows Azure Storage Operations project consists of a PutMessage() method, as shown in Listing 4-15.

Listing 4-15. PutMessage() Method in WStorageHelper.cs

```
public void AddMessage(string queueName, CloudQueueMessage queueMessage)
{
    CloudQueue q = QueueClient.GetQueueReference(queueName);
    q.AddMessage(queueMessage);
}

//Calling the method
int ttlsecs=300;

StorageHelper.AddMessage(txtQueueName.Text, new CloudQueueMessage(messageBody),
ttlsecs);
```

The AddMessage() method of the CloudQueue object creates the REST message request and sends it synchronously to the Queue service. It uses the System.Net.HttpWebRequest to send the REST message over HTTP. Upon the success of the operation, the Queue service returns an HTTP status code: HTTP/1.1 201 Created.

Get Messages

In the previous section, you learned to send messages to queues in the Queue service. In this section, you learn to retrieve these messages using the Get Messages operation. The URI for the Get Messages operation is of the format account name>.queue.core.windows.net/<queue name>/messages. The URI for the Get Messages operation supports additional optional parameters, as listed in Table 4-10.

Table 4-10. Get Message URI Parameters

Parameter	Description	Example
numofmessages	An integer value specifying the total number of messages you want retrieved. You can retrieve a maximum of 32 messages in a single call. By default, the operation retrieves only one message at a time.	account name>.queue.core.windows.net/ <queue name>/messages?numofmessages= 10
visibilitytimeout	An integer value representing the visibility of the message in seconds after it's received by a receiving application. The default visibilitytimeout value is 30 seconds, which means that after a	account name>.queue.core.windows.net/ <queue name>/messages?visibilitytime

message is received, it will remain invisible to other applications for 30 seconds, unless it's deleted by the receiving application. The maximum visibilitytimeout value is two hours.

out=60

Listing 4-16 shows the REST API request for the Get Messages operation.

Listing 4-16. *Get Messages REST Request*

```
GET /myfirstazurequeue/messages?numofmessages=10&visibilitytimeout=60&timeout=30_
HTTP/1.1
x-ms-date: Thu, 18 Jun 2009 05:34:13 GMT
Authorization: SharedKey proazurestorage:qB9P717GTC6nd6rX4Ed16r6Qkx02QwJxLcr
Host: proazurestorage.queue.core.windows.net
```

In Listing 4-16, the URI points to the myfirstazurequeue queue. numofmessages=10 instructs the Queue service to retrieve only 10 messages. visibilitytimeout=60 instructs the Queue service to make the retrieved messages invisible to other applications for 60 seconds, unless the receiving application deletes them. Listing 4-17 shows the REST API response from the Queue service for the Get Messages operation.

Listing 4-17. *Get Messages REST Response*

```
HTTP/1.1 200 OK
Content-Type: application/xml
Server: Queue Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: c10542ae-fa9e-45fd-b036-3f0b77ed611e
Date: Thu, 18 Jun 2009 05:35:43 GMT
Content-Length: 3900

<?xml version="1.0" encoding="utf-8"?>
<QueueMessagesList>
  <QueueMessage>
    <MessageId>ba16723c-8b4c-48dd-9d80-d5d2731bcbd8</MessageId>
    <InsertionTime>Thu, 18 Jun 2009 05:36:43 GMT</InsertionTime>
    <ExpirationTime>Thu, 18 Jun 2009 05:37:28 GMT</ExpirationTime>
    <PopReceipt>AgAAAAEAAAAAAAAAIBeHw9bvYQE=</PopReceipt>
    <TimeNextVisible>Thu, 18 Jun 2009 05:36:43 GMT</TimeNextVisible>
    <MessageText>bXlmaXJzdGF6dXJlbWVzc2FnZQ==</MessageText>
  </QueueMessage>
  <QueueMessage>
    <MessageId>c0d92c72-2f9f-4c14-a177-7cf988c2532d</MessageId>
    <InsertionTime>Thu, 18 Jun 2009 05:36:43 GMT</InsertionTime>
    <ExpirationTime>Thu, 18 Jun 2009 05:37:28 GMT</ExpirationTime>
    <PopReceipt>AgAAAAEAAAAAAAAAIBeHw9bvYQE=</PopReceipt>
    <TimeNextVisible>Thu, 18 Jun 2009 05:36:43 GMT</TimeNextVisible>
    <MessageText>bXlmaXJzdGF6dXJlbWVzc2FnZQ==</MessageText>
  </QueueMessage>
  <QueueMessage>
    <MessageId>f3ae9ccd-b97c-4bae-bc22-744cadd2c9c0</MessageId>
```

```

<InsertionTime>Thu, 18 Jun 2009 05:36:43 GMT</InsertionTime>
<ExpirationTime>Thu, 18 Jun 2009 05:37:28 GMT</ExpirationTime>
<PopReceipt>AgAAAAEAAAAAAAAAIBeHw9bvYQE=</PopReceipt>
<TimeNextVisible>Thu, 18 Jun 2009 05:36:43 GMT</TimeNextVisible>
<MessageText>bXlmaXJzdGF6dXJlbWVzc2FnZQ==</MessageText>
</QueueMessage>
</QueueMessagesList>

```

Listing 4-17 shows the HTTP header and body of the Get Messages operation response. For the sake of brevity, only three messages are shown. The HTTP response body consists of a list of messages in XML format. Every `<QueueMessage />` element represents a message. When you retrieve a message, the `MessageId` and the `PopReceipt` properties of the message are important for deletion purposes. The recommended pattern is to receive the message, process it, and then delete it before it becomes visible to other applications when the `visibilitytimeout` period expires. The `TimeNextVisible` value specifies the expiration time of the `visibilitytimeout` period. The `ExpirationTime` specifies the time when the message will be marked for deletion if not retrieved and/or deleted by a receiving application. This value was set when the message was sent to the queue. Figure 4-9 shows the working of the Get Messages operation in the `Windows Azure Storage Operations.exe` application.

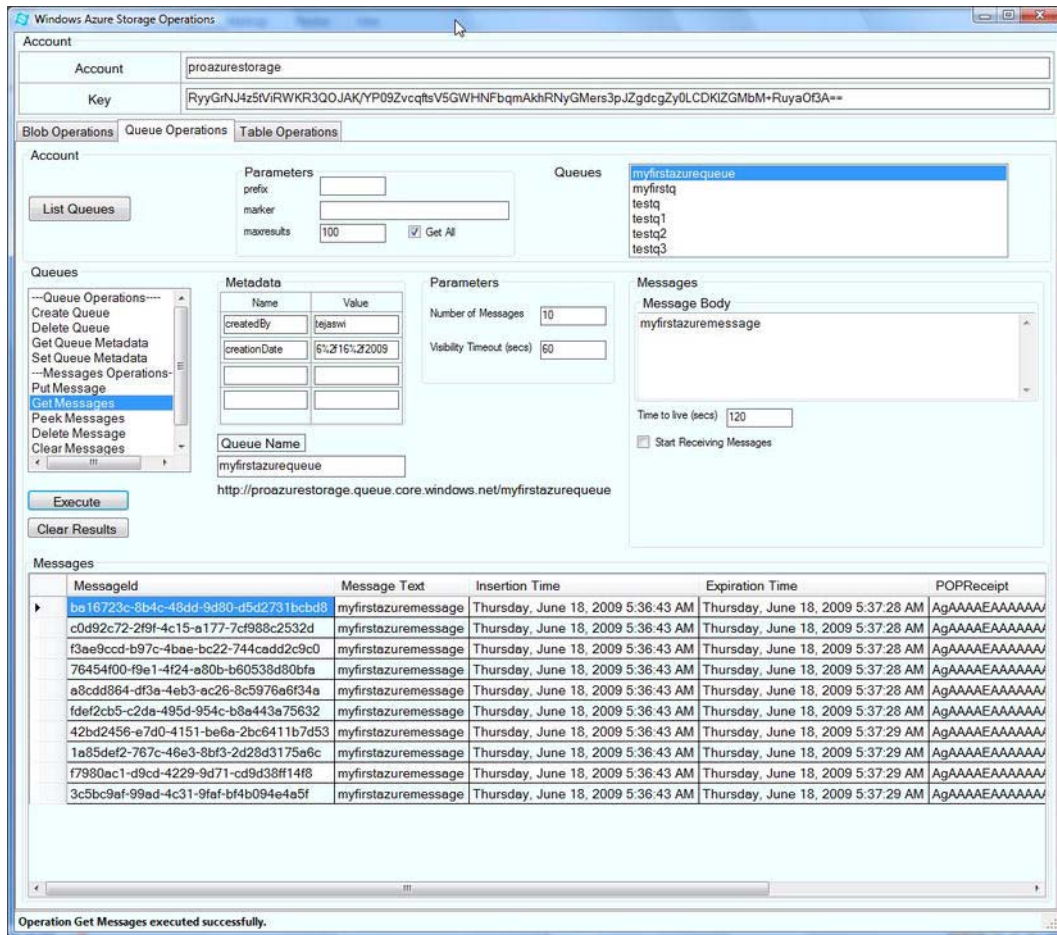


Figure 4-9. Get Messages in the Windows Azure Storage Operations application

As illustrated in Figure 4-9, you can use the Get Messages operation for a queue using the Windows Azure Storage Operations application. The steps for retrieving messages are as follows:

1. Select a queue (such as myfirstazurequeue) that already contains some messages.
2. In the Queues section, select the Get Messages operation.

3. Click the Execute button to get a list of messages from the selected queue.
4. Optionally, you can specify the Number of Messages and Visibility Timeout in the Parameters section.

The retrieved messages are populated in the DataGridView control in the Messages section. Each message is represented by a row in the DataGridView control. The control displays all the properties of the retrieved messages. To delete a message, select a row in the DataGridView and press the Delete button on your keyboard.

To help you understand the programming model of the Get Messages operation, open the Visual Studio Solution Chapter4.sln from the Chapter 4 source directory. The `WASStorageHelper.cs` file in the Windows Azure Storage Operations project consists of two overloaded `GetMessages()` methods, as shown in Listing 4-18.

Listing 4-18. *GetMessages() Method in WindowsAzureStorage.cs*

```
public IEnumerable<CloudQueueMessage> GetMessages(string queueName, int numberOfMessages,
int visibilityTimeoutInSecs)
{
    CloudQueue q = QueueClient.GetQueueReference(queueName);
    return q.GetMessages(numberOfMessages, new TimeSpan(0, 0,
visibilityTimeoutInSecs));
}

public IEnumerable<CloudQueueMessage> GetMessages(string queueName, int
numberOfMessages, TimeSpan timeout)
{
    CloudQueue q = QueueClient.GetQueueReference(queueName);
    return q.GetMessages(numberOfMessages, timeout);
}
```

The `numofmessages` parameter represents the number of messages to retrieve. The `visibility timeout` represents the length of time these retrieved messages will be invisible to other clients. If you don't delete these messages in the `visibilitytimeout` period specified, these messages will be read by other clients or the same client when it tries to read again. The `visibility timeout` parameter is used in making sure the message is processed at least once when multiple clients are accessing the same queue. If the processing of a message fails, then it will be automatically read by other clients. If the processing of the message succeeds, it must be deleted by the client processing the message. In order to avoid processing of the message multiple times, you need to make sure the `visibilitytimeout` period is longer than the message processing time.

Unlike MSMQ, the `Microsoft.WindowsAzure.StorageClient` API does not provide any queue listener events. But, the event objects in .NET Framework enables you to build your own. In the `ProAzureCommonLib` project, I have created an event class `MessageReceivedEventArgs`, an event handler delegate `MessageReceivedEventHandler` and a listener class `QueueListener` that defines the `MessageReceived` event. See Listing 4-10.

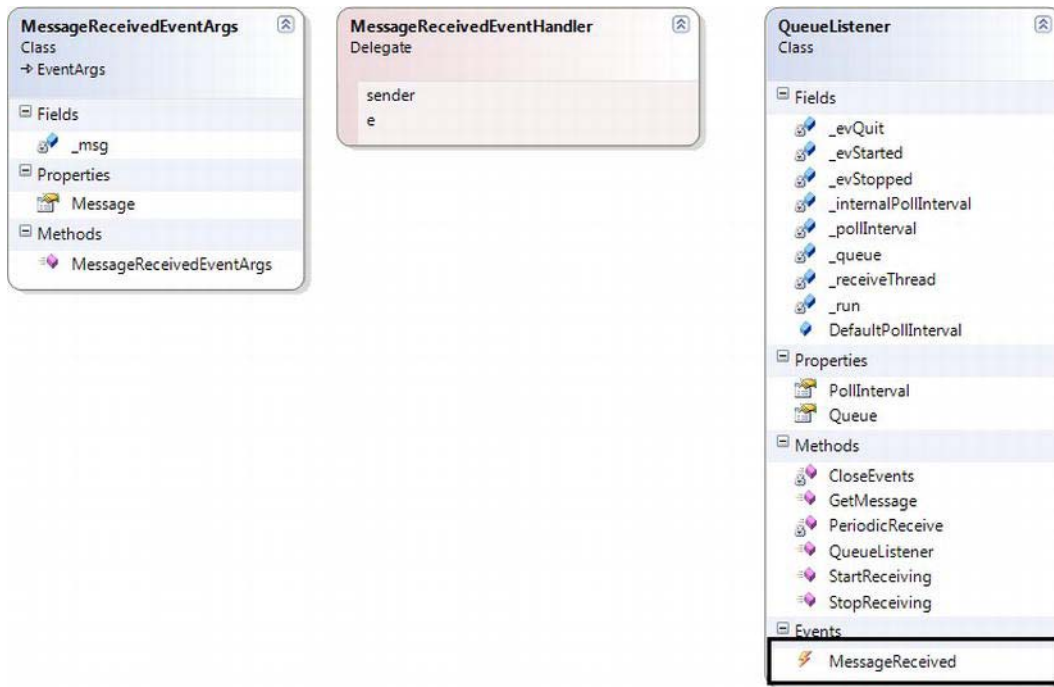


Figure 4-10. Custom *QueueListener*

Figure 4-10 illustrates class diagram for the event handler and *QueueListener*. The client class can implement the *MessageReceived* event to receive messages from the Queue service.

■ **Note** The even-driven model is a purely client-side implementation for ease of client programming. In the background, the event is fired periodically and calls the same *Get Messages* operation discussed in earlier section. The REST API for the Queue service doesn't offer events.

Listing 4-19 shows the usage of the *QueueListener* class for receiving a *MessageReceived* event whenever a new message arrives in the specified queue.

Listing 4-19. *Listening for Queue messages*

```
listener.MessageReceived -= new MessageReceivedEventHandler(listener_MessageReceived);
listener.PollInterval = 10000;
listener.StartReceiving();

void listener_MessageReceived(object sender, MessageReceivedEventArgs e)
```

```

{
    //Cast the message
    CloudQueueMessage m = e.Message as CloudQueueMessage;
    //Process the message
}

```

■ **Tip** When failed messages remain in the queue and are not processed by any message receivers, they remain in the queue till they expire. These messages are called poison messages or orphan messages. Poison messages can cost you money in the cloud or simply interfere with your regular message processing producing erroneous results. The Queue service does not explicitly track poison messages because it does not know whether it is poison or not. Therefore, your application needs to keep track of poison messages in the queue listener and delete them after processing has failed. The `CloudQueueMessage` class has a property named `DequeueCount` that gives you the number of times a message has been dequeued. You can use this property to identify poison messages in your queue listener and delete them immediately.

Asynchronous API

Until now, I have covered only synchronous methods for calling Queue service. In a real-world application, I recommend using asynchronous API instead of synchronous because in asynchronous method invocations, you are not blocking the calling thread and therefore the chances of getting a deadlock are limited. Especially in scenarios where the managed API (in this case, the Storage Client API) is making asynchronous calls to the service. The Storage Client API makes asynchronous REST calls to the Windows Azure Queue service and waits on the same thread for the response. If your synchronous call is waiting for the call to return on a thread and the asynchronous call is waiting on the `ThreadPool` to release a thread, there is a deadlock because your synchronous call will not return until the REST asynchronous call from within the API returns and the REST asynchronous call will not return because all the threads in the `ThreadPool` are exhausted.

As a workaround to this issue, and a best practice anyways, I recommend you to use asynchronous methods in the Storage Client API wherever possible. In stateless web applications, it involves a bit more work, because the request thread is synchronous, but the efforts in building asynchronous calling mechanisms in such applications will definitely pay off in terms of scalability. The Storage Client API for Queue Service consists of asynchronous methods for most of the operations. Listing 4-20 shows a pattern for invoking the asynchronous methods `BeginAddMessage()` and `BeginGetMessage()`. You can use the same pattern for invoking all the asynchronous methods in the Storage Client API, including Blob and Table storage.

Listing 4-20. Asynchronous Method Invocation Pattern

```

public void AddMessageAsync(string queueName, CloudQueueMessage queueMessage, int
ttlsecs)
{
    CloudQueue q = QueueClient.GetQueueReference(queueName);

```

```

        using (System.Threading.ManualResetEvent evt = new
System.Threading.ManualResetEvent(false))
        {
            q.BeginAddMessage(queueMessage, TimeSpan.FromSeconds(ttlsecs), new
AsyncCallback(result =>
            {
                var qc = result.AsyncState as CloudQueue;
                qc.EndAddMessage(result);
                evt.Set();
            }

            ), q);

            evt.WaitOne();
        }
    }

    public IEnumerable<CloudQueueMessage> GetMessagesAsync(string queueName, int
numberOfMessages, int visibilityTimeoutInSecs)
    {
        CloudQueue q = QueueClient.GetQueueReference(queueName);
        IEnumerable<CloudQueueMessage> ret = null;

        using (System.Threading.ManualResetEvent evt = new
System.Threading.ManualResetEvent(false))
        {
            q.BeginGetMessages(numberofMessages,
TimeSpan.FromSeconds(visibilityTimeoutInSecs), new AsyncCallback(result =>
            {
                var qc = result.AsyncState as CloudQueue;
                ret = qc.EndGetMessages(result);
                evt.Set();
            }

            ), q);

            evt.WaitOne();
        }

        return ret;
    }
}

```


In both the methods, note that I am creating a manual event that will be reset using the `evt.Set()` method after the operations is complete. I am not using the `IAsyncCallback` object's `WaitOne()` method to wait because when I use a lambda expression, the reset automatically happens even before the lambda expression code segment gets executed. If you are using a separate function to end the asynchronous method call instead of a lambda expression, you don't need to manually set the event. The above mentioned code pattern can be reused in all the asynchronous method calls in the `StorageClient` library.

■ **Note** You can find more information about this potential deadlock on the Windows Azure Storage Team blog (<http://blogs.msdn.com/b/windowsazurestorage/archive/2010/11/23/windows-azure-storage-client-library-potential-deadlock-when-using-synchronous-methods.aspx>).

Now that you understand Windows Azure Storage Queue service, let's look at some common scenarios in which Queues are used.

Queue Scenarios

In the previous sections, you saw the details of working with the Windows Azure Queue service. This section covers some of the basic application communication scenarios that can use the Windows Azure Queue service.

Scenario 1: Windows Azure Web and Worker Role Communications

Consider a scenario in which you're designing an ecommerce web application in Windows Azure with a Web role front end and several Worker roles for back-end processing work. The Web role instances continuously send purchase order information to the Worker roles for order processing. In this scenario, you can use the Windows Azure Queue service to queue purchase order messages for the Worker roles, as shown in Figure 4-11.

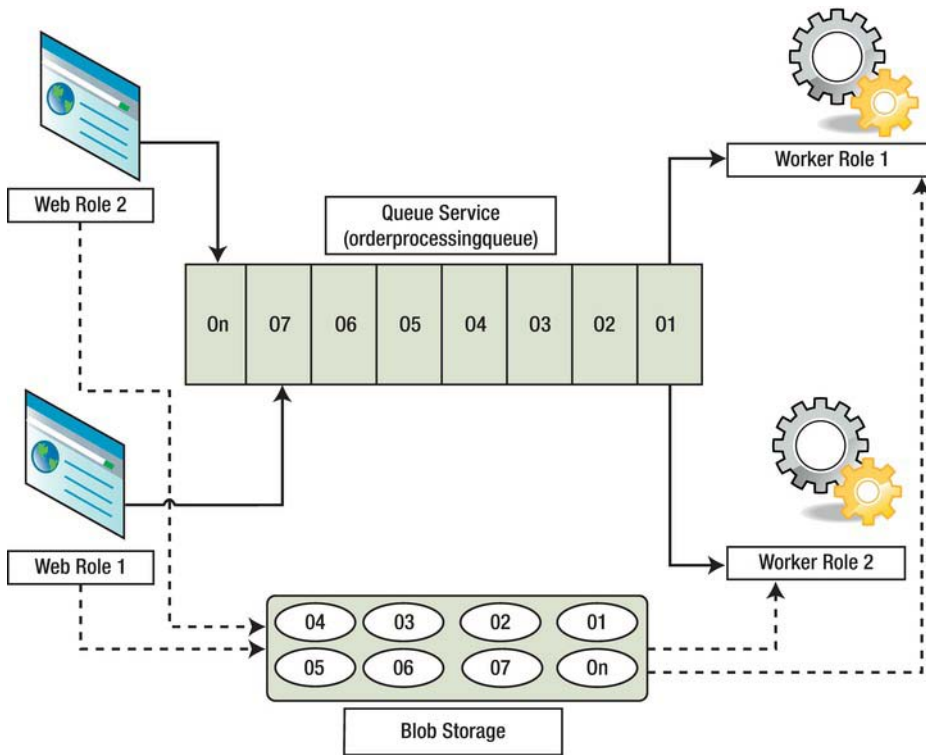


Figure 4-11. Web role/Worker role communication using the Queue service

In Figure 4-11, Web role instances 1 and 2 send orders to the order-processing queue. Worker Roles 1 and 2 dequeue the order messages and process the orders. Because not all orders have to be processed immediately, Worker roles can pick up from the queue only the orders that are ready to be processed. This way, you can create an effective message communication system between Web roles and Worker roles, taking advantage of the scalable and highly available Queue service infrastructure. If the order message size exceeds 8KB, you can store the message body in the Blob service and pass a link to the blob as a queue message, as shown in Figure 4-11. When the Worker role dequeues the message, it can retrieve the contents of the order from the Blob service.

Scenario 2: Worker Role Load Distribution

Continuing Scenario 1, depending on the volume of messages, you can either adjust the number of queues or the number of instances of Worker roles for processing orders. For example, if you identify during your testing phase that one Worker role can process only ten orders at a time, you can configure Worker roles to pick up only ten messages from the queue. If the number of messages in the queue keeps increasing beyond the number that Worker roles can process, you can create more instances of Worker roles on demand and increase the order-processing capacity. Similarly, if the queue is under-utilized, you can reduce the Worker role instances for processing orders.

In this scenario, the Queue service plays the role of capacity indicator. You can think of the queues in the Queue service as indicators of the system's processing capacity. You can also use this pattern to

process scientific calculations and perform business analysis. Figure 4-12 illustrates the Worker role load-distribution scenario.

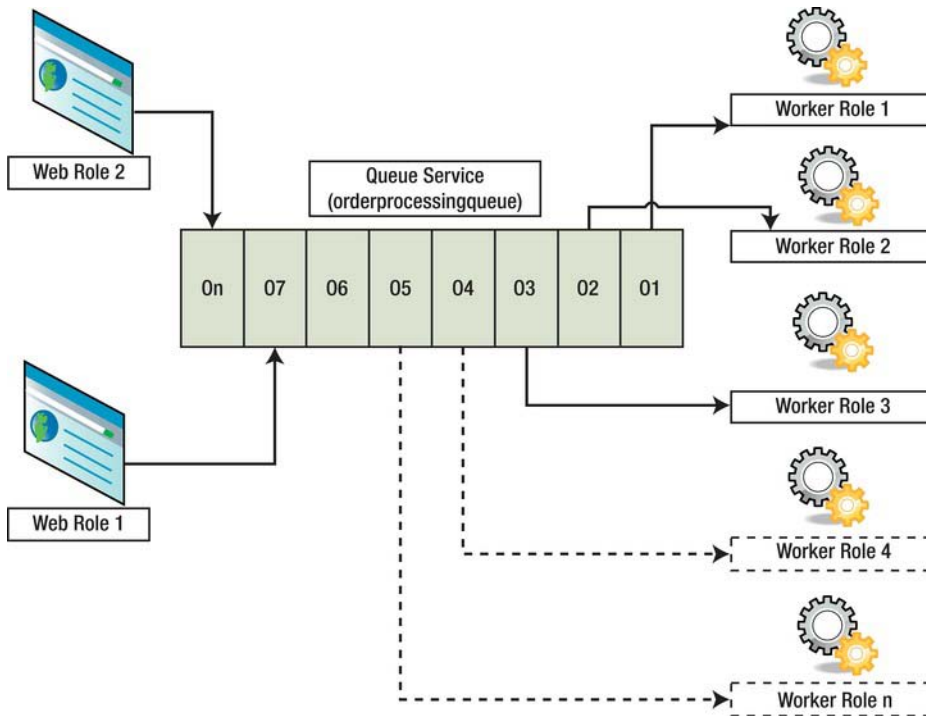


Figure 4-12. Worker role load distribution

In Figure 4-12, Worker Roles 1 through 3 can process average order loads. When the number of orders backs up into the queue, you can spawn more Worker roles (4 through *n*) depending on demand and the need for overall order-processing capacity.

Scenario 3: Interoperable Messaging

Large enterprises use applications from different vendors, and these applications seldom interoperate with each other. An enterprise may end up buying an expensive third-party tool that acts as the interoperability bridge between these applications. Instead, the enterprise could use the Queue service to send messages across the applications that don't interoperate with each other naturally. The Queue service exposes a REST API based on open standards. Any programming language or application capable of Internet programming can send and receive messages from the Windows Azure Queue service using the REST API. Figure 4-13 illustrates the use of the Queue service to interoperate between a Java-based Sales application and a .NET-based CRM application.

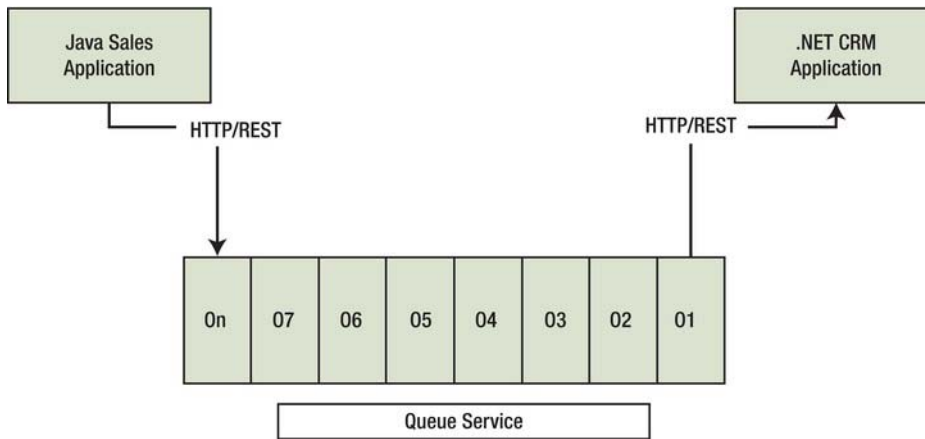


Figure 4-13. Interoperable messaging

Scenario 4: Guaranteed Processing

In Scenario 1, every order needs guaranteed processing. Any loss in orders can cause financial damage to the company. So, the Worker roles and the Queue service must make sure every order in the queue is processed. You can implement guaranteed processing using the following four simple principles:

- Set the `visibilitytimeout` parameter to a value large enough to last beyond the average processing time for the messages.
- Set the `visibilitytimeout` parameter to a value small enough to make the message visible if message processing fails in a consumer (Worker role) or a consumer crashes.
- Don't delete a message until it's processed completely.
- Design the message consumers (Worker roles) to be idempotent (that is, they should account for handling the same message multiple times without an adverse effect on the application's business logic).

Figure 4-14 illustrates guaranteed message processing in the context of the order-processing example discussed in Scenario 1.

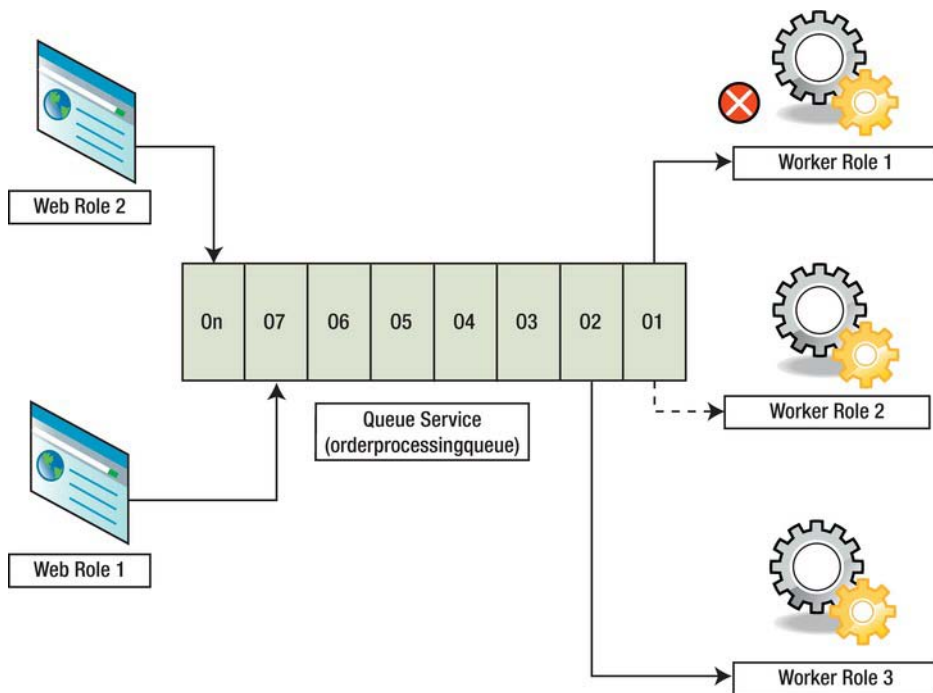


Figure 4-14. *Guaranteed processing*

In Figure 4-14, two Web roles create orders, and three Worker roles process orders. Consider the following steps:

1. Worker Role 1 reads order O1 for processing. Worker roles typically take 15 seconds to process an order. The visibilitytimeout for messages is set to 60 seconds.
2. Worker Role 1 starts processing order O1. At this point, O1 isn't visible to other Worker roles for 60 seconds.
3. Worker Role 1 crashes after 10 seconds.
4. After 60 seconds, O1 becomes visible again because Worker Role 1 wasn't able to delete it.
5. Worker Role 2 reads O1 and processes it.
6. After processing is complete, Worker Role 2 deletes O1 from the queue.

The important points to note here are that Worker Role 1 didn't delete the message from the queue before processing was complete, and the visibilitytimeout was set to an appropriate time window to exceed the processing time of an order. This pattern is commonly used in batch processing systems.

Summary

The Queue service provides a scalable and highly available store and delivery mechanism for exchanging messages across distributed applications. It provides reliable message delivery from message producers to message consumers.

Don't expect the performance of the Queue service to match your on-premises message brokers like MSMQ or ServiceBroker, because of its reliance on HTTP REST protocol. The Queue service exposes a REST API, making it easily accessible across multiple platforms and programming languages. In this chapter, you saw some of the important operations and scenarios for using the Queue service. The next chapter covers Windows Azure tables.

Bibliography

MSDN. (n.d.). *ADO.NET Data Services Specification*. Retrieved from MSDN Developer's Network:
<http://msdn.microsoft.com/en-us/library/cc668808.aspx>.

MSDN. (2009, May). *Windows Azure Blob — Programming Blob Storage*. Retrieved from MSDN:
<http://go.microsoft.com/fwlink/?LinkId=153400>.

MSDN. (2009, May). *Windows Azure Queue — Programming Queue Storage*. Retrieved from MSDN:
<http://go.microsoft.com/fwlink/?LinkId=153402>.

MSDN. (2009, May). *Windows Azure SDK*. Retrieved from MSDN: <http://msdn.microsoft.com/en-us/library/dd179367.aspx>.

MSDN. (2009, May). *Windows Azure Table — Programming Table Storage*. Retrieved from MSDN:
<http://go.microsoft.com/fwlink/?LinkId=153401>.

Windows Azure Storage

Part III – Tables

The Windows Azure Table service provides structured storage in the cloud. Windows Azure tables aren't relational database tables, but follow a simple yet highly flexible model of entities and properties. In the simplest of terms, tables contain entities, and entities have properties. The Table service is designed for massive scalability and availability, supporting billions of entities and terabytes of data. It's designed to support high volume, but smaller sized objects. For example, you can use the Table service to store user profiles and session information in high-volume Internet sites. But if you also want to store the photos of users, you should store the images in the Blob storage and save the link to the photo in Table service, but Table service has limitations on the size of each entity object. In this chapter, you will learn about the Windows Azure Table Storage service in detail. I have covered all the basics you need to start developing with the Table Storage API and, like other chapters, I have dedicated a section for Table Storage scenarios.

Table service provides you with the NoSQL storage option in the Windows Azure platform. NoSQL is important because it is said to scale efficiently, as compared to its relational counterparts, in certain scenarios. Table service is typically used in high-scale public facing applications such as social networking and blogging. The query patterns in these applications are much different than query patterns used in enterprise business applications where relational databases have better applicable scenarios.

There is no limit on the number of tables and entities you can create in a Table service. There is a limit of 100TB on the size of the tables in your account, which is capped at the storage account level. Similar to Blobs and Queues, the Table service also has a URL endpoint, as shown in Figure 5-1.

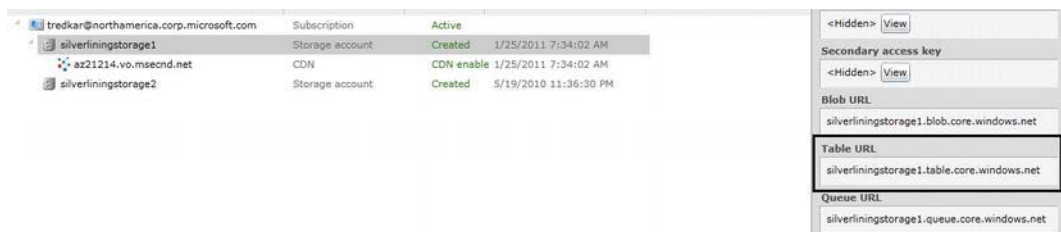


Figure 5-1. Table service endpoint URL

Table Service Architecture

The Table service architecture consists of a four-level hierarchical structure: Account, Table, Entity, and Properties, as shown in Figure 5-2.

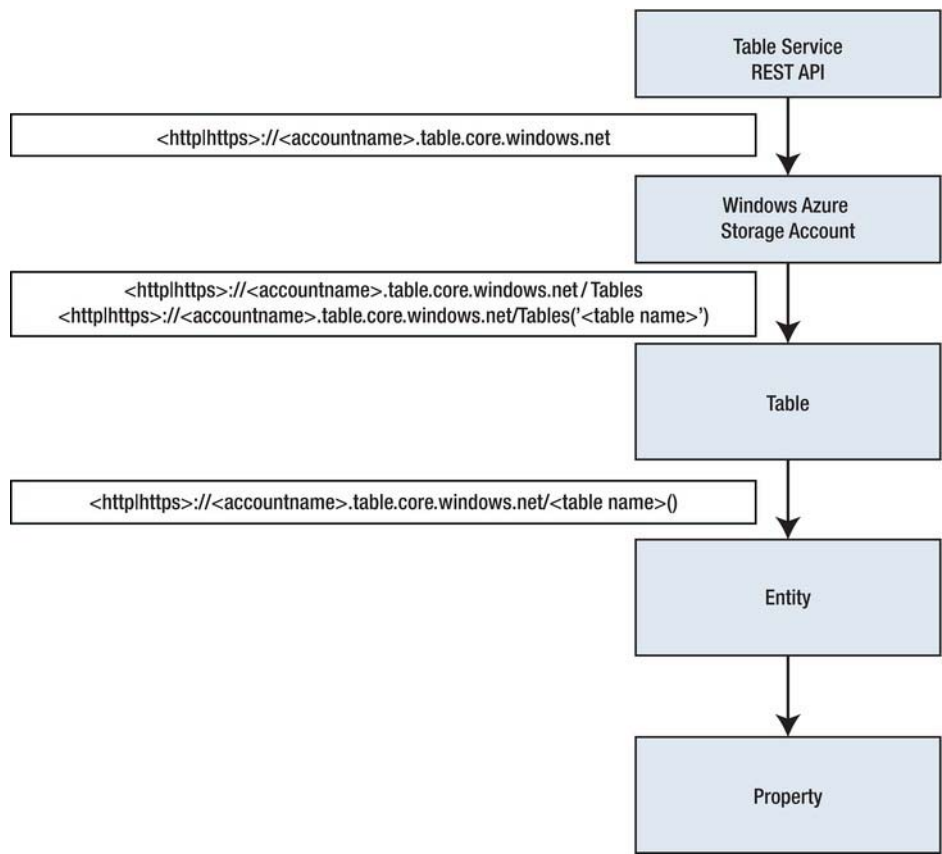


Figure 5-2. Table service architecture

Your Windows Azure storage account is the entry point to the Table service via the REST API.

Windows Azure Storage Account

The URI scheme for accessing the Table service via a storage account is

`<http|https>://<account name>.table.core.windows.net`

where `<account name>` is the globally unique name you created for your storage account. For example, the Table service for the storage account that I created in chapter 4 can be referenced as:

`<http|https>://proazurestorage.table.core.windows.net`

Table

A table is a container for storing data. Data is stored in tables as collection of entities. There can be any number of tables in an account in the Table service. A table stores entities and makes them available to applications via the REST API and .NET client-side libraries like ADO.NET Data Services and LINQ. The Table service supports only private access, which means you must have the account shared access key to access tables in the Table service.

You can access a table with the following URI:

```
<http|https>://<accountname>.table.core.windows.net/Tables('<table name>')
```

where *<table name>* is the name of the table you want to access. You can access all the tables in an account with the following URI:

```
<http|https>://<account name>.table.core.windows.net/Tables
```

For example, if you create a table named *userprofiles* in the *proazurestorage* account, you can reference it using the following URI:

```
<http|https>://proazurestorage.table.core.windows.net/Tables('userprofiles')
```

The naming constraints on a table are as follows:¹

- Table names must be valid DNS names.
- Table names must be unique within an account.
- Table names must contain only alphanumeric characters.
- Table names must begin with an alphabetical character.
- Table names are case sensitive.
- Table names can't be fewer than 3 or more than 63 characters in length.

If a table name or the URI violates the naming convention, the server returns an HTTP status code 400 (Bad request).

Entity

Entities are analogous to rows in a relational database table. There is no limit on the number of entities that can be stored in a table. You can retrieve all the entities in a table with the following URI:

```
<http|https>://<account name>.table.core.windows.net/<table name>()
```

where *<table name>* is the name of the table you want to access, and the parentheses instructs the Table service to retrieve the entities in the specified table.

¹Source: Windows Azure SDK documentation

Property

An entity consists of a set of name-value pairs called *properties*. Properties are analogous to columns in a relational database table. An entity must have three mandatory properties: PartitionKey, RowKey, and Timestamp. PartitionKey and RowKey are of string data type, and Timestamp is a read-only DateTime property maintained by the system. The combination of PartitionKey and RowKey uniquely identifies an entity. You must design PartitionKey and RowKey carefully as part of your table design exercise.

The Table service organizes data into several storage nodes based on the entities' PartitionKey property values. Entities with same PartitionKey are stored on a single storage node. A *partition* is a collection of entities with the same PartitionKey. A RowKey uniquely identifies an entity within a partition.

The Table service provides a single index in which entity records are sorted first by PartitionKey, and then by RowKey. All the entities in a single partition have the same PartitionKey, so you can safely assume that all the entities in a partition are lexically sorted by RowKey. Figure 5-3 illustrates the design of an example PartitionKey and RowKey for a table.

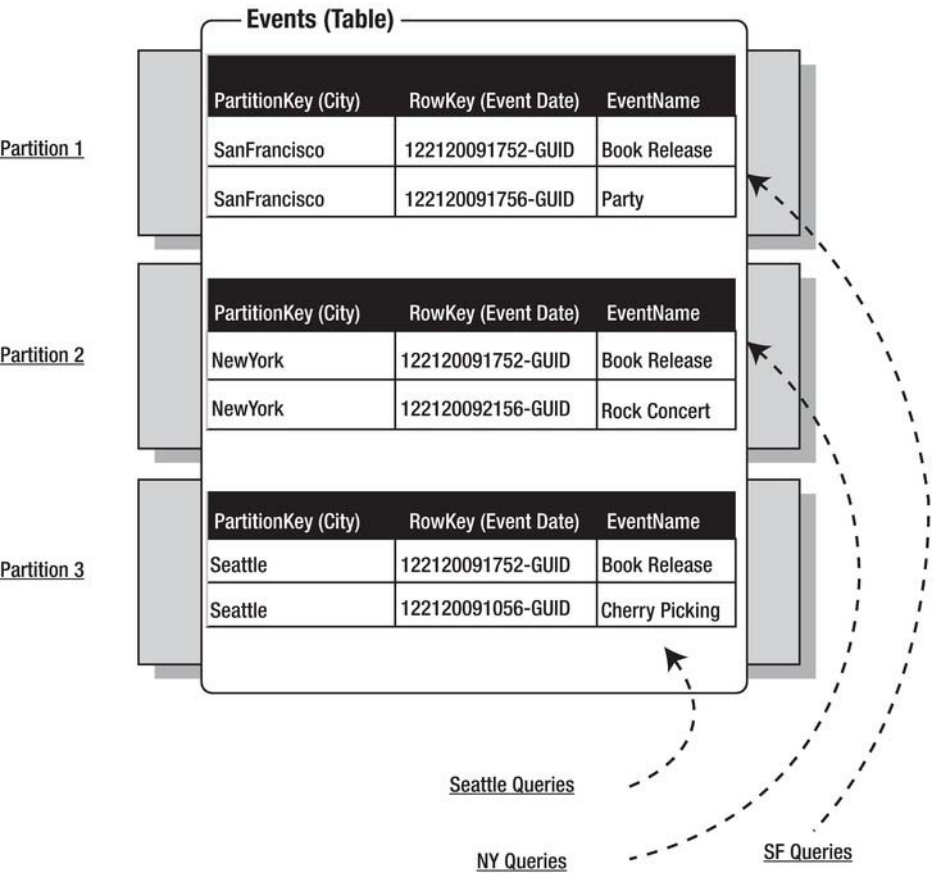


Figure 5-3. PartitionKey and RowKey

In Figure 5-3, imagine you're designing an event management web site. On your web site, the most dominant user query is, "Give me today's events in my city." So, the application queries the Events table for all the events on a particular day sorted with the most recent event at the top. The example in Figure 5-3 illustrates the Events table with its PartitionKey, RowKey, and EventName properties.

Because the most dominant query retrieves all the events from a city, as an architect of the system, you want the dominant query to execute on a single partition (or storage node) for maximum query performance. If a San Francisco user comes to the web site, the application retrieves all the San Francisco events from a single partition. As explained earlier, the Table service groups entities with the same PartitionKey on the same partition. To achieve the desired distribution of entities, you define City as the PartitionKey for the entity; doing so groups all the events in a particular city on a single partition, as shown in Figure 5-3.

The table spans three partitions: the events "Book Release" and "Party" in San Francisco are stored on Partition 1, and New York and Seattle events are stored on Partition 2 and Partition 3, respectively. The distribution of a table across multiple partitions is opaque to the application.

The next part of the dominant query involves sorting events by date and time. Remember that the RowKey uniquely identifies an entity within a partition, and the entities are sorted by RowKey within a partition. So, you want all the events in a city (or partition) sorted by date and time. To sort the entities in a partition by date and time, you define RowKey as a function of date and time. You can achieve this by subtracting (`DateTime.MaxValue.Ticks - EventTime.Ticks`), where `EventTime` is the date and time of the event.

The combination of PartitionKey and RowKey uniquely identifies an entity within a table, so you can't have a duplicate PartitionKey and RowKey pair. There can't be two events in the same city starting at the exact same date and time. If there are, then you have to design the Partition and Row keys to uniquely identify them. To create a unique PartitionKey and RowKey pair, you can append the RowKey with a GUID or any unique identifier of your choice. Because there can be only one PartitionKey and one RowKey in an entity, you must concatenate strings to create a PartitionKey and RowKey for every table design. In the future, when the Table service supports multiple indexes and RowKeys, the partitioning and sorting design will be more refined.

If the dominant query was, "Get me today's Book Releases (Event Type) across all the Cities," the PartitionKey would be a function of time and Event Type, because you would want all the events of the same type across all the cities partitioned together. But you would also have to consider the impact of the volume of data on the query. The previous examples assume the number of entities on each partition is low enough for the query to perform optimally. If there are millions of events in a particular city, you would further refine the PartitionKey to reduce the load on the query. In some cases, it may be worth duplicating the data for two or three dominant queries for boosting performance. Of course, you will pay the extra cost of storing data but the cost of storage is very low if you factor in performance impact of consumer facing applications. Typically, in mobile applications, lot of data is streamed to the mobile devices based on the user's location or geocode. When the user logs in, the geocode is sent to the server and based on the geocode, the data (e.g., local news) is streamed to the user's device.

So, would you store the data based on geocode as the partition key? It depends, because if the application supports adding multiple locations, then the user may add ten different locations from which he wants to receive news. Now, if you design the Partition Key as the geocode, then the query will have to scan ten partitions to compose the aggregated results. Therefore, I recommend designing the Partition Keys and Row Keys based on the application's performance requirements. You can also store duplicate data with a different Partition key and Row Key pair or cache the data in Windows Azure AppFabric Caching service.

While you're designing PartitionKeys, consider a tradeoff between scalability and performance. Depending on the capacity requirements and usage volume of your application, the PartitionKey may play an important role in scalability and performance. Having more partitions distributed over multiple storage nodes makes the table more scalable, whereas narrowing entities on a single partition may yield

better performance, assuming the number of entities on a partition is low enough for the query to perform optimally.

■ **Tip** Design your PartitionKeys and RowKeys in an iterative manner. Stress- and performance-test your design for every iteration. Then, choose an optimum PartitionKey that satisfies your application's performance and scalability requirements.

■ **Caution** The following characters are not allowed in PartitionKey and RowKey values:

The forward slash (/) character

The backslash (\) character

The number sign (#) character

The question mark (?) character

More information is available at: <http://msdn.microsoft.com/en-us/library/dd179338.aspx>

Table 5-1 lists the supported data types for property values and their Common Language Runtime (CLR) counterparts.

Table 5-1. Property Value Data Types

Data Type	Corresponding CLR Data Type
Binary	byte[]
Boolean	bool
DateTime	DateTime
Double	Double
Guid	Guid
Int32	int or Int32

Int64	long or Int64
String	string

The following are some of the characteristics of and constraints on entities and properties:

- Tables support flexible schema. This means a table can contain entities that have property values of different data types. For example, in a UserProfiles table, you can have an entity record representing a ZipCode property with an integer data type (“ZipCode”, 94582) and another entity record with a string data type for the same property (“ZipCode”, “CK45G”).
- An entity can contain at the most 255 properties (including the PartitionKey, RowKey, and Timestamp properties, which are mandatory).
- The total size of an entity including all the property names and values can’t exceed 1MB.
- Timestamp is a read-only value maintained by the system.
- PartitionKey and RowKey can’t exceed 1KB in size each.
- Property names can contain only alphanumeric characters and the underscore (_) character. The following characters aren’t supported in property names: backslash (\), forward slash (/), dash (-), number sign (#), and question mark (?).

REST API

The REST API for the Table service is available at the table and entity levels of the hierarchy. The Table service API is compatible with the ADO.NET Data Services REST API. The differences between the Table service API and the ADO.NET Data Services API are highlighted in the Table services API section of the Windows Azure SDK documentation.² In this section, you learn about the Table service REST API with specific examples. You also learn to interact with the Table service programmatically, using the .NET Client Library and the Storage Client libraries from the Windows Azure SDK. The REST API enables you to send HTTP messages to the Table service and its resources.

REST is an HTTP-based protocol; you specify the URI of the resource as well as the function you want to execute on the resource. Every REST call involves an HTTP request to the storage service and an HTTP response from the storage service. The programming examples in this section use the .NET Client library and/or Storage Client library to access the Table service. Both of them ultimately result in REST API calls to the Table service. You can choose to program the Table service directly using the REST API.

■ **Note** ADO.NET Data Services provides a REST API for accessing any data service on the Web. You can find the specification for the ADO.NET Data Services at <http://msdn.microsoft.com/en-us/library/cc668808.aspx>.

² See <http://msdn.microsoft.com/en-us/library/dd135720.aspx/>

Request

The following sections describe the Table service REST API's HTTP request components.

HTTP Verb

The HTTP verb represents the action or operation you can execute on the resource indicated in the URI. The Table service REST API supports the following verbs: GET, PUT, MERGE, POST, and DELETE. Each verb behaves differently when executed on a different resource.

Request URI

The request URI represents the URI of a resource you're interested in accessing or executing a function on. Example resources in the Table service include table and entity. An example URI to create a table named Events in an account named proazurestorage is

POST `http://proazurestorage.table.core.windows.net/Tables`

Note that unlike in the Blob and Queue services, the URI doesn't include the name of the table (Events). The request body includes the details of the table to be created. The HTTP verb POST instructs the service to create the table, and the request body points to the resource that needs to be created.

URI Parameters

Typically, URI parameters are the extra parameters you specify to fine-tune your operation execution. They may include the operation parameters or filter parameters for the results. In the Table service API, the URI parameters support the ADO.NET Data Service Framework query options \$filter and \$top, as described in the ADO.NET Data Service specification at <http://msdn.microsoft.com/en-us/library/cc668809.aspx>.

The \$filter parameter retrieves only the tables and entities that match the filter criteria specified in the URI. The following URI shows a sample usage of the \$filter parameter:

`http://proazurestorage.table.core.windows.net/ProAzureReader()?
$filter=PurchaseDate%20eq%20datetime'2009-05-20T00:00:00'`

ProAzureReader is the name of the table, and ProAzureReader() retrieves all the entities from the table. The URI further applies a filter "PurchaseDate eq datetime'2009-05-20T00:00:00'" for restricting the number of returned entities.

The \$top parameter retrieves only Top(*n*) number of tables or entities specified in the URI. The following URI shows a sample usage of the \$top parameter:

`http://proazurestorage.table.core.windows.net/ProAzureReader()?$top=3`

Again, ProAzureReader is the name of the table, and ProAzureReader() retrieves all the entities from the table. The \$top parameter instructs the Table service to retrieve only the top three entities from the table.

Request Headers

Request headers follow the standard HTTP 1.1 name-value pair format. Depending on the type of request, the header may contain security information, date time information, or instructions embedded

as name-value pairs. In the Storage Service REST API, the request header must include the authorization information and a Coordinated Universal Time (UTC) timestamp for the request. The timestamp can be in the form of either an HTTP/HTTPS Date header or an x-ms-Date header.

The authorization header format is as follows

```
Authorization="[SharedKey|SharedKeyLite] <Account Name>:<Signature>"
```

Where SharedKey|SharedKeyLite is the authentication scheme, <Account Name> is the storage service account name, and <Signature> is a Hash-based Message Authentication Code (HMAC) of the request computed using the SHA256 algorithm and then encoded by using Base64 encoding.

To create the signature, follow these steps:

1. Create the signature string for signing. The signature string for the Storage service request consists of the following format:

```
VERB\n
```

```
Content - MD5\n
```

```
Content - Type\n
```

```
Date\n
```

```
CanonicalizedHeaders
```

```
CanonicalizedResource
```

VERB is the uppercase HTTP verb such as GET, PUT, and so on. Content - MD5 is the MD5 hash of the request content. CanonicalizedHeaders is the portion of the signature string created using a series of steps described in the “Authentication Schemes” section of the Windows Azure SDK documentation at <http://msdn.microsoft.com/en-us/library/dd179428.aspx>. CanonicalizedResource is the storage service resource in the request URI. The CanonicalizedResource string is also constructed using a series of steps described in the “Authentication Schemes” section of the Windows Azure SDK documentation.

2. Use the System.Security.Cryptography.HMACSHA256.ComputeHash() method to compute the SHA256 HMAC encoded string.
3. Use the System.Convert.ToBase64String() method to convert the encoded signature to Base64 format.

Listing 5-1 shows an example request header for an entity GET operation.

Listing 5-1. Request Header

```
User-Agent: Microsoft ADO.NET Data Services
x-ms-date: Sat, 20 Jun 2009 22:42:54 GMT
x-ms-version: 2009-04-14
Authorization: SharedKeyLite
    proazurestorage:qWuBFkungfapSPIAFsrxeQ+j1uVRHyMUyEPiVOC832A=
Accept: application/atom+xml,application/xml
Accept-Charset: UTF-8
DataServiceVersion: 1.0;NetFx
MaxDataServiceVersion: 1.0;NetFx
```

Host: proazurestorage.table.core.windows.net

In Listing 5-1, the request header consists of x-ms-date, x-ms-version, and Authorization values. The x-ms-date represents the UTC timestamp, and the x-ms-version specifies the version of the storage service API you're using. The header also specifies the version of the ADO.NET Data Service API. The Authorization SharedKey header value is used by the Storage service to authenticate and authorize the caller.

The Table service REST API also supports the HTTP 1.1 If-Match conditional header. The If-Match conditional header is a mandatory header sent by the ADO.NET Data Service API. For update, merge, and delete operations, the Table service compares the specified ETag value with the ETag value on the server. If they don't match, an HTTP 412 (PreCondition Failed) error is sent back in the response. You can force an unconditional update by specifying a wildcard (*) for the If-Match header in the request.

Request Body

The request body consists of the contents of the request operation. Some operations require a request body, and some don't. For example, the Create Table operation's request body consists of an ADO.NET entity in the form of an Atom feed, whereas the Query Table operation requires an empty request body. Atom is an application-level protocol for publishing and editing web resources³ defined by the Internet Engineering Task Force (IETF). You see other request body examples later in this chapter.

■ **Note** For more information about the Atom format in ADO.NET Data Services messages, visit the “Atom Format” section of the ADO.NET Data Services specification at <http://msdn.microsoft.com/en-us/library/cc668811.aspx>.

Response

The HTTP response of the Table service API typically includes the components described in the following sections.

Status Code

The status code is the HTTP status code that indicates the success or failure of the request. The most common status codes for the Table service API are 200 (OK), 201 (Created), 400 (BadRequest), 404 (NotFound), 409 (Conflict), and 412 (PreCondition Failed).

³ Source: ADO.NET Data Services Framework, <http://msdn.microsoft.com/en-us/library/cc668811.aspx>

Response Headers

The response headers include all the standard HTTP 1.1 headers plus any operation-specific headers returned by the Table service. The x-ms-request-id response header uniquely identifies a request. Listing 5-2 shows an example response header for a Query Entities operation.

Listing 5-2. Query Entities Response Header

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Transfer-Encoding: chunked
Content-Type: application/atom+xml; charset=utf-8
Server: Table Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: a1eccc1c-8c1f-4fca-8ca9-69850684e553
Date: Sat, 20 Jun 2009 22:43:45 GMT
```

Response Body

The response body consists of values returned by the operation. These values are specific to each operation. For example, the Query Entity operation returns an ADO.NET entity set, which is in the form of an Atom feed. Listing 5-3 shows an example of the response body for the following entity query:

```
GET http://proazurestorage.table.core.windows.net/ProAzureReader()?$
    filter=PartitionKey%20eq%20'06202009'
```

The response consists of two entities.

Listing 5-3. Query Entity Response Body

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xml:base=http://proazurestorage.table.core.windows.net/
xmlns:d=http://schemas.microsoft.com/ado/2007/08/dataservices
xmlns:m=http://schemas.microsoft.com/ado/2007/08/dataservices/metadata
xmlns="http://www.w3.org/2005/Atom">
  <title type="text">ProAzureReader</title>
  <id>http://proazurestorage.table.core.windows.net/ProAzureReader</id>
  <updated>2009-05-20T22:43:46Z</updated>
  <link rel="self" title="ProAzureReader" href="ProAzureReader" />
  <entry m:etag="W/&quot;datetime'2009-05-20T13%3A01%3A10.5846Z'&quot;;">
    <id>http://proazurestorage.table.core.windows.net/ProAzureReader
(PartitionKey='06202009',RowKey='12521567980278019999')</id>
    <title type="text"></title>
    <updated>2009-05-20T22:43:46Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="ProAzureReader"
href="ProAzureReader(PartitionKey='06202009',RowKey='12521567980278019999')"/>
    <category term="proazurestorage.ProAzureReader"
```

```

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <content type="application/xml">
    <m:properties>
      <d:PartitionKey>06202009</d:PartitionKey>
      <d:RowKey>12521567980278019999</d:RowKey>
      <d:Timestamp m:type="Edm.DateTime">2009-05-20T13:01:10.5846Z
</d:Timestamp>
      <d:City>mumbai</d:City>
      <d:Country>india</d:Country>
      <d:EntryDate m:type="Edm.DateTime">2009-05-20T12:59:32.198Z
</d:EntryDate>
      <d:Feedback>Good Book :). But don't write again.</d:Feedback>
      <d:PurchaseDate m:type="Edm.DateTime">2009-05-20T00:00:00Z
</d:PurchaseDate>
      <d:PurchaseLocation>web</d:PurchaseLocation>
      <d:PurchaseType>New</d:PurchaseType>
      <d:ReaderName>tredkar</d:ReaderName>
      <d:ReaderUrl></d:ReaderUrl>
      <d:State>maharashtra</d:State>
      <d:Zip>400028</d:Zip>
    </m:properties>
  </content>
</entry>

<entry m:etag="W/&quot;datetime'2009-05-20T11%3A40%3A24.834Z'&quot;;">
  <id>http://proazurestorage.table.core.windows.net/ProAzureReader
(PartitionKey='06202009',RowKey='12521568028370519999')</id>
  <title type="text"></title>
  <updated>2009-05-20T22:43:46Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="ProAzureReader"
href="ProAzureReader(PartitionKey='06202009',
RowKey='12521568028370519999')"/>
  <category term="proazurestorage.ProAzureReader"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <content type="application/xml">
    <m:properties>
      <d:PartitionKey>06202009</d:PartitionKey>
      <d:RowKey>12521568028370519999</d:RowKey>
      <d:Timestamp m:type="Edm.DateTime">2009-05-20T11:40:24.834Z</d:Timestamp>
      <d:City></d:City>
      <d:Country></d:Country>
      <d:EntryDate m:type="Edm.DateTime">2009-05-20T11:39:22.948Z</d:EntryDate>
      <d:Feedback>Good Book :). But don't write again.</d:Feedback>
      <d:PurchaseDate m:type="Edm.DateTime">2009-05-20T00:00:00Z
</d:PurchaseDate>
      <d:PurchaseLocation></d:PurchaseLocation>
      <d:PurchaseType>New</d:PurchaseType>
      <d:ReaderName></d:ReaderName>
      <d:ReaderUrl></d:ReaderUrl>

```

```

        <d:State></d:State>
        <d:Zip></d:Zip>
    </m:properties>
</content>
</entry>
</feed>

```

■ **Tip** To test the REST API, I recommend using the Fiddler Tool available at www.fiddler2.com/fiddler2/. In this book, I use this tool to trace client/server communications.

ADO.NET Data Services Library (.NET Client Library)

The Table service API provides a subset of the ADO.NET Data Service API, so you can use the ADO.NET Data Services client library to work with tables and entities in the Table service. The System.Data.Services.Client assembly consists of the ADO.NET Data Services and .NET Client library classes.

■ **Note** For more information about the Table service's support for the ADO.NET Data Services .NET Client library, visit the latest Table services API MSDN documentation at <http://msdn.microsoft.com/en-us/library/dd894032.aspx>.

You don't have to use the ADO.NET Data Services library to interact with tables and entities in the Table service. You may choose to work directly at the REST API level by constructing REST messages on your own.

■ **Note** For more information about the ADO.NET Data Services .NET client library, visit <http://msdn.microsoft.com/en-us/library/cc668789.aspx>.

If you're using .NET Client library, the Table service lets you use a subset of Language Integrated Queries (LINQ) to interact with tables and entities. For more information about LINQ support in the Table service, visit the "Summary of Table Service Functionality" (<http://msdn.microsoft.com/en-us/library/dd135720.aspx>) and "Writing LINQ Queries" (<http://msdn.microsoft.com/en-us/library/dd894039.aspx>) sections of the Table service API Windows Azure SDK documentation.

In this book, I have used some of the ADO.NET Data Services .NET client library constructs as an alternative to using the Table service's REST API directly.

Storage Client API

Even though the REST API and the operations in the REST API are easily readable, the API doesn't automatically create the client stubs like those created by WSDL-based web services. You have to create your own client API and stubs for REST API operations. This makes the client programming more complex and increases the barriers to entry for developers. To reduce this barrier, the Windows Azure SDK team has created the `Microsoft.WindowsAzure.StorageClient` library available in the Windows Azure SDK. This library can be used to invoke REST APIs of the Windows Azure Storage service. The `Microsoft.WindowsAzure.StorageClient` library abstracts the REST API by providing a closed-source interface and therefore sufficient for most of the applications.

In the following sections, I have covered the table storage APIs from `Microsoft.WindowsAzure.StorageClient`.

■ **Note** You don't have to use any of the `StorageClient` API to make REST calls to the Storage service; you can instead create your own client library. In order to keep the book conceptual, I have used the `StorageClient` API to interact with the Storage service throughout this book.

Windows Azure StorageClient Table API

The `Microsoft.WindowsAzure.StorageClient` namespace consists of classes representing the entire blob hierarchy. Figure 5-4 illustrates the core classes for programming Table service applications.

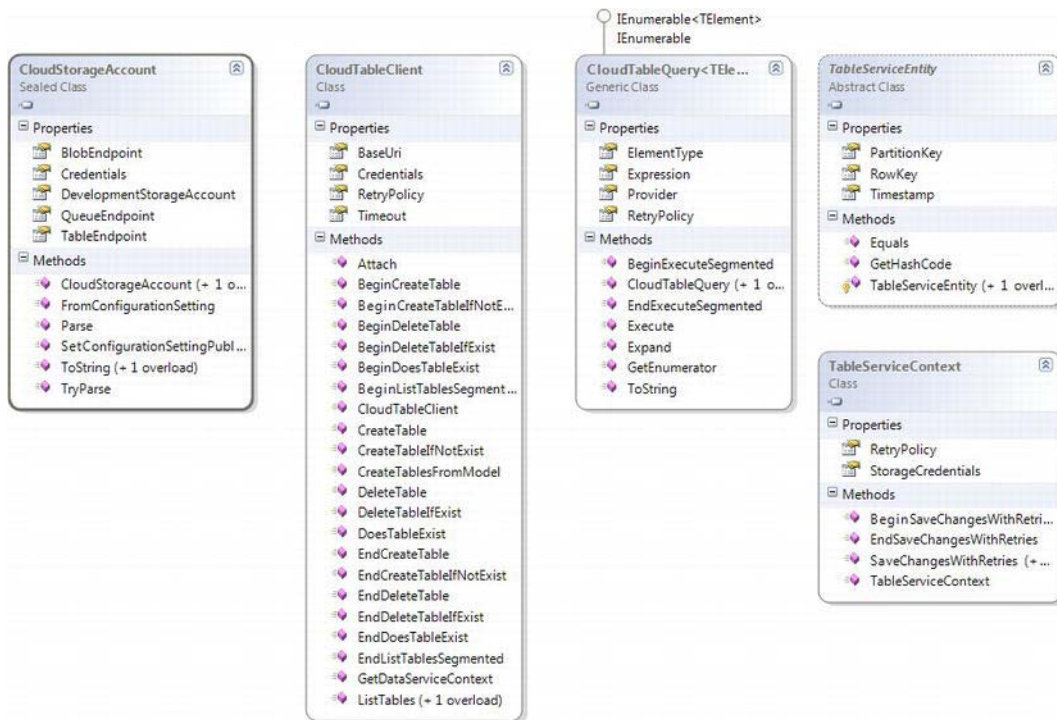


Figure 5-4. Table class hierarchy

As shown in Figure 5-4, five core classes are required for table operations. Table 5-2 describes each of them.

■ **Tip** The Windows Azure StorageClient API is the recommended method for programming storage service applications. The API provides synchronous as well as asynchronous methods to interact with the Storage service REST APIs.

Table 5-2. Classes for the Table Service

Class Name	Description
CloudStorageAccount	A helper class for retrieving account information from the configuration file or creating an instance of the storage account object from account parameters.

CloudTableClient	A wrapper class to interact with the Table service. It has methods like CreateTable(), DeleteTable(), GetDataServiceContext(), and ListTables().
TableServiceContext	Inherits from the System.Data.Services.Client.DataServiceContext class. It adds additional authentication functionality required by the Table service.
TableServiceEntity	An abstract class representing an entity(row) in a table. It has the mandatory properties (PartitionKey, RowKey, and Timestamp) defined in it. You may inherit your entity class from this class and provide additional properties.
CloudTableQuery<TElement>	Can be used to work with continuation tokens in Table service. A continuation token is similar to the NextMarker property you saw in the Blob and Queue services. It's a pointer to the next object available that wasn't retrieved due to the limit set on the results retrieved either by the application or the Table service itself.

The steps for programming simple table applications with these table classes are as follows:

1. Add the following using statement to your C# class:

```
using Microsoft.WindowsAzure.StorageClient;
```

2. Instantiate the CloudStorageAccount class from configuration file:

```
CloudStorageAccount storageAccountInfo =  
CloudStorageAccount.FromConfigurationSetting(configurationSettingName);
```

3. Or, instantiate the CloudStorageAccount class using account information:

```
CloudStorageAccount storageAccountInfo = new CloudStorageAccount(new  
StorageCredentialsAccountAndKey(accountName, accountKey), new Uri(blobEndpointURI), new  
Uri(queueEndpointURI), new Uri(tableEndpointURI));
```

4. Create an instance of CloudTableClient:

```
CloudTableClient tableStorageType = storageAccountInfo.CreateCloudTableClient ();
```

5. When you have an instance of the CloudTableClient class, you can execute operations on the table storage service as follows:

```
Create Table  
tableStorageType.CreateTable(tableName);  
Delete Table
```

```
tableStorageType.DeleteTable(tableName);
```

```
Get Tables
```

```
IEnumerable<string> tables = tableStorageType.ListTables();
```

The Table service is quite different from the Blob and Queue services because the tables you create in the Table service are custom and depend on the application's data storage requirements. Unlike Table service, the Queue and Blob services don't require custom schemas to be created. In the next few sections, you will learn how to design your own Table storage model and call some of these functions at every level of the Table service hierarchy.

Example Table Model

In this section you will learn to create a simple application with a one-table schema. The purpose of this exercise is to demonstrate a broader overview of the Table service's features.

■ **Note** The Windows Azure Table storage is not a relational database therefore you cannot define relationships between two tables. Referential integrity or normal forms don't apply to the Table storage; everything is denormalized.

The application you will create is called Pro Azure Reader Tracker. (You can go to the Pro Azure Reader Tracker web site and provide feedback.) The application has only one table, called ProAzureReader. The first step in developing for Table storage is to create an entity model representing each table record.

Figure 5-5 illustrates the ProAzureReader class representing the table schema (or the entity).

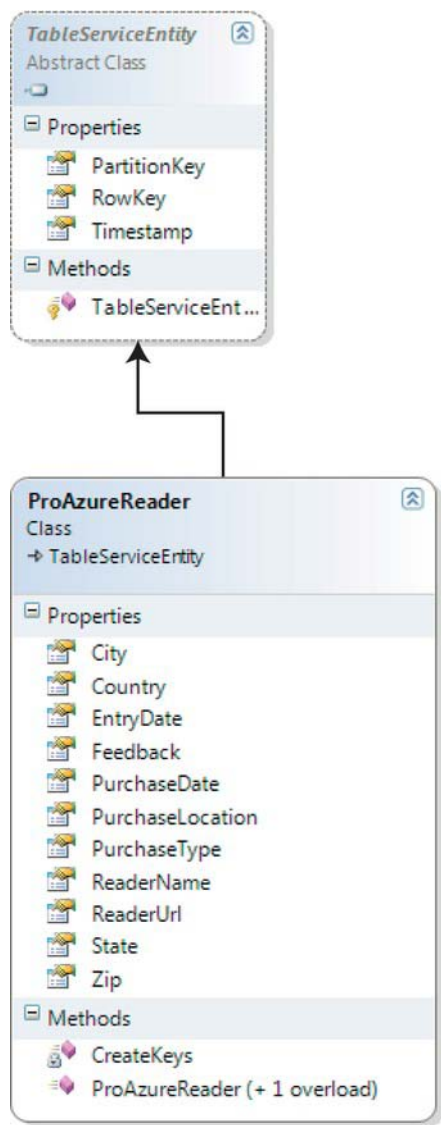


Figure 5-5. ProAzureReader schema

As shown in Figure 5-5, the ProAzureReader class inherits from the TableStorageEntity class from the Storage Client library. The TableServiceEntity class defines the mandatory entity properties required by the Table service: PartitionKey, RowKey, and Timestamp. The ProAzureReader class defines the properties required to capture reader information and feedback:

- The Feedback property represents the reader's feedback.

- The `EntryDate` property represents the data-entry date.
- The `PurchaseDate` property represents the date the book was purchased by the reader.
- The `PurchaseLocation` property represents the location where the reader purchased the book.
- The `PurchaseType` property represents whether the purchase was a new or used book.
- The rest of the properties represent user information including name, address, and personal URL.

The `ProAzureReader` class also creates the `PartitionKey` and `RowKey` for the entity record.

Figure 5-6 illustrates how the `ProAzureReaderDataContext` class inherits from the `TableStorageDataServiceContext` class, which in turn inherits from the `System.Data.Services.Client.DataServiceContext` class of the ADO.NET Data Services .NET client library.

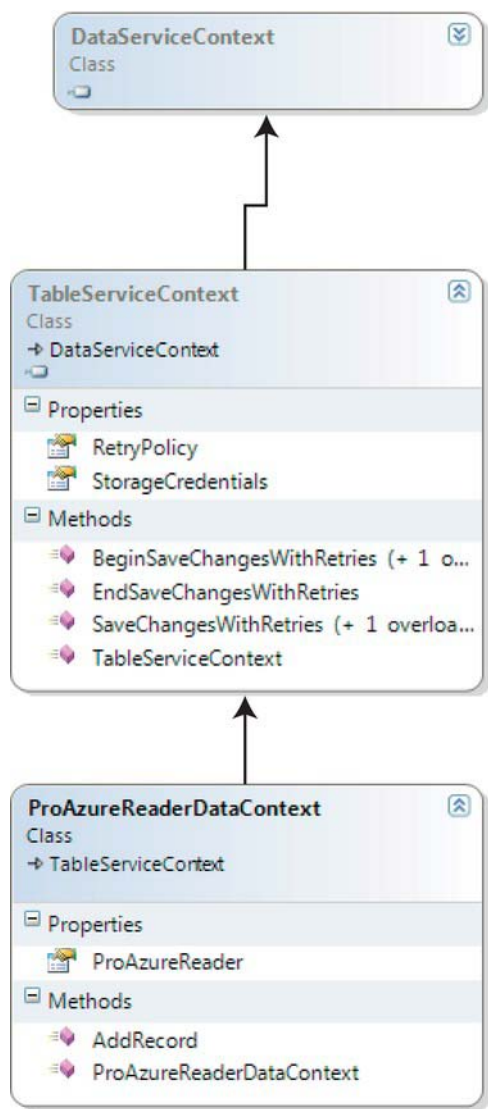


Figure 5-6. ProAzureReaderDataContext class

The DataServiceContext class represents the runtime context of ADO.NET Data Services. The context is a client-side construct and maintains the client-side state of invocations for update management between the client and the service.

Finally, the ProAzureReaderDataSource class is a utility class that wraps all the data queries and is used for binding the data with client-side controls. See Figure 5-7.

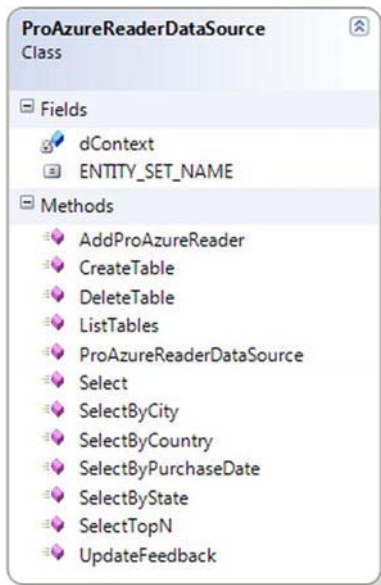


Figure 5-7. *ProAzureReaderDataSource* class

As illustrated in Figure 5-7, the *ProAzureReaderDataSource* class consists of methods for inserting an entity into the table and retrieving entities from the table.

To design the PartitionKey, first you need to find out the most dominant query in the application. The application lists all the feedback entered by readers on a particular day. When you go the web page, you see all the feedback entries for the day. As a result, the most dominant query in the application can be phrased as, “Get all the entities entered today.” If you design the PartitionKey as the same as the entity’s *EntryDate* property, all the entries with the same *EntryDate* are placed in the same partition by the Table service. This executes the query locally on the partition and yields better query performance.

The query should list the results sorted by time with the most recent entry at the top. To achieve this, the RowKey must be a function of *EntryDate*. Listing 5-4 shows the code for the *ProAzureReader* class and the *ProAzureReaderDataContext* class.

Listing 5-4. *ProAzureReader* Schema Classes

ProAzureReader.cs

```
public class ProAzureReader : TableServiceEntity
{
    public ProAzureReader()
    {
        CreateKeys();
    }
    public DateTime PurchaseDate
    { get; set; }
    public DateTime EntryDate
    { get; set; }
    public string Country
```

```

        { get; set; }
        public string State
        { get; set; }
        public string City
        { get; set; }
        public string Zip
        { get; set; }
        public string PurchaseLocation
        { get; set; }
        public string PurchaseType
        { get; set; }
        public string ReaderName
        { get; set; }
        public string ReaderUrl
        { get; set; }
        public string Feedback
        { get; set; }

        private void CreateKeys()
        {
            EntryDate = DateTime.UtcNow;
//By Entry Date: [Query: Get records entered today]
            PartitionKey = EntryDate.ToString("MMddyyyy");

            RowKey = string.Format("{0:10}_{1}",
DateTime.MaxValue.Ticks - EntryDate.Ticks, Guid.NewGuid());
        }
    }
}

ProAzureReaderDataContext.cs
public class ProAzureReaderDataContext : TableServiceContext
{
    public ProAzureReaderDataContext() : base(null, null) { }

    public IQueryable<ProAzureReader> ProAzureReader
    {
        get
        {
            return this.CreateQuery<ProAzureReader>("ProAzureReader");
        }
    }

    public void AddRecord(
        DateTime purchaseDate,
        string country,
        string state,
        string city,

```

```

        string zip,
        string purchaseLocation,
        string purchaseType,
        string readerName,
        string readerUrl,
        string feedback)
    {
        ProAzureReader pa = new ProAzureReader(city);
        pa.Country = country;
        pa.Feedback = feedback;
        pa.PurchaseDate = purchaseDate;
        pa.PurchaseLocation = purchaseLocation;
        pa.PurchaseType = purchaseType;
        pa.ReaderName = readerName;
        pa.ReaderUrl = readerUrl;
        pa.State = state;
        pa.Zip = zip;

        this.AddObject("ProAzureReader", pa);
        this.SaveChanges();
    }
}

```

As shown in Listing 5-4, the `CreateKeys()` methods in the `ProAzureReader` class sets the values of the `PartitionKey` and the `RowKey`. The `CreateKeys` method is called by the constructor of the class. The `PartitionKey` and `RowKey` are a function of the `EntryDate` property. The `RowKey` has a GUID associated with it to take into account multiple entities with the same `EntryDate`. If the dominant query of the application was, “Get all the records from a City,” you could design the `PartitionKey` as a function of the `City` property.

The `ProAzureReaderDataContext` class defines an `AddRecord` method for adding a new entity to the table. The method uses the `AddObject()` and `SaveChanges()` methods of the base class to save the entity to the table.

Next, you see how to use this schema model in executing table and entity operations.

Account Operations

The storage account provides an entry point to the Table service via the Table service endpoint URI. There are no methods at the Account level in the Table service hierarchy. The URI endpoint of a specific account is of the format `http://<account name>.table.core.windows.net`.

Table Operations

The Table service defines three methods at the table level of the hierarchy: `Create Table`, `Delete Table`, and `Query Tables`. Table 5-3 lists and describes the three operations, and Table 5-4 lists some important characteristics of these methods.

Table 5-3. Table Operations

Operation	Description
Create Table	Creates a new table under the given storage account. The table is actually created as an entity in a master Tables table.
Delete Table	Marks the specified table and its contents for deletion. The garbage collector deletes marked tables on a periodic basis. So, if you delete a table and try to create it immediately, the Table service complains that the table already exists.
Query Tables	Gets a list of tables from the specified storage account.

Table 5-4. Table Operations Characteristics

Operation	HTTP Verb	Cloud URI	Development Storage URI	HTTP Version	Permissions
Create Table	POST	http://<account name>.table.core.windows.net/Tables	http://127.0.0.1:10002/<devstorage account>/Tables	HTTP/1.1	Only the account owner can call this operation.
Delete Table	DELETE	http://<account name>.table.core.windows.net/Tables(' <table name>')	http://127.0.0.1:10002/<devstorage account>/Tables(' <table name>')	HTTP/1.1	Only the account owner can call this operation.
Query Tables	GET	http://<account name>.table.core.windows.net/Tables()	http://127.0.0.1:10002/<devstorage account>/Tables	HTTP/1.1	Only the account owner can call this operation.

The <account name> is the storage account name in the cloud, and the <devstorageaccount> is the development storage account. Observe that unlike with blob containers, the operations can be called only with account owner privileges. The following sections discuss some of the operations from Table 5-4 in detail. Even though the operations are different, the programming concepts behind them are similar. To keep the book at a conceptual level, I will discuss only the Create Table and Query Tables operations, because they cover most of the discussed concepts. Studying these operations in detail will enable you to understand the programming concepts behind all the table operations. The ProAzureReaderTracker_WebRole web role included with this chapter’s source code contains the implementation of the table operations.

Create Table

The Create Table operation creates a table in the storage account. Behind the scenes, the Table service creates an entity with the specified name in the master table Tables.

The URI for the Create Table operation is of the format `http://<account name>.table.core.windows.net/Tables`. Tables give you a structured storage data structure in the cloud. Because of the standard REST interface and Internet scale, you can create tables anywhere, anytime, and in any programming language that supports Internet programming. The Create Table REST request looks like Listing 5-5.

Listing 5-5. Create Table REST Request

```
POST /Tables HTTP/1.1
User-Agent: Microsoft ADO.NET Data Services
x-ms-date: Sun, 21 Jun 2009 18:42:29 GMT
Authorization: SharedKeyLite proazurestorage:
pwFouPw+BPWzlaQPyccII+K8zb+v6qygxZhp9fCdqRA=
Accept: application/atom+xml,application/xml
Accept-Charset: UTF-8
DataServiceVersion: 1.0;NetFx
MaxDataServiceVersion: 1.0;NetFx
Content-Type: application/atom+xml
Host: proazurestorage.table.core.windows.net
Content-Length: 499
Expect: 100-continue
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<entry xmlns:d=http://schemas.microsoft.com/ado/2007/08/dataservices
xmlns:m=http://schemas.microsoft.com/ado/2007/08/dataservices/metadata
xmlns="http://www.w3.org/2005/Atom">
  <title />
  <updated>2009-05-21T18:42:29.656Z</updated>
  <author>
    <name />
  </author>
  <id />
  <content type="application/xml">
    <m:properties>
      <d:TableName>MyFirstAzureTable</d:TableName>
    </m:properties>
  </content>
</entry>
```

Listing 5-5 shows the request to create a table named `MyFirstAzureTable`. The POST HTTP verb instructs the Table service to create a table. The request body consists of an ADO.NET entity set in Atom feed format. For the Create Table operation, the Table service responds with a status code of HTTP/1.1 201 Created or HTTP/1.1 409 Conflict if a table with the same name already exists. The Create Table response is shown in Listing 5-6.

Listing 5-6. Create Table REST Response

HTTP/1.1 201 Created

Cache-Control: no-cache

Content-Type: application/atom+xml; charset=utf-8

Location: http://proazurestorage.table.core.windows.net/Tables('MyFirstAzureTable')

Server: Table service Version 1.0 Microsoft-HTTPAPI/2.0

x-ms-request-id: 7347b965-9efb-4958-bcf5-d3616563fb28

Date: Sun, 21 Jun 2009 18:44:29 GMT

Content-Length: 836

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<entry xml:base=http://proazurestorage.table.core.windows.net/
xmlns:d=http://schemas.microsoft.com/ado/2007/08/dataservices
xmlns:m=http://schemas.microsoft.com/ado/2007/08/dataservices/metadata
xmlns="http://www.w3.org/2005/Atom">
  <id>http://proazurestorage.table.core.windows.net/Tables('MyFirstAzureTable')
</id>
  <title type="text"></title>
  <updated>2009-05-21T18:44:29Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Tables" href="Tables('MyFirstAzureTable')" />
  <category term="proazurestorage.Tables"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <content type="application/xml">
    <m:properties>
      <d:TableName>MyFirstAzureTable</d:TableName>
    </m:properties>
  </content>
</entry>
```

In Listing 5-6, the first line represents the status code of the operation. The x-ms-request-id represents a unique request identifier that can be used for debugging or tracing. The response body also contains an ADO.NET entity set in Atom feed format.

Figure 5-8 illustrates the Create Table operation in the ProAzureReaderTracker_WebRole web role.

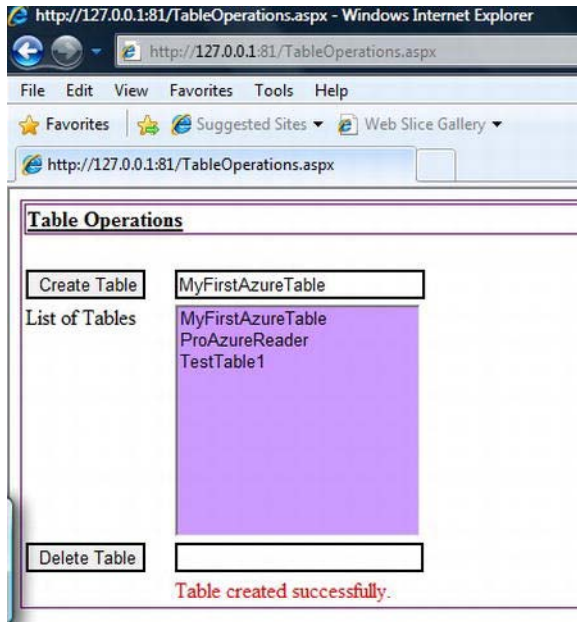


Figure 5-8. Create Table operation from the *ProAzureReaderTracker_WebRole* web role

As shown in Figure 5-8, to create a table, you must do the following:

1. Run the *ProAzureReaderTracker_WebRole* web role locally or in the cloud.
2. Make sure the appropriate table storage endpoint is specified in the *ServiceConfiguration.cscfg* file of the *ProAzureReaderTracker* cloud service.
3. Make *TableOperations.aspx* the default start page.
4. Run *ProAzureReaderTracker* cloud service.
5. Enter a table name (such as *MyFirstAzureTable*) in the text field next to the Create Table button.
6. Click the Create Table button to create the table in the Table service. If the table is created successfully, it appears in the List of Tables list box.

There is one more way to create a table using the schema model you created earlier in this section. To create a table using the schema model, go to *TableOperations.aspx* and click the Create *ProAzureReader* Table link button to create the *ProAzureReader* table from the schema.

To help you understand the programming model of the Create Table operation, open the Visual Studio Solution *Chapter4.sln* from the Chapter 4 source directory. The *WASStorageHelper* class in the *ProAzureCommonLib* contains a helper function called *CreateTable()*, as shown in Listing 5-7.

Listing 5-7. *CreateTable() Method in the WStorageHelper Class*

```
public void CreateTable(string tableName)
{
    TableClient.CreateTable(tableName);
}

```

The CreateTable() method calls the CreateTable() method on the CloudTableClient class from the StorageClient library. Listing 5-8 shows definition of the DeleteTable() method.

Listing 5-8. *Delete Table*

```
public void DeleteTable(string tableName)
{
    TableClient.DeleteTable(tableName);
}

```

Similar to the CreateTable method, the DeleteTable() method simply calls the DeleteTable() method of the CloudTableClient class.

Query Tables

The Query Tables operation returns a list of all the tables in a storage account. The Table service returns a maximum of 1,000 items in a single query. But similar to the NextMarker element you saw in the Blob and Queue services, the Table service returns a pointer x-ms-continuation-NextTableName. You can send a follow-up request to the Table service to retrieve the remaining items by passing x-ms-continuation-NextTableName as a URI parameter. The Query Tables REST request looks like Listing 5-9.

Listing 5-9. *Query Tables REST Request*

```
GET /Tables()?$top=50 HTTP/1.1
User-Agent: Microsoft ADO.NET Data Services
x-ms-date: Sun, 21 Jun 2009 18:42:10 GMT
Authorization: SharedKeyLite proazurestorage:
hZTV+6FS1lWguxB4vBiDvbubPMAlt2kK+kIpVmrYme8=
Accept: application/atom+xml,application/xml
Accept-Charset: UTF-8
DataServiceVersion: 1.0;NetFx
MaxDataServiceVersion: 1.0;NetFx
Host: proazurestorage.table.core.windows.net
Connection: Keep-Alive

```

In Listing 5-9, the HTTP verb used is GET. Note the URI parameter \$top=50; this parameter instructs the Table service to return a maximum of 50 items for this call. Listing 5-10 shows the response from the Table service for the Query Tables operation.

Listing 5-10. Query Tables REST Response

```

HTTP/1.1 200 OK
Cache-Control: no-cache
Content-Type: application/atom+xml; charset=utf-8
Server: Table Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: d3ca497d-65d2-4fb5-a51e-3babec57e525
Date: Sun, 21 Jun 2009 18:44:09 GMT
Content-Length: 1630

<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xml:base=http://proazurestorage.table.core.windows.net/
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata
xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Tables</title>
  <id>http://proazurestorage.table.core.windows.net/Tables</id>
  <updated>2009-05-21T18:44:10Z</updated>
  <link rel="self" title="Tables" href="Tables" />
  <entry>
    <id>
      http://proazurestorage.table.core.windows.net/Tables('ProAzureReader')
    </id>
    <title type="text"></title>
    <updated>2009-05-21T18:44:10Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="Tables" href="Tables('ProAzureReader')"/>
    <category term="proazurestorage.Tables"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <content type="application/xml">
      <m:properties>
        <d:TableName>ProAzureReader</d:TableName>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id>
      http://proazurestorage.table.core.windows.net/Tables('TestTable1')
    </id>
    <title type="text"></title>
    <updated>2009-05-21T18:44:10Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="Tables" href="Tables('TestTable1')"/>
    <category term="proazurestorage.Tables"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <content type="application/xml">

```

```
<m:properties>
  <d:TableName>TestTable1</d:TableName>
</m:properties>
</content>
</entry>
</feed>
```

As shown in Listing 5-10, the Query Tables response contains two tables ProAzureReader and TestTable1. Figure 5-9 illustrates the Query Tables operation in the ProAzureReaderTracker_WebRole web role.

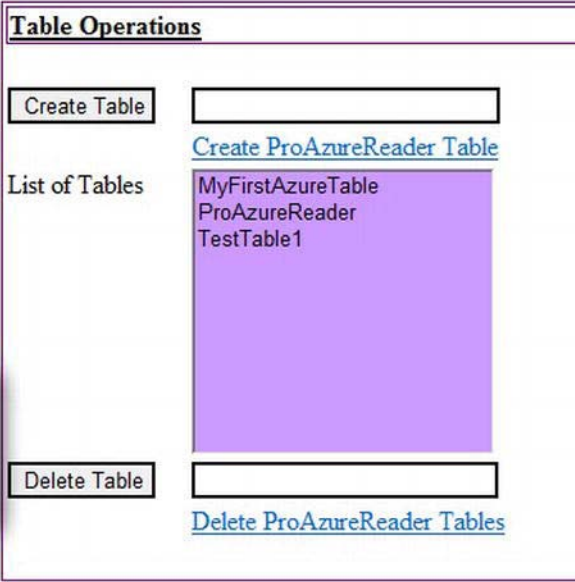


Figure 5-9. Query Tables operation in the ProAzureReaderTracker_WebRole web role

As illustrated in Figure 5-9, the TableOperations.aspx loads all the tables from the storage account in the list box. To help you understand the programming model of the Query Tables operation, open the Visual Studio Solution Chapter4.sln from the Chapter 4 source directory. The WStorageHelper class in ProAzureCommonLib contains a helper method called ListTables(), as shown in Listing 5-11.

Listing 5-11. *ListTables() Method in the WStorageHelper Class*

```

public IEnumerable<string> ListTables()
{
    return TableClient.ListTables();
}

```

In Listing 5-11, the ListTables() method calls the ListTables() method on the CloudTableClient object from StorageClient library. The TableClient object utilizes the System.Data.Services.Client.DataServiceQuery object to retrieve a list of tables from the Table service.

Figure 5-10 illustrates the sequence diagram for the Query Tables operation.

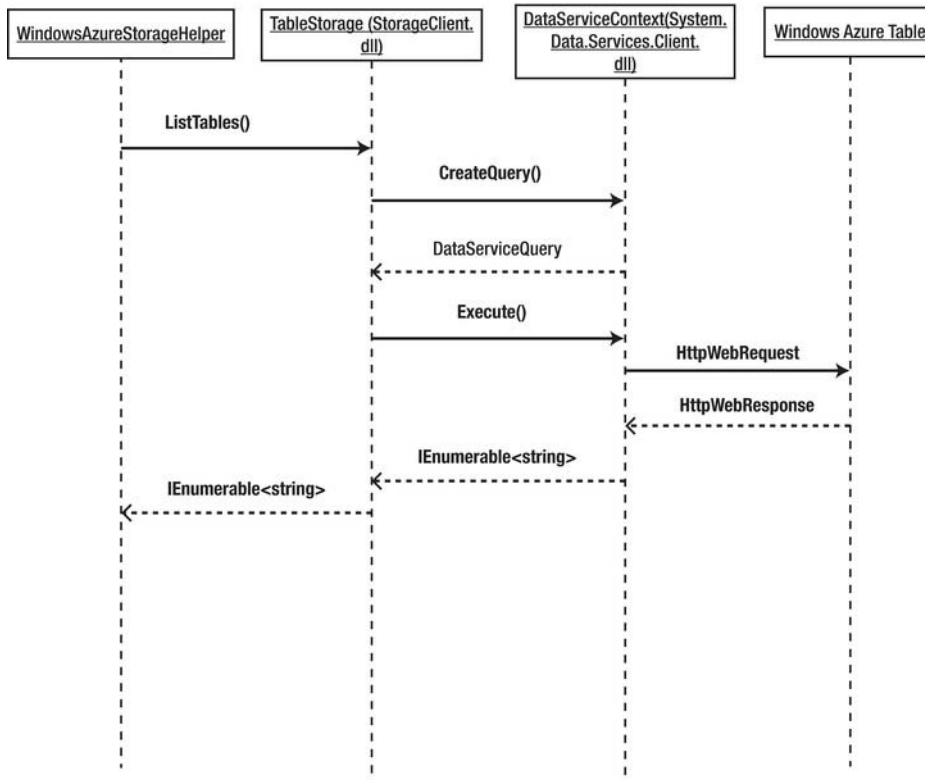


Figure 5-10. *List Tables sequence diagram*

As illustrated in Figure 5-10, the WStorageHelper object calls the ListTables() method on the CloudTableClient object from the StorageClient library. The TableStorage object utilizes the DataServiceContext object to create an instance of the DataServiceQuery class. TableStorage then calls

the `Execute()` method on the `DataServiceQuery` object to retrieve the list of tables from the Table service. The Table service returns a list of all the table names from the storage account.

Entity Operations

Entities support several operations, as listed in Table 5-5.

Table 5-5. Entity Operations

Operation	Description
Query Entities	Queries for a list of entities in a table.
Insert Entity	Adds a new entity to the table.
Update Entity	Updates or replaces an entire entity in the table.
Merge Entity	Only updates the properties of an entity in the table. Properties with null values are ignored by this operation.
Delete Entity	Deletes an existing entity from a table.

Table 5-6 lists some of the important characteristics of the entity operations listed in Table 5-5.

Table 5-6. Entity Operations Characteristics

Operation	HTTP Verb	Cloud URI	Development Storage URI	HTTP Version	Permissions
Query Entities	GET	<code>http://<account name>.table.core.windows.net/<table name>()?&\$filter=<query-expression></code>	<code>http://127.0.0.1:10002/<devstorage account>/<table name>()?&\$filter=<query-expression></code>	HTTP/1.1	Only the account owner can call this operation.
Insert Entity	POST	<code>http://<account name>.table.core.windows.net/<table name></code>	<code>http://127.0.0.1:10002/<devstorage account>/<table name></code>	HTTP/1.1	Only the account owner can call this operation.

Operation	HTTP Verb	Cloud URI	Development Storage URI	HTTP Version	Permissions
Update Entity	PUT	<code>http://<account name>.table.core.windows.net/<table name>(PartitionKey="x", RowKey="y")</code>	<code>http://127.0.0.1:10002/<devstorage account>/<table name>(PartitionKey="x", RowKey="y")</code>	HTTP/1.1	Only the account owner can call this operation.
Merge Entity	MERGE	<code>http://<account name>.table.core.windows.net/<table name>(PartitionKey="x", RowKey="y")</code>	<code>http://127.0.0.1:10002/<devstorage account>/<table name>(PartitionKey="x", RowKey="y")</code>	HTTP/1.1	Only the account owner can call this operation.
Delete Entity	DELETE	<code>http://<account name>.table.core.windows.net/<table name>(PartitionKey="x", RowKey="y")</code>	<code>http://127.0.0.1:10002/<devstorage account>/<table name>(PartitionKey="x", RowKey="y")</code>	HTTP/1.1	Only the account owner can call this operation.

The *<account name>* is the storage account name in the cloud, and the *<devstorageaccount>* is the development storage account. The *<table name>* is the name of the table you want to query on. The following sections discuss some of the operations from Table 5-6 in detail. Even though the operations are different, the programming concepts behind them are similar. To keep the book at a conceptual level, I discuss the Query Entities, Insert Entity, and Merge Entity operations, because they cover most of the discussed concepts. By studying these three operations in detail, you can understand the programming concepts behind all the entity operations.

Query Entities

The URI for the Query Entities operation is of the form `http://<account name>.table.core.windows.net/<table name>()?&filter=<query-expression>` or `http://<account name>.table.core.windows.net/<table name>(PartitionKey="x", RowKey="y")`. Entities are analogous to rows in a relational table. So, you need a flexible mechanism to specify query parameters and filter criteria for the query. The URL parameters for the Query Entities operation support the ADO.NET Data Services query options as defined in the ADO.NET Data Service Specifications.⁴ The *\$filter* and *\$top* URL parameters discussed earlier in this section are the most commonly used criteria for querying entities in a table.

⁴ ADO.NET Data Services Query Options: <http://msdn.microsoft.com/en-us/library/cc668809.aspx>

You can use LINQ to query entities in a table. When you enumerate over a LINQ statement, the query is created and sent to the server, and results are retrieved. Listing 5-12 shows an example Query Entities REST request.

Listing 5-12. Query Entities REST Request

```
GET /ProAzureReader()?$top=2 HTTP/1.1
User-Agent: Microsoft ADO.NET Data Services
x-ms-date: Mon, 22 Jun 2009 02:35:26 GMT
Authorization: SharedKeyLite
    proazurestorage:K+P5VD/AIhS22b6yui04LR1kxx1V4v4/Cy5rc+5nIr0=
Accept: application/atom+xml,application/xml
Accept-Charset: UTF-8
DataServiceVersion: 1.0;NetFx
MaxDataServiceVersion: 1.0;NetFx

Host: proazurestorage.table.core.windows.net
```

Listing 5-12 shows the request for querying the ProAzureReader table with a \$top=2 criteria to retrieve top two items. The Query Entities operation can return only 1,000 items in a single call. If the number of items that fit the filter criteria is greater than 1,000 or the query times out, the Table services sends two continuation tokens: - x-ms-continuation-NextPartitionKey and x-ms-continuation-NextRowKey in the response. Similar to the NextMarker token you saw in the Blob and Queue services, these tokens point to the first item in the next data set. Listing 5-13 shows the REST response from the Table service for the Query Entities operation.

Listing 5-13. Query Entities REST Response

```
HTTP/1.1 200 OK

Cache-Control: no-cache
Content-Type: application/atom+xml; charset=utf-8
Server: Table Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: ab64434d-9a8d-4090-8397-d8a9dad5da8a

x-ms-continuation-NextPartitionKey: 1!12!MDYyMDIwMDk-
x-ms-continuation-NextRowKey: 1!76!MTI1MjE1NjgwMjgzNzA1MTk5OTI1fM

Date: Mon, 22 Jun 2009 02:38:19 GMT
Content-Length: 3592

<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xml:base="http://proazurestorage.table.core.windows.net/"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text">ProAzureReader</title>
  <id>http://proazurestorage.table.core.windows.net/ProAzureReader</id>
  <updated>2009-05-22T02:38:19Z</updated>
  <link rel="self" title="ProAzureReader" href="ProAzureReader" />
  <entry m:etag="W/&quot;datetime'2009-05-20T23%3A30%3A15.251Z'&quot;;">
    <id>http://proazurestorage.table.core.windows.net/
    ProAzureReader(PartitionKey='06202009',RowKey='12521567602930729999')
```



```

</id>
<title type="text"></title>
<updated>2009-05-22T02:38:19Z</updated>
<author>
  <name />
</author>
<link rel="edit" title="ProAzureReader"
href="ProAzureReader(PartitionKey='06202009',RowKey='12521567602930729999')"/>
<category term="proazurestorage.ProAzureReader"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<content type="application/xml">
  <m:properties>
    <d:PartitionKey>06202009</d:PartitionKey>
    <d:RowKey>
      12521567602930729999
    </d:RowKey>
    <d:Timestamp m:type="Edm.DateTime">2009-05-20T23:30:15.251Z</d:Timestamp>
    <d:City></d:City>
    <d:Country></d:Country>
    <d:EntryDate m:type="Edm.DateTime">2009-05-20T23:28:26.927Z</d:EntryDate>
    <d:Feedback>Good Book :). But don't write again.</d:Feedback>
    <d:PurchaseDate m:type="Edm.DateTime">2009-05-20T00:00:00Z</d:PurchaseDate>
    <d:PurchaseLocation></d:PurchaseLocation>
    <d:PurchaseType>New</d:PurchaseType>
    <d:ReaderName></d:ReaderName>
    <d:ReaderUrl></d:ReaderUrl>
    <d:State></d:State>
    <d:Zip></d:Zip>
  </m:properties>
</content>
</entry>
<entry m:etag="W/&quot;datetime'2009-05-20T13%3A01%3A10.5846Z'&quot;">
  <id>http://proazurestorage.table.core.windows.net/
ProAzureReader(PartitionKey='06202009',RowKey='12521567980278019999')
</id>
<title type="text"></title>
<updated>2009-05-22T02:38:19Z</updated>
<author>
  <name />
</author>
<link rel="edit" title="ProAzureReader"
href="ProAzureReader(PartitionKey='06202009',RowKey='12521567980278019999')"/>
<category term="proazurestorage.ProAzureReader"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<content type="application/xml">
  <m:properties>
    <d:PartitionKey>06202009</d:PartitionKey>
    <d:RowKey>12521567980278019999</d:RowKey>
    <d:Timestamp m:type="Edm.DateTime">2009-05-20T13:01:10.5846Z</d:Timestamp>
    <d:City>mumbai</d:City>
    <d:Country>india</d:Country>
    <d:EntryDate m:type="Edm.DateTime">2009-05-20T12:59:32.198Z</d:EntryDate>

```

```

    <d:Feedback>Good Book :). But don't write again.</d:Feedback>
    <d:PurchaseDate m:type="Edm.DateTime">2009-05-20T00:00:00Z</d:PurchaseDate>
    <d:PurchaseLocation>web</d:PurchaseLocation>
    <d:PurchaseType>New</d:PurchaseType>
    <d:ReaderName>tredkar</d:ReaderName>
    <d:ReaderUrl></d:ReaderUrl>
    <d:State>maharashtra</d:State>
    <d:Zip>400028</d:Zip>
  </m:properties>
</content>
</entry>
</feed>

```

In Listing 5-13, the first line represents the status code of the operation. The response body consists of ADO.NET entity set in Atom feed format. The body shows two items retrieved. The `x-ms-continuation-NextPartitionKey` and `x-ms-continuation-NextRowKey` indicate pointers to the first item from the next data set. To retrieve the remaining items, you can pass the two tokens as the `NextPartitionKey` and `NextRowKey` URL parameters of a subsequent call. The `x-ms-continuation-NextPartitionKey` and `x-ms-continuation-NextRowKey` are used to page on Query Entity results.

In the Pro Azure Reader Tracker application, you can implement the following queries:

- Get all the entries entered today (the dominant query).
- Get entries by city, state, or country.
- Get the Top(*n*) entries.
- Get entries by purchase date.

Listing 5-14 shows the implementation of each of these queries using LINQ. The methods are implemented in the `ProAzureReaderDataSource` class.

Listing 5-14. *Query Entities in the ProazureReaderDataSource Class*

```

private TableServiceContext dContext;

public CloudStorageAccount Account {get;set;}
public CloudTableClient TableClient { get; set; }

public const string ENTITY_SET_NAME = "ProAzureReader";
public ProAzureReaderDataSource()
{
    Init("StorageAccountConnectionString");
    dContext = TableClient.GetDataServiceContext();
    dContext.RetryPolicy = RetryPolicies.Retry(3, TimeSpan.FromSeconds(5));
}

public ProAzureReaderDataSource(string storageAccountConnectionString)
{
    Init(storageAccountConnectionString);
    dContext = TableClient.GetDataServiceContext();
    dContext.RetryPolicy = RetryPolicies.Retry(3, TimeSpan.FromSeconds(5));
}

```

```

    }

    private void Init(string configurationSettingName)
    {
        if (RoleEnvironment.IsAvailable)
        {
            CloudStorageAccount.SetConfigurationSettingPublisher(
                (configName, configSettingPublisher) =>
                {
                    var connectionString =
RoleEnvironment.GetConfigurationSettingValue(configName);
                    configSettingPublisher(connectionString);
                }
            );
        }
        else
        {
            CloudStorageAccount.SetConfigurationSettingPublisher(
                (configName, configSettingPublisher) =>
                {
                    var connectionString =
ConfigurationManager.ConnectionStrings[configName].ConnectionString;
                    configSettingPublisher(connectionString);
                }
            );
        }

        Account = CloudStorageAccount.FromConfigurationSetting(configurationSettingName);

        TableClient = Account.CreateCloudTableClient();
        TableClient.RetryPolicy = RetryPolicies.Retry(3, TimeSpan.FromMilliseconds(100) );
    }

    public IEnumerable<ProAzureReader> Select()
    {
        var results = from g in dContext.CreateQuery<ProAzureReader>(ENTITY_SET_NAME)
        where g.PartitionKey == DateTime.UtcNow.ToString("MMddyyyy")
        select g;

        var r = results.ToArray<ProAzureReader>();
        return r;
    }

    public IEnumerable<ProAzureReader> SelectByCity(string city)
    {
        var results = from g in dContext.CreateQuery<ProAzureReader>(ENTITY_SET_NAME)
        where g.PartitionKey == DateTime.UtcNow.ToString("MMddyyyy")
        && g.City == city
        select g;
    }

```

```

    var r = results.ToArray<ProAzureReader>();
    return r;
}
public IEnumerable<ProAzureReader> SelectByState(string state)
{
    var results = from g in dContext.CreateQuery<ProAzureReader>(ENTITY_SET_NAME)
                  where g.PartitionKey == DateTime.UtcNow.ToString("MMddyyyy")
                  && g.State == state
                  select g;

    var r = results.ToArray<ProAzureReader>();
    return r;
}
public IEnumerable<ProAzureReader> SelectByCountry(string country)
{
    var results = from g in dContext.CreateQuery<ProAzureReader>(ENTITY_SET_NAME)
                  where g.PartitionKey == DateTime.UtcNow.ToString("MMddyyyy")
                  && g.Country == country
                  select g;

    var r = results.ToArray<ProAzureReader>();
    return r;
}

public IEnumerable<ProAzureReader> SelectByPurchaseDate(DateTime purchaseDate)
{
    var results = from g in dContext.CreateQuery<ProAzureReader>(ENTITY_SET_NAME)
                  where g.PurchaseDate.Equals(purchaseDate )
                  select g;

    var r = results.ToArray<ProAzureReader>();
    return r;
}

public IEnumerable<ProAzureReader> SelectTopN(int topNumber)
{
    var results =
dContext.CreateQuery<ProAzureReader>(ENTITY_SET_NAME).Take(topNumber);
    var r = results.ToArray<ProAzureReader>();
    return r;
}

public void AddProAzureReader(ProAzureReader newItem)
{
    dContext.AddObject(ENTITY_SET_NAME, newItem);
    dContext.SaveChangesWithRetries(SaveChangesOptions.None);
}
public void UpdateFeedback(string PartitionKey, string RowKey, string feedback)
{
    var results = from g in dContext.CreateQuery<ProAzureReader>(ENTITY_SET_NAME)

```

```

        where g.PartitionKey == PartitionKey
        && g.RowKey == RowKey
        select g;

var e = results.FirstOrDefault<ProAzureReader>();
e.Feedback = feedback;
dContext.MergeOption = MergeOption.PreserveChanges;
dContext.UpdateObject(e);
dContext.SaveChanges();
}

public void UpdateUrl(string PartitionKey, string RowKey, string url)
{
    var results = from g in dContext.CreateQuery<ProAzureReader>(ENTITY_SET_NAME)
                  where g.PartitionKey == PartitionKey
                  && g.RowKey == RowKey
                  select g;
    var e = results.FirstOrDefault<ProAzureReader>();
    e.ReaderUrl = url;
    dContext.MergeOption = MergeOption.PreserveChanges;
    dContext.UpdateObject(e);
    dContext.SaveChanges();

}

```

In Listing 5-14, each method implements a LINQ query for Query Entities on the ProAzureReader table. On the Default.aspx page of the ProAzureReaderTracker_WebRole web role is a link button for each of the queries; see Figure 5-11.

Pro Azure Reader Tracker

Please enter your details and click Submit

Your Name/Email

Country

City

State

Zip

Book purchase location

Purchase Type

Purchase Date

Your Url
(e.g. Facebook, LinkedIn, etc.)

Any feedback?

q

usa

san ramon

ca

94582

web

New

6/21/2009

Good Book :). But don't write again.

Submit

Filter Text

2

By City

By State

By Country

Today's Entries

Top N (50)

6/21/2009

By Purchase Date

from , , ,

Url:

purchased a New book at on 6/20/2009 12:00:00 AM

Has the following Feedback:

Good Book :). But don't write again.

tredkar from india, maharashtra, mumbai, 400028

Url:

purchased a New book at web on 6/20/2009 12:00:00 AM

Has the following Feedback:

Good Book :). But don't write again.

Figure 5-11. Query entities in the ProAzureReaderTracker_WebRole web role

As shown in Figure 5-11, you can specify filter criteria in the filter text box and click one of the link buttons to execute the query.

■ **Tip** If you're running the web application on the local machine, you can run the Fiddler trace tool and capture the request and response contents of each query.

Insert Entity

The Insert Entity operation inserts an entity into the specified table. This operation requires the PartitionKey and RowKey to be specified. The URI for the insert Entity operation is of the format `http://<account name>.table.core.windows.net/<table name>`. A typical Insert Entity REST request looks like Listing 5-15.

Listing 5-15. Insert Entity REST Request

```
POST /ProAzureReader HTTP/1.1
User-Agent: Microsoft ADO.NET Data Services
x-ms-date: Mon, 22 Jun 2009 03:25:47 GMT
Authorization: SharedKeyLite proazurestorage:
    mazZ5pykdE1CmH5+SDe7fqWDLQpnWDcK1pgWDvyzXss=
Accept: application/atom+xml,application/xml
Accept-Charset: UTF-8
DataServiceVersion: 1.0;NetFx
MaxDataServiceVersion: 1.0;NetFx
Content-Type: application/atom+xml
Host: proazurestorage.table.core.windows.net
Content-Length: 1178
Expect: 100-continue

<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<entry xmlns:d=http://schemas.microsoft.com/ado/2007/08/dataservices
xmlns:m=http://schemas.microsoft.com/ado/2007/08/dataservices/metadata
xmlns="http://www.w3.org/2005/Atom">
  <title />
  <updated>2009-05-22T03:25:47.469Z</updated>
  <author>
    <name />
  </author>
  <id />
  <content type="application/xml">
    <m:properties>
      <d:City>san ramon</d:City>
      <d:Country>usa</d:Country>
      <d:EntryDate m:type="Edm.DateTime">2009-05-22T03:25:46.976Z</d:EntryDate>
      <d:Feedback>Excellent Book</d:Feedback>
      <d:PartitionKey>06222009</d:PartitionKey>
      <d:PurchaseDate m:type="Edm.DateTime">2009-05-21T00:00:00</d:PurchaseDate>
      <d:PurchaseLocation>amazon.com</d:PurchaseLocation>
      <d:PurchaseType>New</d:PurchaseType>
      <d:ReaderName>tejaswi</d:ReaderName>
      <d:ReaderUrl m:null="false" />
      <d:RowKey>12521566596530239999_7e9f46ea</d:RowKey>
      <d:State>ca</d:State>
      <d:Timestamp m:type="Edm.DateTime">0001-01-01T00:00:00</d:Timestamp>
      <d:Zip>94582</d:Zip>
    </m:properties>
```

</content>

In Listing 5-15, the HTTP verb used is POST, to instruct the Table service that this is an insert operation. The request body contains an ADO.NET entity set. The `m:properties` element defines the property names and values of the entity. In reality, it represents a serialized `ProAzureReader` object discussed earlier in this chapter. The `m:type` attribute specifies the data type of the property. The default property is `Edm.String` if the property is omitted. After the entity is created successfully, the response header consists of an HTTP 1.1 201 (Created) status code. The response body also contains the same ADO.NET entity set that was part of the request body.

The Submit button on the `Default.aspx` page in the `ProAzureReaderTracker_WebRole` project is tied to an Insert Entity operation because it inserts a new entity into the `ProAzureReader` table. Figure 5-12 shows the `Default.aspx` page.

Pro Azure Reader Tracker

Please enter your details and click Submit

Your Name/Email

tejaswi

Country

usa

City

san ramon

State

ca

Zip

94582

Book purchase location

amazon.com

Purchase Type

New

Purchase Date

6/21/2009

Your Url
(e.g. Facebook, LinkedIn, etc.)

Any feedback?

Excellent Book

Submit

Filter Text

2

[By City](#)

[By State](#)

[By Country](#)

[Today's Entries](#)

[Top N \(50\)](#)

6/21/2009

[By Purchase Date](#)

tejaswi from usa, ca, san ramon, 94582

Url:

purchased a New book at amazon.com on 6/21/2009 12:00:00 AM

Has the following Feedback:

Excellent Book

q from usa, ca, san ramon, 94582

Url:

purchased a New book at web on 6/21/2009 12:00:00 AM

Has the following Feedback:

Good Book :). But don't write again.

Figure 5-12. `Default.aspx` in the `ProAzureReaderTracker_WebRole` web role

In Figure 5-12, when you enter all the reader properties and click Submit, a new entity is created and the data list is refreshed, showing the new entity at the top. Listing 5-16 shows the code for the `btnSubmit_Click` event from the `Default.aspx.cs` file. Listing 5-16 also shows the `AddProAzureReader()` method from the `ProAzureReaderDataSource` class, which is called by the `btnSubmit_Click` method to insert a new entity.

Listing 5-16. Insert Entity

```
//Default.aspx
protected void btnSubmit_Click(object sender, EventArgs e)
{
    try
    {
        ProAzureReader newReader = new ProAzureReader()
        {
            City = txtCity.Text,
            Country = txtCountry.Text,
            Feedback = txtFeedback.Text,
            PurchaseDate = DateTime.Parse(txtPurchaseDate.Text),
            PurchaseType = ddlPurchaseType.SelectedItem.Text,
            PurchaseLocation = txtPurchaseLocation.Text,
            ReaderName = txtName.Text,
            ReaderUrl = txtUrl.Text,
            State = txtState.Text,
            Zip = txtZip.Text

        };

        ProAzureReaderDataSource ds = new ProAzureReaderDataSource();
        ds.AddProAzureReader(newReader);
    }
    catch (Exception ex)
    {
        lblStatus.Text = "Error adding entry " + ex.Message;
    }
}

//ProAzureDataSource.cs
public void AddProAzureReader(ProAzureReader newItem)
{
    dContext.AddObject(ENTITY_SET_NAME, newItem);
    dContext.SaveChangesWithRetries(SaveChangesOptions.None);
}
```

As shown in Listing 5-16, the `AddProAzureReader()` method calls the `AddObject()` and `SaveChangesWithRetries()` methods on the `ProAzureReaderDataContext` object. `SaveChangesOptions` is an ADO.NET Data Services enumeration with four possible values:

- *None* specifies that the operation is non-batch and should stop if any error occurs.
- *Batch* specifies that multiple changes are packaged into one change set and sent to the server in a single call.
- *ContinueOnError* specifies that subsequent operations are attempted even if an error occurs in one of the operations.
- *ReplaceOnUpdate* replaces all the properties of an entity on the server with the new ones specified.

Figure 5-13 illustrates the sequence diagram for the Insert Entity operation.

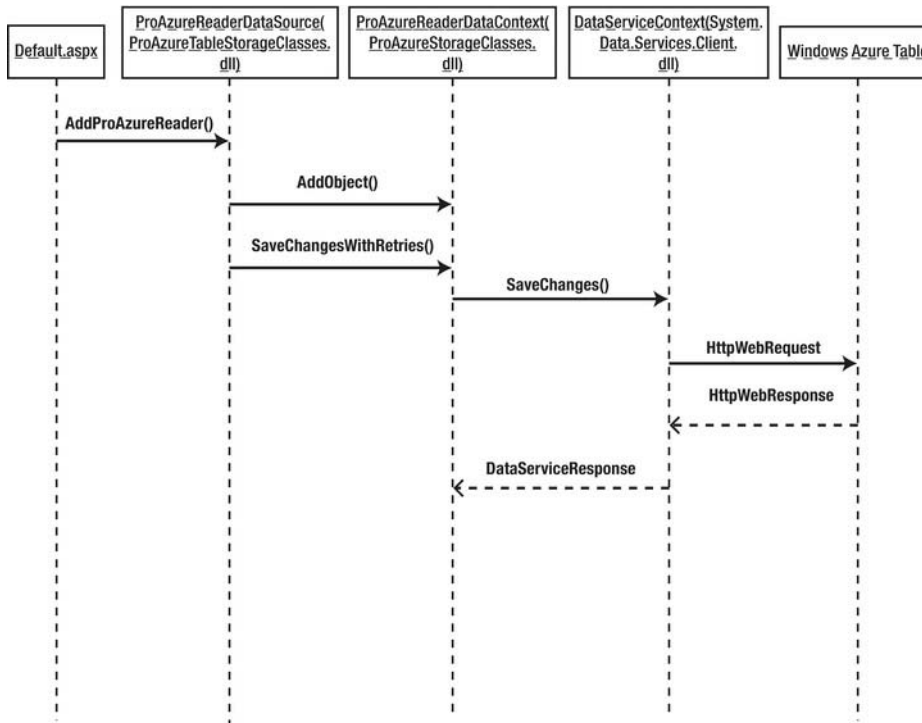


Figure 5-13. Insert Entity sequence diagram

As shown in Figure 5-13, the Default.aspx page calls the AddProAzureReader() method on the ProAzureReaderDataSource object. Default.aspx gathers the user input, creates a new ProAzureReader object, and passes it as a parameter to the AddProAzureReader() method. The AddProAzureReader() method calls the AddObject() and SaveChangesWithRetries() methods on the ProAzureReaderDataContext object, which in turn calls the SaveChanges() method on its parent class (DataServiceContext) object. The SaveChanges() method sends the HTTP request and receives the response from the Table service.

■ **Note** The Table service supports ACID transactions for batch operations on multiple entities on a single partition (with the same PartitionKey).⁵ The constraints are as follows: same PartitionKey, one transaction per entity in the batch, no more than 100 transactions in the batch, and the total batch payload size should be less than 4MB.

Merge Entity

The Merge Entity operation updates the properties of an entity without replacing an entity from the specified table. It requires the PartitionKey and RowKey to be specified. The URI for the Merge Entity operation is of the format `http://<account name>.table.core.windows.net/<table name>(PartitionKey="x", RowKey="y")`. A typical Merge Entity REST request looks like Listing 5-17.

Listing 5-17. Merge Entity REST Request

```
MERGE /ProAzureReader(PartitionKey='06222009',RowKey='12521566596530999') HTTP/1.1
```

```
User-Agent: Microsoft ADO.NET Data Services
```

```
x-ms-date: Mon, 22 Jun 2009 07:02:04 GMT
```

```
Authorization: SharedKeyLite proazurestorage:motXsCh9vzZZpNLbJ8xsNgm095
```

```
Accept: application/atom+xml,application/xml
```

```
Accept-Charset: UTF-8
```

```
DataServiceVersion: 1.0;NetFx
```

```
MaxDataServiceVersion: 1.0;NetFx
```

```
Content-Type: application/atom+xml
```

```
If-Match: W/"datetime'2009-05-22T07%3A01%3A13.043Z'"
```

```
Host: proazurestorage.table.core.windows.net
```

```
Content-Length: 1355
```

```
Expect: 100-continue
```

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<entry xmlns:d=http://schemas.microsoft.com/ado/2007/08/dataservices
xmlns:m=http://schemas.microsoft.com/ado/2007/08/dataservices/metadata
xmlns="http://www.w3.org/2005/Atom">
  <title />
  <updated>2009-05-22T07:02:04.948Z</updated>
  <author>
    <name />
  </author>
  <id>http://proazurestorage.table.core.windows.net/ProAzureReader
(PartitionKey='06222009',RowKey='12521566596530239999')
</id>
  <content type="application/xml">
```

⁵ Performing Entity Group Transactions: <http://msdn.microsoft.com/en-us/library/dd894038.aspx>

```

<m:properties>
  <d:City>san ramon</d:City>
  <d:Country>usa</d:Country>
  <d:EntryDate m:type="Edm.DateTime">2009-05-22T03:25:46.976Z</d:EntryDate>
  <d:Feedback>Excellent Book</d:Feedback>
  <d:PartitionKey>06222009</d:PartitionKey>
  <d:PurchaseDate m:type="Edm.DateTime">2009-05-21T00:00:00Z</d:PurchaseDate>
  <d:PurchaseLocation>amazon.com</d:PurchaseLocation>
  <d:PurchaseType>New</d:PurchaseType>
  <d:ReaderName>tejaswi</d:ReaderName>
  <d:ReaderUrl>http://www.bing.com</d:ReaderUrl>
  <d:RowKey>12521566596530239999_7e9f46ea-4230-4abb-bbd1</d:RowKey>
  <d:State>ca</d:State>
  <d:Timestamp m:type="Edm.DateTime">2009-05-22T07:01:13.043Z</d:Timestamp>
  <d:Zip>94582</d:Zip>
</m:properties>
</content>
</entry>

```

In Listing 5-17, the HTTP verb specified is MERGE, to instruct the Table service that this is a Merge operation. The request body contains an ADO.NET entity set. The m:properties element define the property names and values of the entity. In the example, it represents a serialized ProAzureReader object as discussed earlier in this chapter. The If-Match header is a required condition the server checks before performing a conditional update. The ETag value of the entity is checked before making the update. For an unconditional update, its value should be a wildcard (*). A successful entity update returns an HTTP 1.1 204 (No Content) status code.

The data list in the Default.aspx page in the ProAzureReaderTracker_WebRole project has an UpdateUrl button that update the ReaderUrl property of an entity. Figure 5-14 shows the Default.aspx page.

Pro Azure Reader Tracker

Please enter your details and click Submit

Your Name/Email

Country

City

State

Zip

Book purchase location

Purchase Type

Purchase Date

Your Url (e.g. Facebook, LinkedIn, etc.)

Any feedback?

Submit

Filter Text

[By City](#) [By State](#) [By Country](#) [Today's Entries](#) [Top N \(50\)](#)

[By Purchase Date](#)

tejaswi from usa, ca, san ramon, 94582
 Url: http://www.bing.com
 purchased a New book at amazon.com on 6/21/2009 12:00:00 AM
 Has the following Feedback:
 Excellent Book
 http://www.bing.com
 UpdateUrl

q from usa, ca, san ramon, 94582
 Url:
 purchased a New book at web on 6/21/2009 12:00:00 AM
 Has the following Feedback:
 Good Book :) . But don't write again.

 UpdateUrl

Figure 5-14. *Default.aspx in the ProAzureReaderTracker_WebRole web role for the Merge Entity operation*

In Figure 5-14, you can update the ReaderUrl property of the ProAzureReader entity. The code for Merge Entity is in the UpdateUrl() method in the ProAzureReaderDataSource class, as shown in Listing 5-18.

Listing 5-18. UpdateUrl() Method

```

public void UpdateUrl(string PartitionKey, string RowKey, string url)
{
    var results = from g in dContext.ProAzureReader
                  where g.PartitionKey == PartitionKey
                      && g.RowKey == RowKey
                  select g;
    var e = results.FirstOrDefault<ProAzureReader>();
    e.ReaderUrl = url;

    dContext.MergeOption = MergeOption.PreserveChanges;

    dContext.UpdateObject(e);
    dContext.SaveChanges();
}

```

As shown in Listing 5-18, the `UpdateUrl()` method retrieves the appropriate entity using a LINQ query. Then, it sets the merge option to `PreserveChanges` and calls the `UpdateObject()` and `SaveChanges()` methods of the `DataServiceContext` object. The merge option instructs the `DataServiceContext` object to track entities in a specific manner locally. The possible options are as follows.

- *AppendOnly* instructs the `DataServiceContext` object to append entities to already-existing entities in the local cache. The existing entities in the cache aren't modified. This is the default option.
- *NoTracking* instructs the `DataServiceContext` object that entities aren't tracked locally. As a result, objects are always loaded from the server. The local values are overwritten with the server values.
- *OverwriteChanges* instructs the `DataServiceContext` object that server values take precedence over client values even if they have changed.
- *PreserveChanges* instructs the `DataServiceContext` object to preserve the local changes even if changes are detected on the server. The `DataServiceContext` object doesn't overwrite the property values but updates the ETag value with the server ETag value. Any properties not changed locally are updated with latest values from the server. In this option, the local changes to the properties aren't lost.

Figure 5-15 illustrates the sequence diagram for the Merge Entity operation.

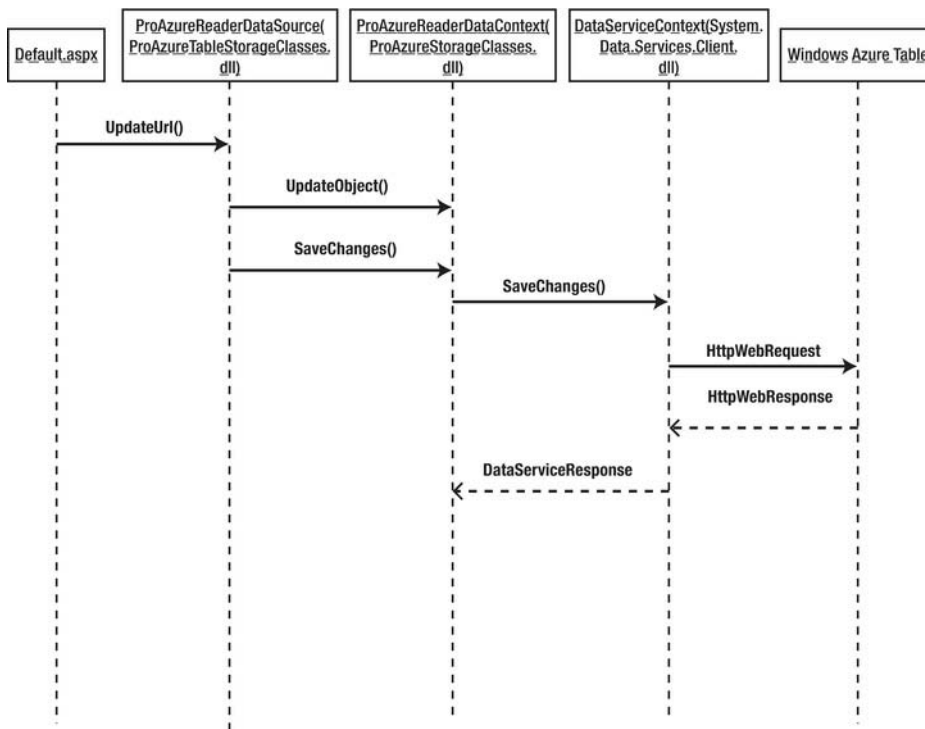


Figure 5-15. Merge Entity sequence diagram

As shown in Figure 5-15, the Default.aspx page calls the UpdateUrl() method on the ProAzureReaderDataSource object. The UpdateUrl() method calls the UpdateObject() and SaveChanges() methods on the ProAzureReaderDataContext object.

■ **Note** In Windows Azure 1.5 SDK, two new features were announced for the Table Storage service — Upsert and Query Projections. The Upsert feature allows you to update or insert a record on the same transaction and Query Projection allows you to query a subset of one or more entities. You can find more information on these features here: blogs.msdn.com/b/windowsazurestorage/archive/2011/09/15/windows-azure-tables-introducing-upsert-and-query-projection.aspx

Steve Marx has developed extension methods for these features because these features are not available in the Windows Azure 1.5 client SDK. blog.smarx.com/posts/extension-methods-for-the-august-storage-features

Storage Analytics

In Chapter 2, you learned how to log information from your compute nodes. This information is useful for finding out the overall usage, health, and data transfer occurring through the compute instances. This information is also useful for calculating the operating costs in more or less real time, so you don't get a surprised by the monthly bill. You can also use this information for throttling or charging your users for overage in a multi-tenant system. Previously, there was no way of tracking calls that were made directly to the Windows Azure storage because you could not intercept them. So, a lot of companies had to divert traffic either through compute instances or estimate the usage based on overall compute usage. In August 2011, Microsoft announced the Windows Azure Storage Analytics API. This API lets you to trace and analyze all the calls made to the Windows Azure storage service, including Blobs, Queues, and Tables. The Storage Analytics comprises of two features: Logging and Metrics. In Logging, you can trace the calls to the storage service, and the Metrics feature lets you capture the usage of your storage at an individual or aggregated basis.

Logging

The following are the characteristics of the Logging feature:

- Logging provides traces of all the calls made to the storage service

- The tracing information is saved as block blobs in a special blob container \$logs

- The log file format is as follows

- `<storage-service-name>/YYYY/MM/DD/hhmm/<counter>.log`

- e.g., `silverliningsstorage1/2011/08/26/1700/000001.log`

- The URL format for accessing the blob is

- `http://<storageaccountname>.blob.core.windows.net/$logs/silverliningsstorage1/2011/08/26/1700/000001.log`

- Each request is blogged to the file representing the hour of execution. For example, a request executed at 5:52 PM will be represented as

- `http://<storageaccountname>.blob.core.windows.net/$logs/silverliningsstorage1/2011/08/26/1700/000006.log`

- The 000006 indicates that there have been 6 entries in this hour so far.

- Each log entry consists of semi-colon separated fields listed here:

- `<version-number>;<request-start-time>;<operation-type>;<request-status>;<http-status-code>;<end-to-end-latency-in-ms>;<server-latency-in-ms>;<authentication-type>;<requestor-account-name>;<owner-account-name>;<service-type>;<request-url>;<requested-object-key>;<request-id-header>;<operation-count>;<requestor-ip-address>;<request-version-header>;<request-header-size>;<request-packet-size>;<response-header-size>;<response-packet-size>;<response-content-length>;<request-md5>;<server-md5>;<etag-identifier>;<last-modified-time>;<conditions-used>;<user-agent-header>;<referrer-header>;<client-request-id>`

- The descriptions of these fields are available here:

<http://msdn.microsoft.com/en-us/library/hh343259.aspx>

Each line is separated by a new line character '\n'

The following example shows one entry for a GetBlob request

```
1.0;2011-08-
26T17:52:46.5143711Z;GetBlob;AnonymousSuccess;200;18;10;anonymous;;silve
rliningstorage1;blob;"http://
silverliningstorage1.blob.core.windows.net/images/aaryandhruv.jpg?timeout=
20000";"/ silverliningstorage1/images/ aaryandhruv.jpg";z33rt405-4f67-48b5-
b033-b43sdf3346c3;0;123.100.2.10;2009-09-
19;252;0;265;100;0;;;"0x8CE1B6EA95033D5";Friday, 26-Aug-11 17:52:46
GMT;;;"8/26/2011 5:52:46 PM z33rt405-4f67-48b5-b033-b43sdf3346c3"
```

The log blob entries also contain metadata representing the type of operations contained in the log – LogType (read, write, delete), start time, end time, and log version – currently 1.0

The storage account administrator can read and delete log entries, but cannot create and update log entries because the \$logs is a reserved container for the Windows Azure storage service to dump logs

■ **Note** From the list of logged fields, you can programmatically analyze all the requests in detail and find out the performance, cost and success rate for all the storage service requests.

Metrics

Listed here are the characteristics of the Metrics feature:

- Metrics gives you aggregated information about requests and capacity of the storage account. Metrics information is statistical in nature and gives you hourly aggregates of requests and daily statistics on capacity information like space consumed, number of blobs and containers.
- Both the Logs and Metrics information is available via the Windows Azure Storage service REST API and managed API.
- Request (or Transaction) metrics are currently stored in the following Windows Azure Tables:
 - \$MetricsCapacityBlob: Represents the only capacity table currently supported. It contains two records created daily representing storage account capacity (currently only Blob storage capacity is supported) and \$logs container capacity.
 - \$MetricsTransactionsBlob: Contains hourly aggregates of service-level * and API-level aggregate data for transactions on Blob service.
 - \$MetricsTransactionsTable: Contains hourly aggregates of service-level and API-level aggregate data for transactions on Table service.

- `$MetricsTransactionsQueue`: Contains hourly aggregates of service-level and API-level aggregate data for transactions on Queue service.

The schemas for the metrics tables can be found here:

<http://msdn.microsoft.com/en-us/library/hh343264.aspx>.

■ **Caution** Storage Analytics is not enabled by default. Once you enable it, you will be charged for the space occupied and transactions handled by the Storage Analytics to your storage account. Because Microsoft changes billing information quite frequently, here is a link to the page that describes Storage Analytics billing in detail. blogs.msdn.com/b/windowsazurestorage/archive/2011/08/03/windows-azure-storage-analytics.aspx

Steve Marx has developed extension methods for storage analytics: blog.smarx.com/posts/analytics-leasing-and-more-extensions-to-the-net-windows-azure-storage-library

Enabling Storage Analytics

The Storage Analytics feature introduces two new REST operations: Set Storage Service Properties and Get Storage Properties. Unfortunately, at the time of writing this section, these two REST operations were not yet added to the .NET Storage Service managed API. But, like all other Storage service operations, you can call the REST operations directly. You must call the Set Storage Service Properties operation for enabling analytics on specific storage services. For example, you have to make a separate call to the Blob, Table and Queue services for enabling storage analytics on each. Listing 5-19 shows the REST header and body format of the request to enable storage analytics in the Blob service.

Listing 5-19. Enable Storage Analytics Request for Blob Service

```
PUT http://silverliningsstorage1.blob.core.windows.net/?restype=service&comp=properties
HTTP/1.1
```

```
x-ms-version: 2009-09-19
x-ms-date: Tue, 26 Aug 2011 05:52:19 GMT
Authorization: SharedKey
silverliningsstorage1:Z1lSsffsr35gUYQ1uucdsXk6/iDsse53g5sUE=
Host: silverliningsstorage1.blob.core.windows.net
```

```
<?xml version="1.0" encoding="utf-8"?>
<StorageServiceProperties>
  <Logging>
    <Version>1.0</Version>
    <Delete>true</Delete>
    <Read>false</Read>
    <Write>true</Write>
    <RetentionPolicy>
      <Enabled>true</Enabled>
```

```

        <Days>14</Days>
      </RetentionPolicy>
    </Logging>
    <Metrics>
      <Version>1.0</Version>
      <Enabled>true</Enabled>
      <IncludeAPIs>true</IncludeAPIs>
      <RetentionPolicy>
        <Enabled>true</Enabled>
        <Days>30</Days>
      </RetentionPolicy>
    </Metrics>
  </StorageServiceProperties>

```

The REST Request header is similar to all the REST request headers you have seen so far in this book. The REST Request body is in XML format and defines two main elements: Logging and Metrics. The request body in Listing 5-19 enables analytics for Delete and Write operations and specifies a retention policy of 14 days, after which the analytics information will be deleted. The request also enables aggregate metrics, API-level summary statistics, and a retention policy of 30 days. Once analytics are enabled, you can get the properties by calling Get Storage Service Properties and Update analytics using Set Storage Service Properties again. The XML format of the Get Storage Service Properties return body is the same as the Request body of the Set Storage Service Properties operation.

■ **Note** C# examples for reading Logging and Metrics data is available from Windows Azure Storage Team Blog [here](http://blogs.msdn.com/b/windowsazurestorage/archive/2011/08/03/windows-azure-storage-logging-using-logs-to-track-storage-requests.aspx)

<http://blogs.msdn.com/b/windowsazurestorage/archive/2011/08/03/windows-azure-storage-logging-using-logs-to-track-storage-requests.aspx>

and here

<http://blogs.msdn.com/b/windowsazurestorage/archive/2011/08/03/windows-azure-storage-metrics-using-metrics-to-track-storage-usage.aspx>

respectively.

Table Storage versus SQLAzure

“How do I decide whether to use SQLAzure or Table storage?” I get this question at least once in every Architecture Design Session that I conduct. My answer is, as you expected, “it depends.” Both the

storage services have distinct advantages over the other in specific scenarios. I usually adhere to the following guidelines in deciding which storage option to use.

For any new application development, consider Table storage as a viable option for storing data. You will be surprised to find out the number of tables that really do not require any normalization.

- **Migration:** For any migration scenario, if your on-premises data resides in a relational database, SQLAzure will make your migration quick and easy with minimal or no code changes.
- **Cost:** Table Storage is much cheaper than SQLAzure from a pure storage cost perspective (including transaction costs). Even if you denormalize data with duplicate data in multiple tables, it will still be cheaper than SQLAzure in most of the cases.
- **Stickiness:** Table storage is proprietary to Windows Azure platform. Once you put your data in there, you will have to invest in tools to bring the data back on-premises in the future. With SQLAzure, you can bring you data back in no time.
- **Utility Tables:** I regularly recommend using Table storage for utility tables that do not have relational requirements. Examples of utility tables are ecommerce transactions, user profiles, tracing information, session, and the like. These tables don't have stringent normalization requirements and each record can atomically represent an event.

If your data model has a well-defined relational model, SQLAzure is the right choice, as it will save you a lot of time in maintaining referential integrity across tables and map that to an object relational model like Entity Framework, NHibernate or Typed Datasets.

In the next section, we will look at two most common scenarios: reading performance counters from table storage and paging in table storage.

Table Service Scenarios

Now that you understand Windows Azure Table storage concepts, let's look at some of the commonly used scenarios. Reading performance counters from Table storage shows you how to read the performance counters data from Table storage that are stored by the diagnostics service running in compute nodes of your service. The second scenario shows paging of data sets when using Windows Azure Table storage.

Scenario 1: Reading Performance Counters from Table Storage

In Chapter 2, you learned that the performance counter logs from the Windows Azure Compute instances are periodically transferred to the Table storage and stored in the `WADPerformanceCountersTable` table. For retrieving these logs, you have to use the Table storage REST API. The structure of the `WADPerformanceCountersTable` is illustrated in Figure 5-16.

PartitionKey	RowKey	Timestamp	EventTickCount	DeploymentId	Role	RoleInstance	CounterName	CounterValue
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804579526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\ASP.NET Applications(_Total_)/Requests/Sec	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804579526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\ASP.NET Applications(_Total_)/Requests Total	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804584526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\Processor(_Total)/% Processor Time	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804584526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\Memory\Available MBytes	1219
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804584526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\ASP.NET\Request Execution Time	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804584526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\ASP.NET Applications(_Total_)/Requests/Sec	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804584526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\ASP.NET Applications(_Total_)/Requests Total	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804589526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\Processor(_Total)/% Processor Time	2,492,511
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804589526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\Memory\Available MBytes	1218
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804589526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\ASP.NET\Request Execution Time	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804589526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\ASP.NET Applications(_Total_)/Requests/Sec	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804589526	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\ASP.NET Applications(_Total_)/Requests Total	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804594525	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\Processor(_Total)/% Processor Time	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804594525	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\Memory\Available MBytes	1217
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804594525	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\ASP.NET\Request Execution Time	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804594525	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\ASP.NET Applications(_Total_)/Requests/Sec	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804594525	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\ASP.NET Applications(_Total_)/Requests Total	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804599525	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\Processor(_Total)/% Processor Time	0
063426804540x	2c4337b9971e	12/1/2010 12:59	63426804599525	2c4337b9971e40de	WebRole1	WebRole1_JN_0	\Memory\Available MBytes	1218

Figure 5-16. WADPerformanceCountersTable

Figure 5-16 shows that data from multiple deployments and roles are stored in the same table therefore you have to query based on Deployment ID for retrieving data for a particular deployment. And to further filter the query, you may also include a time range and the roles you are interested in. Listing 5-20 shows structure of the PerformanceData class that inherits from the TableServiceEntity class and the PerformanceDataContext class that inherits from the TableServiceContext class. This class will represent the returned object on the query.

Listing 5-20. WADPerformanceCountersTable Structure

```
public class PerformanceData : Microsoft.WindowsAzure.StorageClient.TableServiceEntity
{
    public Int64 EventTickCount { get; set; }
    public string DeploymentId { get; set; }
    public string Role { get; set; }
    public string RoleInstance { get; set; }
    public string CounterName { get; set; }
    public double CounterValue { get; set; }
}

public class PerformanceDataContext : TableServiceContext
{
    public IQueryable<PerformanceData> PerfData
    {
        get
        {
            return this.CreateQuery<PerformanceData>("WADPerformanceCountersTable");
        }
    }

    public PerformanceDataContext(string baseAddress, StorageCredentials credentials)
        : base(baseAddress, credentials)
    {
    }
}
```

```
}
```

I have not included the Partition Key and the Row Key in the class because they are of no value considering that we are only interested in the counter values over a period of time. After you have created the queryable `PerfData` property on the `PerformanceDataContext` class, you can then execute LINQ queries against the `PerfData` property for retrieving values. Listing 5-21 shows two query functions that retrieve the average memory usage and the average processor percentage over a given time range for the specified deployment.

Listing 5-21. Retrieve Memory and Performance Usage data

```
public static double GetAverageProcessorTime(string deploymentId, string storageAccountName,
string storageKey, int timeFrameInMinutes)
{
    try
    {
        var account = new CloudStorageAccount(new
StorageCredentialsAccountAndKey(storageAccountName, storageKey), true);
        var context = new PerformanceDataContext(account.TableEndpoint.ToString(),
account.Credentials);
        var data = context.PerfData;
        DateTime tf =
DateTime.UtcNow.Subtract(TimeSpan.FromMinutes(timeFrameInMinutes));

        System.Collections.Generic.List<PerformanceData> selectedData = (from d in
data
                                where
d.CounterName == @"\"Processor(_Total)\"% Processor Time"
                                &&
d.DeploymentId == deploymentId
                                &&
(DateTime.Compare(tf, d.Timestamp) < 0)
                                select
d).ToList<PerformanceData>());

        return (from d in selectedData
                where d.CounterName == @"\"Processor(_Total)\"% Processor Time"
                select d.CounterValue).Average();

    }
    catch (System.Exception ex)
    {
        throw ex;
    }
}
```

```

    public static double GetAverageMemoryUsageInMbytes(string deploymentId, string
storageAccountName, string storageKey, int timeFrameInMinutes)
    {
        try
        {
            var account = new CloudStorageAccount(new
StorageCredentialsAccountAndKey(storageAccountName, storageKey), true);
            var context = new PerformanceDataContext(account.TableEndpoint.ToString(),
account.Credentials);

            var data = context.PerfData;

            DateTime tf =
DateTime.UtcNow.Subtract(TimeSpan.FromMinutes(timeFrameInMinutes));

            System.Collections.Generic.List<PerformanceData> selectedData = (from d in
data
d.CounterName == @"\Memory\Available MBytes"                                where
d.DeploymentId == deploymentId                                             &&
(DateTime.Compare(tf, d.Timestamp) < 0)                                     &&
d).ToList<PerformanceData>());                                             select

            if (selectedData.Count > 0)
            {
                return (from d in selectedData
                        where d.CounterName == @"\Memory\Available MBytes"
                        select d.CounterValue).Average();
            }
            else
            {
                return 0;
            }
        }
        catch (System.Exception ex)
        {
            throw ex;
        }
    }
}

```

In Listing 5-21, the LINQ query retrieves all the records for the specified performance counter that have timestamps between now and the specified time frame. You can find this code in the Service Management API Windows client application from the Chapter 2 source code.

Scenario 2: Paging in Table storage

Web user interfaces are not designed to display a lot of records at the same time in a listing format. In the cloud, you may have to display data directly from the Table storage into an ASP.NET web page. Windows Azure Table storage also has the limit of returning no more than 1000 results for your query even if more than 1000 entities fit the query. The response consists of a continuation token in the header indicating that there are more entities that match the query criteria. The `CloudTableQuery<TElement>` generic class returns a continuation token for the query results. The asynchronous methods for retrieving and using the continuation token is shown in Listing 5-22.

Listing 5-22. Continuation Token

```
private IAsyncResult BeginAsyncOperation(object sender, EventArgs e, AsyncCallback cb,
object extradata)
{
    var query = new
ProAzureReaderDataContext(CloudConfiguration.GetStorageAccount()).ProAzureReader.Take(3).AsTableServiceQuery();
    if (Session["segment"] == null)
    {
        return query.BeginExecuteSegmented(cb, query);

    }
    else
    {
        var segment = Session["segment"] as ResultSegment;
        return query.BeginExecuteSegmented(segment, cb, query);
    }
}

private void EndAsyncOperation(IAsyncResult result)
{
    var tableQuery = result.AsyncState as CloudTableQuery<ProAzureReader>;
    ResultSegment<ProAzureReader> resultSegment =
tableQuery.EndExecuteSegmented(result);

    Session["segment"] = resultSegment.ContinuationToken;
    //this.readers = resultSegment.Results.ToList();
}
```

In Listing 5-22, an asynchronous query is executed on the Table storage using the `CloudTableQuery.BeginExecuteSegmented()` method. This method returns the `ResultSegment` object when the

asynchronous method ends. The continuation token is a property of the `ResultSegment` class. The continuation token is saved in the user's session object and used in the next query. Typically, the continuation token will be associated with the user's paging action for viewing results on the next page.

■ **Tip** When executing a query against Table storage, there might be cases where the connection is throttled and your query receives an exception. The type of exception returned by the Storage service indicates whether the query can be retried or not. All the exceptions with code ≥ 400 and < 500 , and 501 and 505 are exception codes that does not allow retry on the query. All other exceptions can be retried using either the pre-defined retry policies in the Windows Azure StorageClient API or creating a custom retry policy⁶.

`RetryPolicies.NoRetry`: No retry is used.

`RetryPolicies.Retry`: Retries N number of times with the same backoff between each attempt.

`RetryPolicies.RetryExponential` (Default): Retries N number of times with an exponentially increasing backoff between each attempt.

All the three storage types: Blobs, Queues, and Tables have `RetryPolicy` property in their client class.

■ **Note** More information on retries can be found on the Windows Azure Team Blob site (<http://blogs.msdn.com/b/windowsazurestorage/archive/2011/02/03/overview-of-retry-policies-in-the-windows-azure-storage-client-library.aspx>).

You can use Table storage for storing ASP.NET Membership, Roles, Profiles, and Session State using the `AspProviders` and `AspProviders` Demo samples that Microsoft has made available through [code.msdn.com](http://code.msdn.microsoft.com/windowsazuresamples) <http://code.msdn.microsoft.com/windowsazuresamples>.

⁶ Windows Azure Storage Team Blog, <http://blogs.msdn.com/b/windowsazurestorage/archive/2011/02/03/overview-of-retry-policies-in-the-windows-azure-storage-client-library.aspx>

Summary

In this chapter, you learned the details in interacting with the Windows Azure Table Storage service programmatically. The Table service provides a structured storage that you can use from anywhere, anytime. If you've already worked with REST, the ADO.NET Data Services Framework, or the ADO.NET Entity Framework, you should find the Table service concepts easy to understand and program with. The knowledge you've acquired from learning Table service REST API, ADO.NET Data Services Client library, and Windows Azure SDK StorageClient should enable you to build your own Table service applications.

This concludes a series of three intense chapters covering the wide range of features and functionality offered by the Blob, Queue, and Table services. You learned in detail about the three types of storage services offered by Windows Azure. You also learned Drives as a special type of Blob storage. The demonstrations and API discussions in these chapters should enable you to use the concepts in your own storage applications. Now, you are also better prepared to choose the right combination of storage services for your solution. Later in the book I have covered the relational database in the Windows Azure called SQLAzure. Make sure you leverage the best possible combination of non-relational, structured and relational storage for your solution after reading these chapters.

The next few chapters cover the Middleware layer of the Windows Azure Platform called the Windows Azure AppFabric.

Bibliography

MSDN. (n.d.). *ADO.NET Data Services Specification*. Retrieved from MSDN Developer's Network:

<http://msdn.microsoft.com/en-us/library/cc668808.aspx>.

MSDN. (2009, May). *Windows Azure Blob — Programming Blob Storage*. Retrieved from MSDN:

<http://go.microsoft.com/fwlink/?LinkId=153400>.

MSDN. (2009, May). *Windows Azure Queue — Programming Queue Storage*. Retrieved from MSDN:

<http://go.microsoft.com/fwlink/?LinkId=153402>.

MSDN. (2009, May). *Windows Azure SDK*. Retrieved from MSDN: [http://msdn.microsoft.com/en-](http://msdn.microsoft.com/en-us/library/dd179367.aspx)

[us/library/dd179367.aspx](http://msdn.microsoft.com/en-us/library/dd179367.aspx).

MSDN. (2009, May). *Windows Azure Table — Programming Table Storage*. Retrieved from MSDN:

<http://go.microsoft.com/fwlink/?LinkId=153401>.

Windows Azure Storage Team Blog: <http://blogs.msdn.com/b/windowsazurestorage/>

VM Role and Windows Azure Connect

As two of the latest additions to the Windows Azure platform, it's hard to think of any more widely anticipated features than VM role and Windows Azure Connect. However, they may also be the most misunderstood. In this chapter, I will help you understand why these features were released, what their purpose is, and the scenarios to which they do and do not apply.

■ **Note** As of the time of writing, both of these technologies are technically still in beta. Thus, they are not supported for production use. For announcements regarding production availability for this and other Azure features, go to <http://blogs.msdn.com/windowsazure>.

The VM role is complementary to the Web and Worker roles. It provides the ability to deploy a Windows Server 2008 R2 Hyper-V-enabled virtual machine to your hosted service. Essentially, you will now have the ability to run your own images in the Azure environment.

Windows Azure Connect provides IPsec protected connections between on-premise machines and cloud role instances. This provides a higher level of integration than previously offered by the AppFabric Service Bus.

VM Role

The first thing to note about the VM role in its current form is that it is not considered infrastructure as a service. In other words, it's not intended to be used in the same way as Amazon Web Services EC2 or other IaaS providers such as Rackspace. This makes sense when you consider the Azure Fabric—the Fabric Controller can bring down any role instance at any time, for many good reasons. In addition, the nodes sit behind a round-robin load balancer. This is what makes Azure strong as a PaaS provider. However, this also means that in order to maintain the Azure SLA, you will need at least *two* instances of your role deployed. This has several architectural impacts, including the following:

- **State management.** Because there are two instances running behind a round-robin load balancer, you will need to store runtime state information outside the VM. Windows Azure Caching is a good candidate for this.

- **File management:** Files will also need to be maintained outside the VM. Because the load balancers route requests in a round-robin fashion, the file systems on each compute instance will quickly become dissimilar and fragmented unless files are stored centrally. If the files are read-only files, then Windows Azure Drives are a great solution. Otherwise, consider using Azure blob storage.

So, what this means today is that VM role is not well suited to running stateful server applications. Obviously, you wouldn't want to install Active Directory to a VM role, but applications like SQL Server and Oracle are also not good candidates at this point.

VM Role Benefits/Tradeoffs

Using VM role provides you the ability to have more control over your image. You can set it up exactly the way you want, install any software necessary, and configure services that you might need. In addition, you still get the benefits of Azure-provided load balancing, failover, and redundancy.

However, once you use the VM role, you are responsible for maintaining the image yourself. This means you have to perform upgrades to the operating system, and patches. If you have an application update, you will need to build and test the new images, rather than simply deploying an application package to the Azure Service Management portal. This also complicates your development cycles. Before, your application developers could test their work by simply deploying their package to the compute emulator. Now there's an extra layer of complexity brought into the development cycle.

Scenarios

There are several scenarios in which VM role makes perfect sense. One would be an application that requires many other products to be installed to run correctly. While you could use Startup Tasks to install this required software, the install will take time. This time increases the amount of time required to spin up the Azure role instances. This has several impacts, most notably that whenever one of your VMs are taken down by the fabric controller, it will take longer for that VM to be restored, resulting in longer periods of diminished capacity.

If you were to pre-install all of this software on a VHD image, then you won't have to wait for the installers to run—the software will be already installed.

■ **Note** You might assume that by handling OS upgrades and patches yourself in VM role, that the application would gain significantly more uptime, as the fabric controller would not need to install these updates. This is only partially true, in that the guest OS would not need to be updated. However, the instance can still be taken down for a variety of reasons, including hardware failure, Host OS updates, or even to gain hardware efficiencies in the data center.

There is another scenario worth mentioning related to installing software. If your application requires third-party software, and the provider didn't write its installer to run in silent or unattended mode, then you need to use the VM role. You can install the software manually on your image, then deploy your image to the cloud.

There are other scenarios, such as complications involving SysWow64 and 32-bit DLLs, but the takeaway from this section is to carefully consider the use of the VM role. It isn't intended to support

IaaS, and you should take this into consideration, especially when migrating an existing application to the cloud. Some applications, such as single-server, on-premises applications are not good candidates for the cloud without significant changes, whether you use VM role or Web/Worker roles. VM role is not always a shortcut to getting to the cloud.

At a high level, there are the following three steps to deploying a VM to Windows Azure:

1. Create a base image in Hyper-V.
2. Apply sysprep.exe.
3. Upload to Windows Azure.

I discuss these in the following sections.

Creating the Virtual Machine

The first step is to build your base VM image. You'll need a Windows 2008 R2 Server with Hyper-V enabled, and either a disk or .iso image for a Windows Server 2008 R2 (along with a valid license key, of course). You will also want to either install the Windows Azure SDK 1.4 to this server, or copy the following file to your server hard drive:

```
C:\Program Files\Windows Azure SDK\v1.4\iso\wavmroleic.iso
```

This assumes you are running SDK version 1.4 and have installed to the default directory. If you installed to another directory, or are using another version, adjust the path accordingly.

■ **Note** The steps to setting up Hyper-V and creating a base virtual machine image are beyond the scope of this book. I will be focusing on any steps that are specific to building and deploying to Windows Azure. The Windows Azure Platform Training Kit has an excellent lab that will take you step by step through this process.

Using Hyper-V Manager, build your base image. The key step in this process that is different is to pay attention to how much memory you assign to the image. It's important that this space is within the disk space allocated for the size of the Azure VM you plan to deploy. For example, the Medium size VM has a 500GB disk size limit. You obviously wouldn't want to allocate 750 GB to your image in this case, unless you are planning on deploying to a Large VM, which would increase the cost of the application.

After installing the OS, add the roles you need for the server (such as IIS), and features such as .NET Framework, Remote Administration, and enabling Remote Desktop access if you want to be able to remote into your instance. You'll also want to install all other server application components necessary to run your application, such as third-party tools, services, and the code required to run your application.

Windows Azure VM Role Integration Components

The Windows Azure VM Role Integration components are required to be installed on the image. These are components that start each time the operating system starts, and they perform tasks that enable the VM to run in the Azure environment. Essentially, the components provide several functions – preparing

the VM for deployment, installing runtime APIs that provide coordination between the VM and the Azure Fabric, installing certificates, and creating local storage resources.

System Preparation

Included with the Integration Components is a System Preparation Tool (`sysprep.exe`). This tool runs the VM you have created through a specialized setup when the instance is initialized. In order to automate this, a file (`C:\unattend.xml`) is installed in the root directory of the VM role instance. This file provides the Windows configuration settings for this specialization phase, and includes the following:

- System locale to en-US
- Time zone to UTC (which must not be changed)
- Turning off Windows Update
- Setting the Administrator password

Provide Coordination Between Your Image and the Windows Azure Fabric

Several service runtime APIs are also installed. These allow the VM to communicate with the Azure environment. This is what enables code from the `Microsoft.WindowsAzure.ServiceRuntime` namespace to perform actions such as retrieving the IP addresses of all other role instances, getting the instance ID, and getting service configuration information. It also enables the VM to communicate its instance state through the load balancer. This is critical to ensuring the fabric controller can determine when a VM is unhealthy and needs to be recycled.

■ **Note** Code that uses the service runtime API needs to be running under an administrator or `LocalSystem` account.

Install Certificates

Remember that certificates are specified in the service definition file and uploaded through the Azure portal? Well, this means that they somehow need to get installed on the VM. The Windows Azure Integration Components handle this for you. All certificates are installed to the `LocalMachine` store location.

If you're thinking that it's easier to install the server certificate directly on the server image, think again. The `sysprep` tool, in the process of generalizing your image, destroys private key information.

■ **Note** For more information on using management certificates with a VM role, go to <http://msdn.microsoft.com/en-us/library/gg697586.aspx>.

Create Local Storage Resources

In your service definition, you have the ability to define local storage on the VM. Because this is in the service definition and not on the VM, the Integration Components must read the service model through the service runtime API, and then allocate the local storage resources as necessary.

Installing the Integration Components

In order to install the components, you will need the `wavmroleic.iso` file, which can be found in the Windows Azure SDK as described previously, or copied to your server hard drive (see Figure 6-1). Here are the steps:

1. While connected to your VM on your server, choose Media → DVD Drive → Insert Disk.
2. Browse to the .iso file, select it, and click Open.
3. This will mount the ISO as a DVD drive, which will bring up the AutoPlay dialog. Double-click `WaIntegrationComponents-x64.msi` to begin the installation.

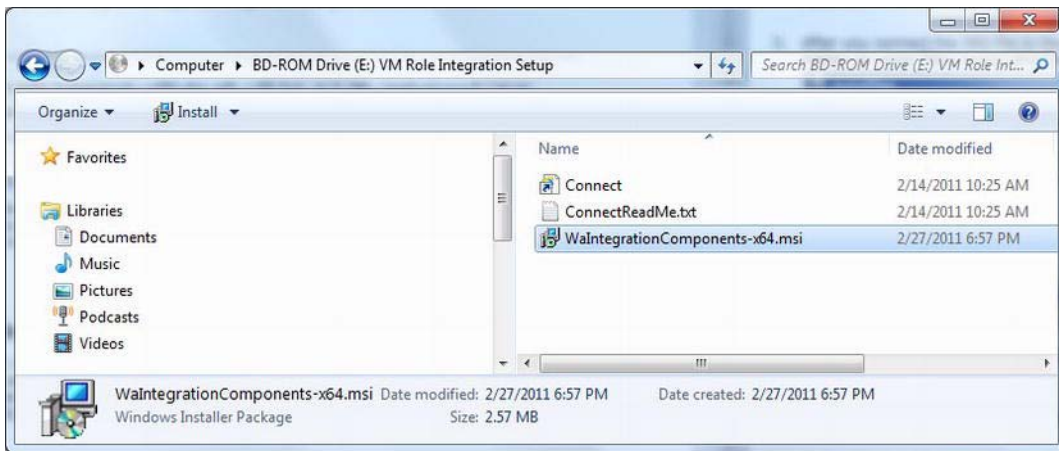


Figure 6-1. Windows Azure integration components install file

4. Run the install wizard. You will need to provide the Administrator password. This is used when the operating system starts, after the image is prepared and deployed to Windows Azure.
5. When the install is complete, you will be asked to restart the system.

■ **Note** It's probably a good idea to create a password-reset disk, in case you ever lose the Administrator password.

System Preparation Tool

Once the VM is restarted, log in and run the System Preparation Tool, which is located at %windir%\system32\sysprep\sysprep.exe. Set the options as shown in Figure 6-2.

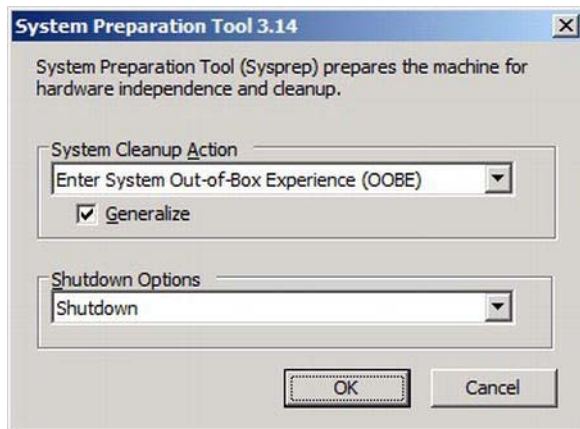


Figure 6-2. System Preparation Tool settings

The sysprep tool will prepare the image by cleaning up specifics such as user and machine settings and log files, private key information, and hardware-dependent information.

Upload Image to Windows Azure

Finally, we are ready to upload the image to Windows Azure! In order to upload to the Windows Azure environment, we will use the `cupload.exe` command-line tool that is provided as part of the Windows Azure SDK. Once the image is uploaded, we will create the Visual Studio project that contains the service definition

From the start menu, navigate to Windows Azure SDK 1.4 and open the Azure SDK Command Prompt as an Administrator. Here, we will execute `cupload.exe`, telling it to execute a PowerShell cmdlet that will upload the vhd file:

```
cupload Add-VMImage -Connection "SubscriptionId=<YOUR-SUBSCRIPTION-ID>;
CertificateThumbprint=<YOUR-CERTIFICATE-THUMBPRINT>" -Description "<YOUR DESCRIPTION>" -
LiteralPath "<PATH-TO-VHD-FILE>" -Name <NAME.VHD> -Location <HOSTED-SERVICE-LOCATION>
```

The subscription ID is your Azure subscription ID. This can be obtained from the Management Portal. The certificate thumbprint is the thumbprint from the management certificate you generated and uploaded to the management portal. Add your own description, the path to your VHD, and the name of the VHD, and then the name of the Windows Azure datacenter where the VHD will be hosted. As of the

time of writing, available options are “East Asia,” “North Central US,” “North Europe,” “South Central US,” “Southeast Asia,” “West Europe.”

Once execution starts, the Windows Azure VHD Verification Tool will run. This tool will mount the VHD and verify it, then compress it into an even smaller copy, with a .preped extension. It will then begin uploading the image, which can take a long time, due to the typically large file size and connection speed.

Viewing Image in Management Portal

To verify that the image was uploaded successfully, Open your Management Portal, click Hosted Services, Storage Accounts & CDN, and then click the VM Images folder on the left, as shown in Figure 6-3. You should see a list of your subscriptions, along with any VMs you have uploaded.

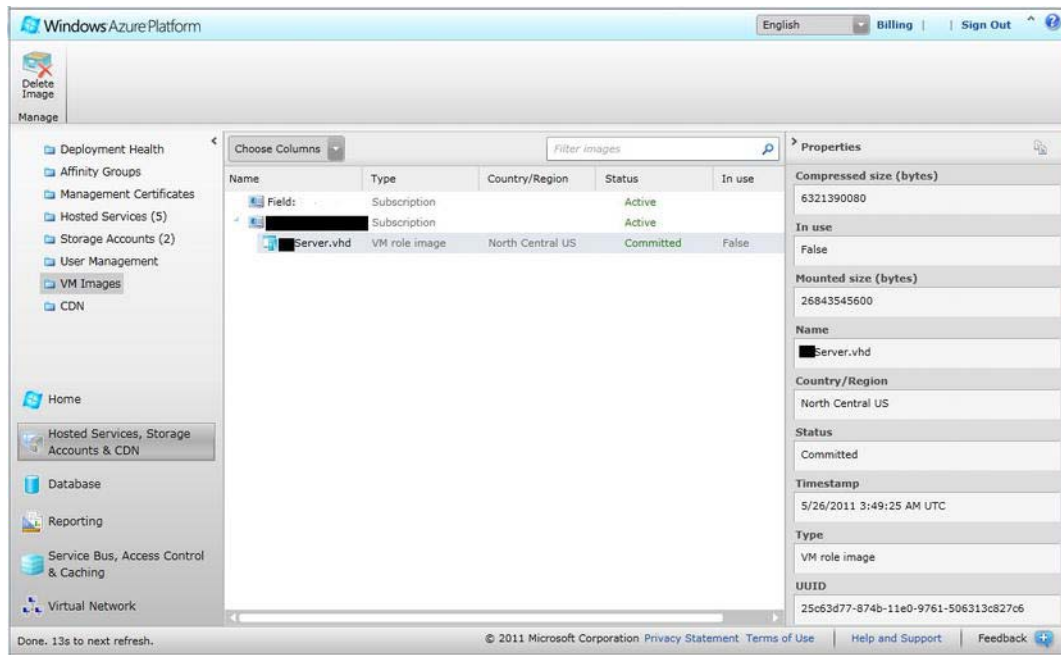


Figure 6-3. VM image in Management Portal

Creating the Hosted Service and Service Definition

Congratulations! You have successfully uploaded a VHD to Windows Azure! But wait... we're not done yet. We need a hosted service in which to deploy the image, which means we need to create a deployment package and configure it to reference the image we just uploaded. Here are the steps:

1. Open Visual Studio and create a new Windows Azure Project, but in this case do not add any roles. Just click OK when the Add Roles dialog appears.

2. Once the solution is created, right-click the Roles folder in Solution Explorer, and choose Add → New Virtual Machine Role.

■ **Note** If the New Virtual Machine Role option is not available, you may need to change a registry setting (remember, this is still technically in beta). This link can help: <http://social.msdn.microsoft.com/Forums/en-US/windowsazuretroubleshooting/thread/84c61a84-89c1-4fef-8d7b-e6419e8c4339>.

3. Double-click the role you just created in Solution Explorer to begin configuring the Hosted Service model. Visual Studio will need access to your Azure subscription, to retrieve the list of VMs that have been uploaded and are available.
4. Select the credentials that you have uploaded to the portal. In the Virtual Hard Disk tab, choose the credentials you used when uploading the VM, and then you will be provided with a drop down listing the VHDs you have uploaded.
5. Choose the appropriate VHD for this hosted service (see Figure 6-4).

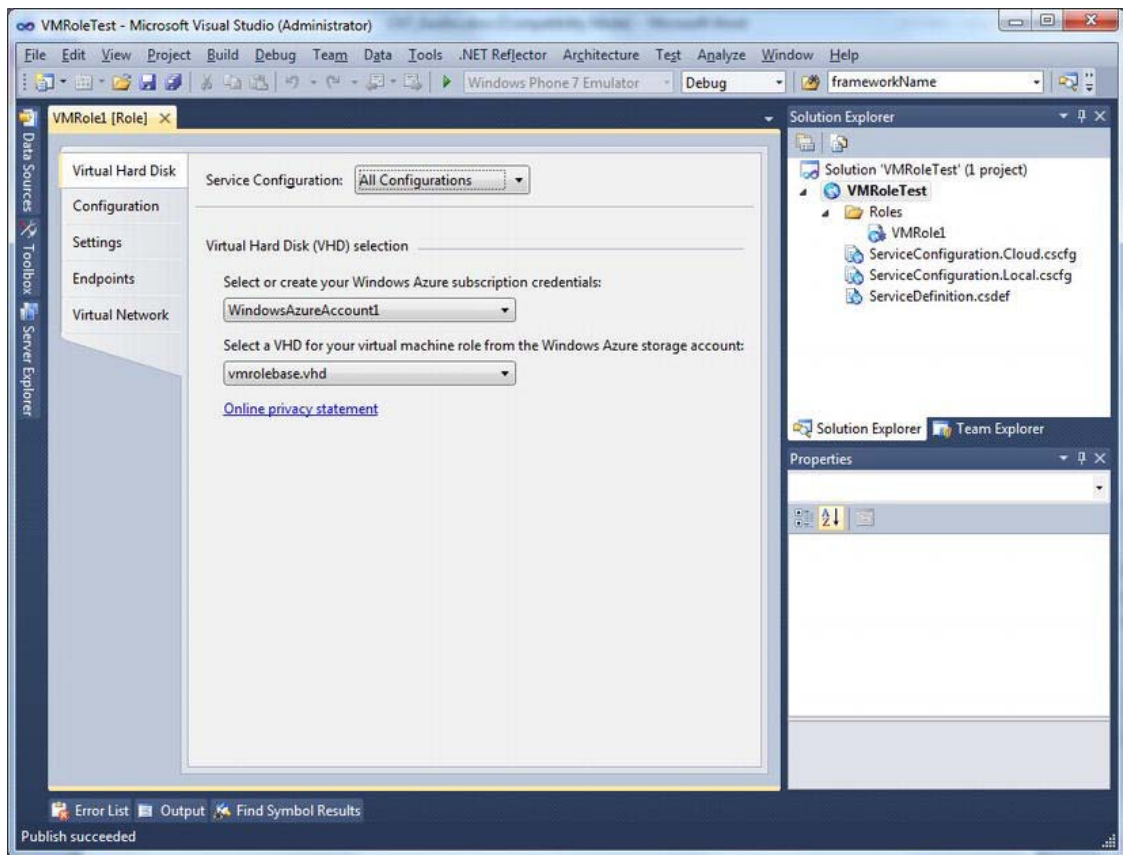


Figure 6-4. Selecting a VHD in Visual Studio

■ **Note** If you have not yet uploaded management certificates to the portal, go to http://msdn.microsoft.com/en-us/wazplatformtrainingcourse_vmrolelab_topic6 for additional information.

6. Next, we need to configure endpoints. Go to the Endpoints tab and add the endpoints required for your application, such as HTTP and HTTPS, plus any TCP ports you need opened (see Figure 6-5). Keep in mind there is a 5-port limit, and that Remote Access will also consume one of your ports.

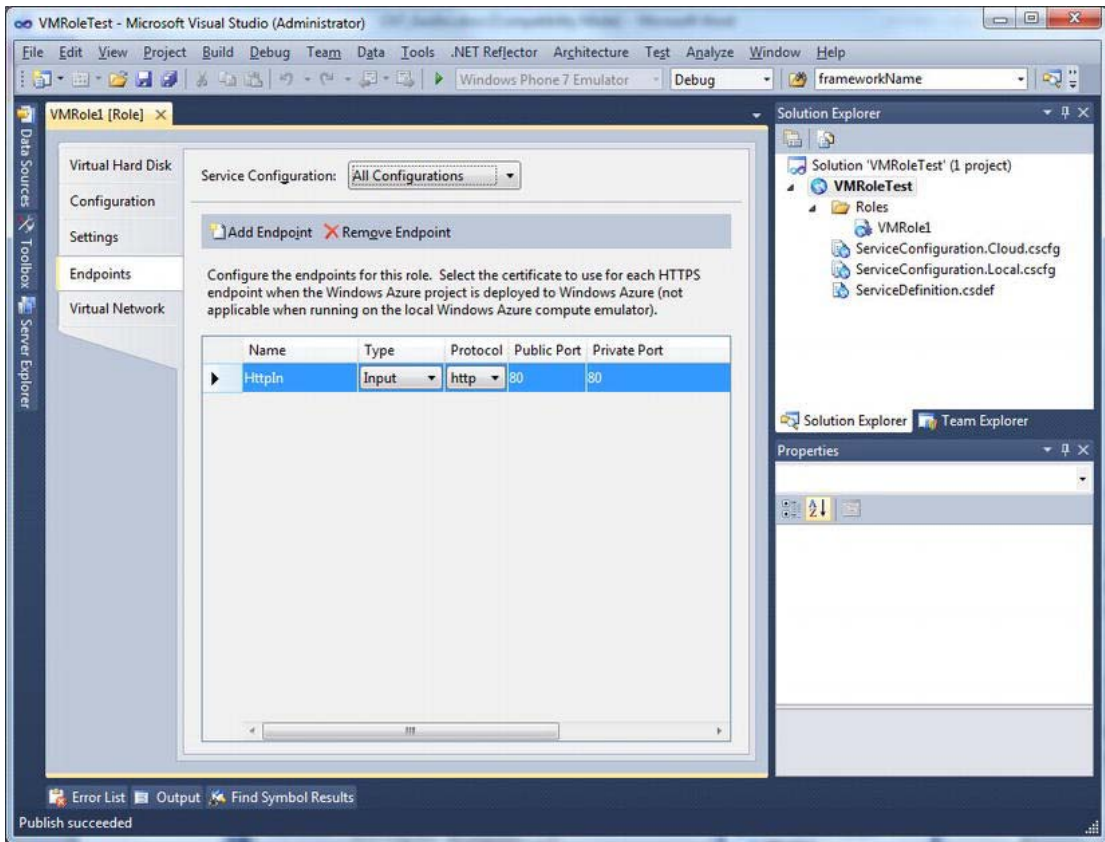


Figure 6-5. Configuring endpoints in Service Definition

Once we have completed these tasks, the final steps are to package the role and (if desired) enable remote access so we can view the instance remotely once deployed.

Packaging and Enabling Remote Access

Packaging our VM role for deployment is not different than doing the same for a Web or Worker role. Right-click the Cloud Service project and select Publish. To enable remote access, click the Configure Remote Desktop Connections link, and enter your credentials, and the Remote Access password, which is either the Administrator password, or another identity you created in the VM for remote access.

Deploying the Hosted Service

To deploy the hosted service, we simply go back to the management portal and deploy our package exactly the same as a Web or Worker role. Navigate to Hosted Services, create or select the hosted service in which to deploy, and deploy your project.

Once these steps are complete, you should have your VM role application running on Windows Azure!

Next, we will take a look at Windows Azure Connect—a handy tool, for instance, if you need to connect your VM to a server running on premises.

Windows Azure Connect

While the Azure product team doesn't refer to Windows Azure Connect as a VPN, I can't really come up with a more accurate comparison. Windows Azure Connect provides an IPsec protected connection between disconnected machines. This could be on-premise machines, or Web/Worker/VM role instances in separate hosted services. As you might expect, the possibilities for hybrid applications becomes mind-boggling, as shown in Figure 6-6.

In the next few sections, we will look at how Windows Azure Connect is different from the Windows Azure AppFabric Service Bus, as well as how to provision and activate Windows Azure Connect endpoints, creating a solution that networks could instances to on-premises machines.

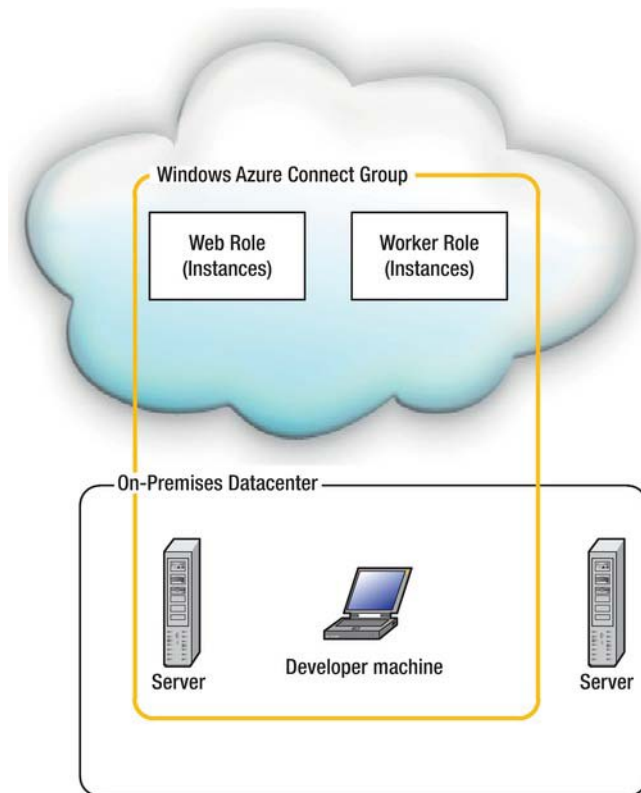


Figure 6-6. Windows Azure Connect group

Windows Azure Connect vs. Service Bus

Connect enables hybrid scenarios and Service Bus enables hybrid scenarios. So how are they different, and when is it appropriate to use one over the other? I like to think of Service Bus as an application-level integration utility, while Connect is a machine-level integration utility.

With Service Bus, you need to build proxies on each side of the firewall, and the applications use a relay service to communicate. If your scenario is one application talking to another application, this is highly appropriate.

With Connect, you have access to the entire machine as if it were in your datacenter. This means you can operate in a more familiar way, such as integrating System Center Operations Manager for monitoring, connecting to a database on-premises, or accessing functionality of legacy systems that could not be ported to the cloud.

Connect brings tremendous power and flexibility. However, keep the following in mind:

1. **Network latency.** Keeping your database on-premises and using Connect is very tempting, especially in regulatory compliance scenarios. However, the distance between your datacenter and the Azure datacenter matters. Performance may suffer. However, if the benefits outweigh the performance cost, then go with Connect.
2. **Bandwidth costs.** Even though the servers are interacting as if they are in the same datacenter, they are not. Even though pricing has not yet been announced for Connect, keep in mind that you are charged for all bandwidth coming out of the datacenter. No matter what, you would always want to make sure you are using bandwidth as efficiently as possible.

■ **Note** For a real-world example that implemented Connect as a replacement for Service Bus, go to <http://www.microsoft.com/windowsazure/learn/real-world-guidance/field-notes/integrating-onpremises-using-connect/>.

Provisioning Windows Azure Connect

The first step in provisioning Windows Azure Connect is to ensure you have access to the CTP. To do so, follow these steps:

1. In the Windows Azure Management Portal, click the Home tab, then the Beta Programs folder in the left-hand navigation.
2. There you will be given the option to request CTP access. Note that this process will change once Connect is in production, you will not need to request access.
3. Once access has been granted, you will be able to access Connect functionality through the Virtual Network tab, as shown in Figure 6-7.

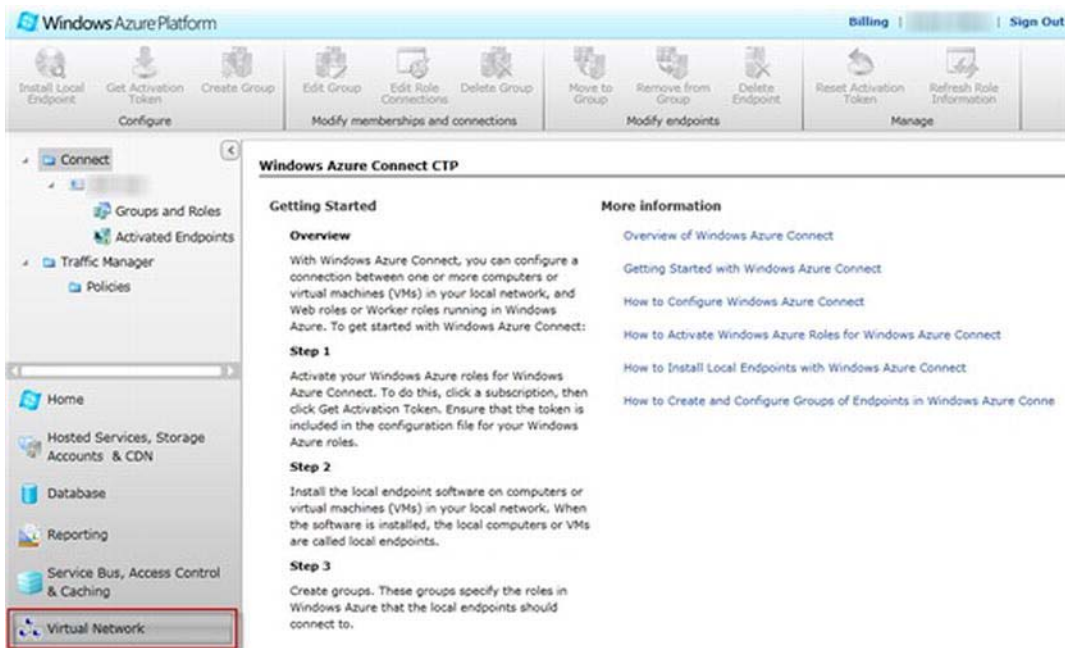


Figure 6-7. Virtual Network – Azure Management Portal

4. Next, you will need to enable Windows Azure Connect for your subscription. In the left-hand navigation pane, click on your subscription name.
5. A dialog box will pop up asking if you want to enable Windows Azure Connect for your subscription. Click Yes.

Once your subscription is enabled, you will need to create endpoints, and create a group containing your roles and on-premises machines.

Activated Endpoints, Groups, and Roles

Endpoints you wish to have available to Connect must be registered with Windows Azure. Once registered, you will have the ability to add these endpoints into groups that will be able to interact with each other as if they were in the same datacenter.

The Management Portal shows two folders in the navigation: Activated Endpoints and Groups and Roles. Activated Endpoints lists the endpoints that you have activated for this subscription. Groups and Roles will provide a tree view showing the groups you have created, and what endpoints exist in each group.

The end result we want is to have a group of endpoints connected, so first we need to activate those endpoints for our subscription. The process is different depending on whether you are activating an endpoint for a local machine, or an Azure role.

Installing and Activating an Azure Endpoint on a Local Machine

In order for local machines to be integrated with Windows Azure Connect, you will need to install the endpoint software on the machine itself.

The following steps are the easiest way to do this:

1. Go to the Virtual network tab on the Management Portal and click on your subscription in the left-hand navigation.
2. Then click Install Local Endpoint in the upper left-hand corner. You'll be presented with a dialog that contains a link (see Figure 6-8).

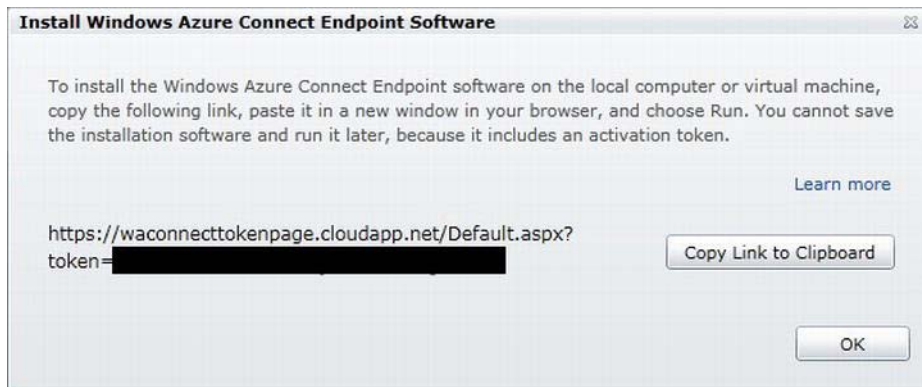


Figure 6-8. Local endpoint installation link

3. Click the button to copy the link to your clipboard.
4. If you are on the machine for which you want to install the endpoint, then open your browser and paste this link into your browser. If you plan to install to another machine, then you will need to paste this link to a text file.
5. Copy the text file to the other machine, and then copy/paste the link into the browser on that machine. This will download and install the endpoint software on your machine.

■ **Note** The local endpoint software is currently only compatible with Windows operating systems.

6. Once installed, note that there really isn't much for UI elements to determine that the endpoint is running. The only UI element is in your system tray (see Figure 6-9).

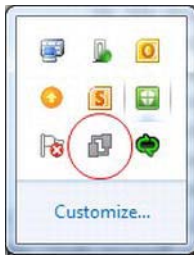


Figure 6-9. Azure Connect UI in system tray

7. If you click the tray icon, you will be presented with options to open Windows Azure Connect, Refresh Policy, or Diagnostics. Opening Windows Azure Connect brings up the dialog shown in Figure 6-10.

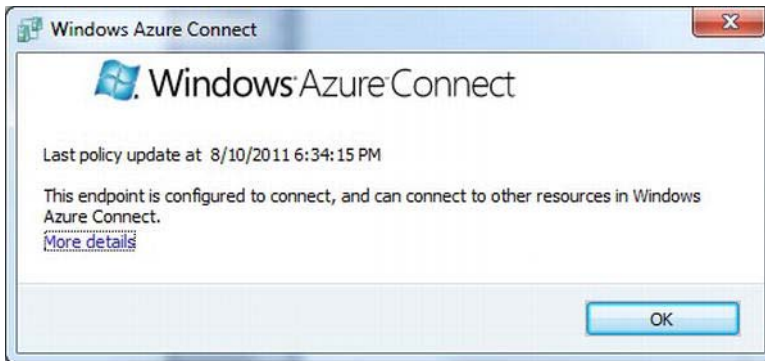


Figure 6-10. Windows Azure Connect endpoint software UI

If you're having connectivity issues, the diagnostics dialog might be useful in detecting issues (see Figure 6-11).

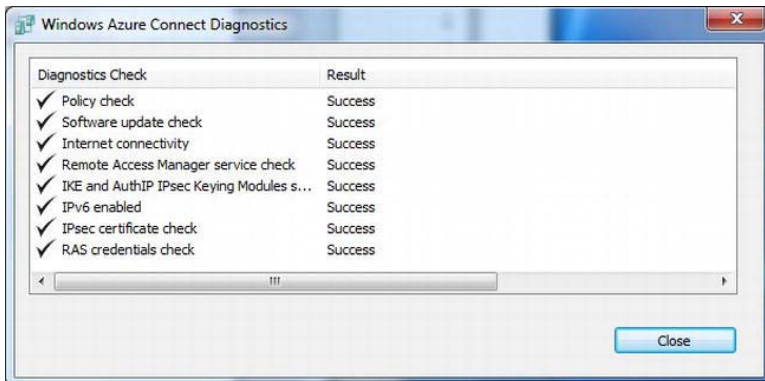


Figure 6-11. Windows Azure Connect Diagnostics dialog

Protocols and Ports

Note that Azure connectivity is based on IPv6 and HTTPS. This means that on the machine hosting the local endpoint software, TCP port 443 outbound must be opened, and firewall exceptions must be created for Internet Control Message Protocol version 6 (ICMPv6) communication. This is critical to establishing an IPv6 link. The endpoint software configures these for you, but you should be aware of these protocol/port/firewall requirements in case you run into issues. Additionally, you will need to configure other firewall exceptions as required by your applications.

Concerning your Windows Azure role instances, the endpoints/firewall rules are configured for you by Windows Azure. If you need a specific port opened for an application running on your instance, then you will need to configure that in the service definition, but otherwise, you won't need to make any specific changes for Windows Azure Connect.

Enabling Windows Azure Connect for a Role

In order to activate Windows Azure Connect for a role, you must get an activation token from the Management Portal, and copy that token into the service configuration for your role. This token tells Windows Azure to add this role to the collection of activated endpoints that can be added to an Azure group (see Figure 6-12).



Figure 6-12. Getting an activation token from the Management Portal

Connect must be enabled in the ServiceDefinition for your deployment. In order to enable Connect, open the role properties by double-clicking, go to the Virtual Network tab, select the Activate Windows Azure Connect checkbox, and paste the token you received from the portal (see Figure 6-13).

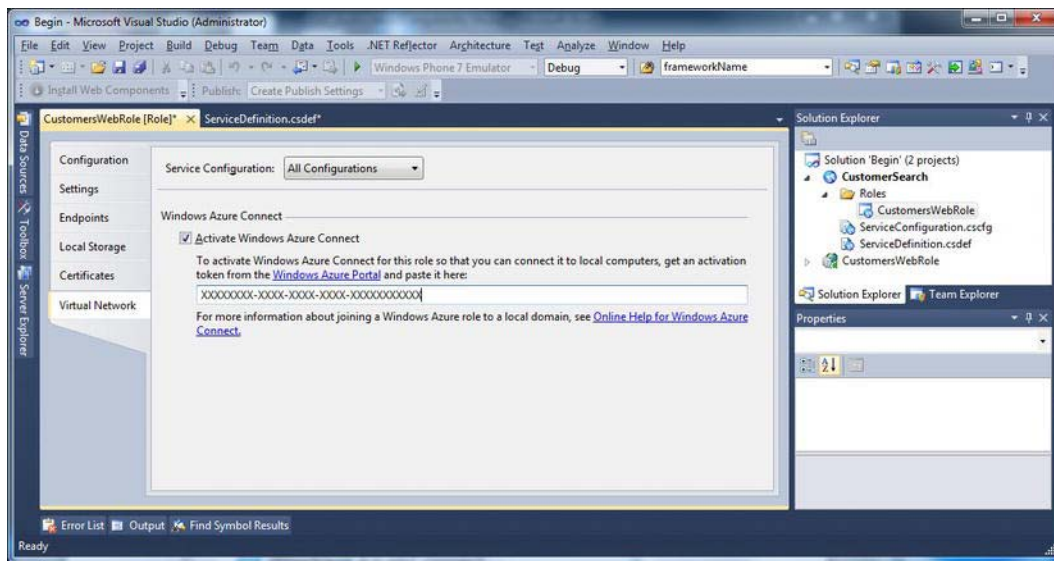


Figure 6-13. Activating Windows Azure Connect for a role

It's really that simple. Behind the scenes, your `ServiceDefinion.csdef` and `ServiceConfiguration.cscfg` files are modified.

```
<Import moduleName="Connect" />
```

The code in Listing 6-1 is added to the service configuration, and define the settings for the Connect environment:

Listing 6-1. *ServiceConfiguration.cscfg* modifications

```
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.ActivationToken" value="<TOKEN>" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.Refresh" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.Diagnostics" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.WaitForConnectivity" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.EnableDomainJoin" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainFQDN" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainControllerFQDN" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainAccountName" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainPassword" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainOU" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DNSServers" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.Administrators" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.DomainSiteName" value="" />
<Setting name="Microsoft.WindowsAzure.Plugins.Connect.Upgrade" value="" />
```

Once you deploy your solution, your endpoints will be activated. You can view these endpoints by clicking the Activated Endpoints folder in the Management Portal.

Creating Connect Groups

Creating groups is very simple once you have activated all the required endpoints. Go to the Management Portal, click the Virtual network tab, select your subscription, and click the Groups and Roles folder. Click Create Group. Provide a name for your group, and add your local endpoints. Note the checkbox for “Allow connections between endpoints in group.” This allows you to control whether the local endpoints can communicate with each other. Keep in mind that the local endpoints may not be in the same datacenter, so you need to decide whether you want to enable them to communicate directly with each other.

Next you add your Azure roles for which you have activated endpoints, or other endpoint groups you have already created. Click Create, and your group will be enabled and capable of communicating directly (see Figure 6-14)!

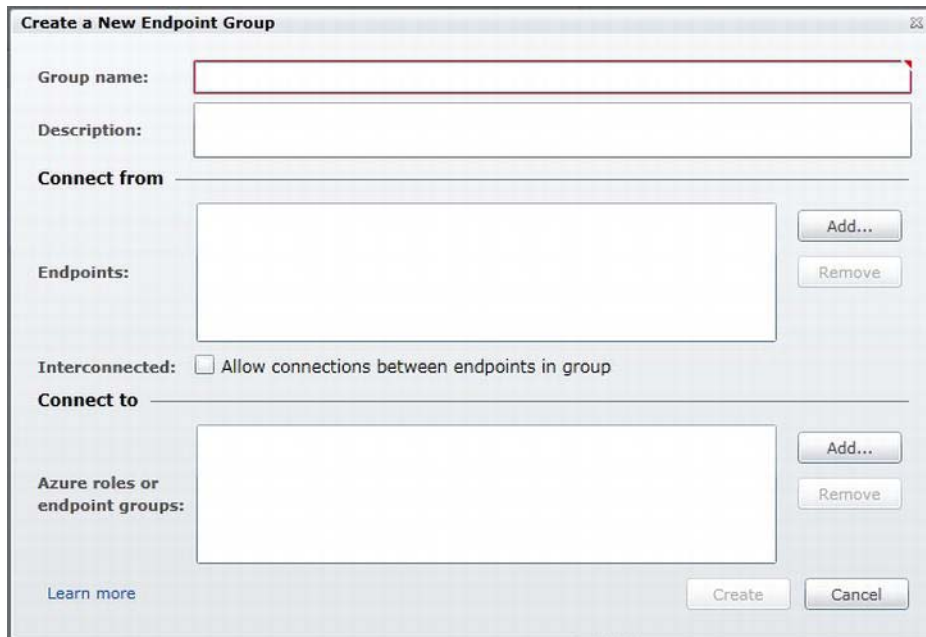


Figure 6-14. Creating a Windows Azure Connect endpoint group

■ **Note** If you un-deploy a role and re-deploy (which may happen often during development), the role will be dropped from the group and you will need to go back to the Management Portal and add it to the group again. Roles can be easily added or removed using the Edit Group button (see Figure 6-15).

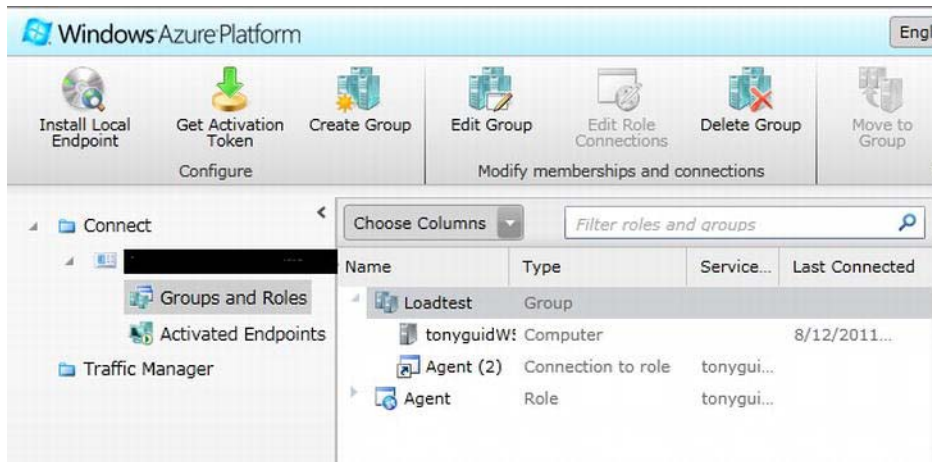


Figure 6-15. Group named Loadtest, with a computer and a role (two instances)

Summary

In this chapter you learned how to create a VM role, as well as the scenarios in which VM role can be useful, and other scenarios where it may not be an appropriate solution. We covered Windows Azure Connect and creating Azure Connect groups to enable machine-level connectivity between environments.

Bibliography

- MSDN: How to Install the Windows Azure Integration Components. <http://msdn.microsoft.com/en-us/library/gg465409.aspx>
- MSDN: Troubleshooting Windows Azure Connect. <http://msdn.microsoft.com/en-us/library/gg433016.aspx>
- MSDN: Overview of Firewall Settings Related to Windows Azure Connect. <http://msdn.microsoft.com/en-us/library/gg433061.aspx>

AppFabric: Access Control Service

Access Control Service (ACS) provides a facility for abstracting your authentication code, as well as mapping disparate claims from multiple identity providers into a single token and claim structure. In addition to saving you the trouble of writing code to authenticate to many identity providers, this makes it easier to write your authorization code, as you can expect to receive a consistent set of claims from ACS.

In this chapter, we will cover ACS in depth, starting with a quick discussion of digital identities. Then we will look at ACS usage scenarios, and move into the functionality provided by ACS. Finally, we will cover some of the programming aspects.

What Is Your Digital Identity?

I personally have at least 15 different identities, and it's tedious as well as insecure to maintain usernames and passwords for every application. You can categorize such identities as critical, important, and less important based on the impact they may have, not only on your digital life but also on your real life if you lose them. The critical ones are enterprise identities you may have with your company or partner companies (such as an Active Directory account) and financial identities with financial service providers like 401K fund managers, online banks, and so on. The important ones are personal e-mail identities like Hotmail, Yahoo Mail, and Gmail. The less-important identities belong to social-networking and other web portal sites and can be reestablished without any effect if necessary.

Where do these identities come from, and how are they maintained? You create an identity when you register with an identity provider. For example, when you join a company, the IT department creates an identity for you in their Active Directory. The identity is maintained in the Active Directory until you leave the company, and sometimes even after you leave. Similarly, when you register for a 401K plan, the plan provider creates a new identity for you that is maintained in the plan provider application. When you create a Hotmail or a Windows Live account, you get a LiveID to share across multiple Microsoft portals and applications online. Even in the same enterprise, most applications maintain their own identity providers like the database. This results in identity silos across organizations that are difficult to maintain. When a person leaves a company, the IT department has to not only delete their identity across all the applications, but also maintain these deletion records for compliance reasons. Partner companies also have to delete the user's identity from their extranet identity system.

As an application developer, often you have to design identity providers within applications; and if these applications are extranet or Internet facing, then the complexity of authentication and authorization increases significantly. You end up spending more effort on authentication and authorization design instead of the application's business logic. Ideally, you're given a standard interface for identity management that is consistent across all applications. The identity-management

architecture needs to be abstracted from the application architecture so you can focus your time on business logic and reduce the costs associated in maintaining the identity-management infrastructure for every application. Some large organizations deploy an enterprise-wide single sign-on solution that all applications can leverage to solve the multiple identity providers problem. But, when you are designing applications for the cloud, there is no enterprise-grade identity provider, like Active Directory, available in the cloud (including Windows Azure platform) that can store your identities. The whole software industry is moving towards a claims-based model in which the authentication responsibility stays with the identity provider whereas the authorization responsibility stays with the application.

Windows Azure AppFabric Access Control Service (ACS) is a cloud service that abstracts the orchestration of authentication for your application. ACS follows a claims-based architecture where users acquire their claims from ACS based on their identity and present the claims to the application. The application is configured to trust ACS and uses the information presented in the claims to determine appropriate entry to users. In simple terms, ACS is a claims-transformation service in the cloud that relieves you of managing identities within your application.

■ **Note** Windows Identity Foundation (WIF) is the underlying framework for building claims-aware applications. WIF is a developer tool and an extension of the .NET Framework that makes it easier for you to develop claims-aware applications using familiar tools like Visual Studio. It is a standalone framework and thus detailed information is outside the scope of this book. Some general concepts and terminology are provided at the end of this chapter. However, you can learn details about WIF from the Identity Developer Training Kit available from Microsoft here (www.microsoft.com/download/en/details.aspx?id=14347).

What Are Claims?

Claims are a set of attributes that an application expects. As a user, you present a set of claims to an application and then let the application decide the privileges it can offer you based on those claims. I consider claims as natural advancements to the role-based authorization model. For example, you may have an application that expects e-mail address, phone number, password, and employee ID from an end user as the set of claims for determining the appropriate access control. You can configure your ACS to provide these user claims to your application independent of the user's identity provider. Figure 7-1 illustrates a simple view of ACS.

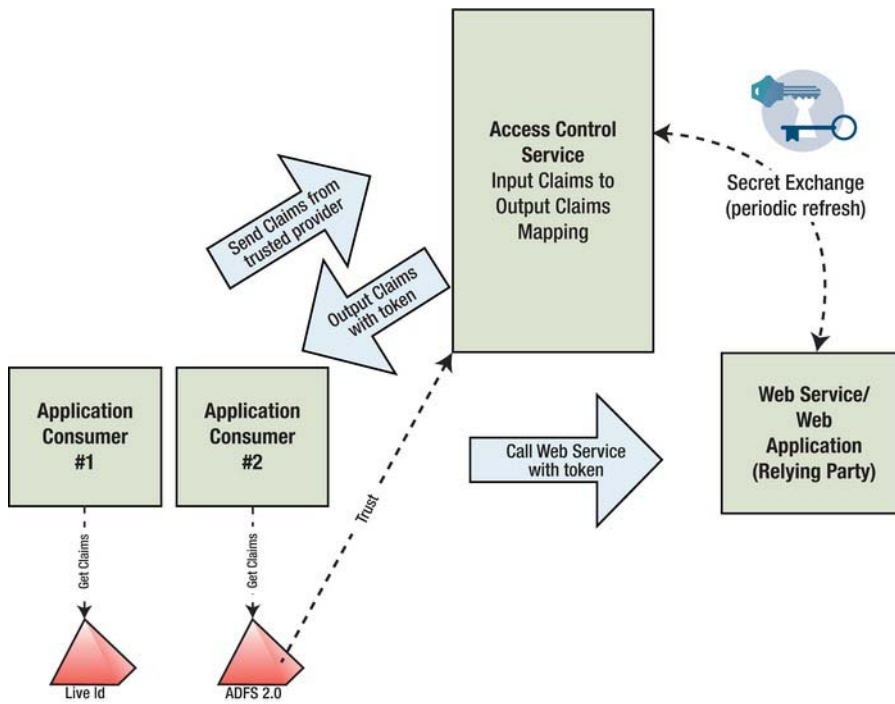


Figure 7-1. A simple view of the Access Control Service

The pattern illustrated in Figure 7-1 is called a *claims-based identity model* because the claims tie together the relying party, ACS, the identity provider, and the consumer. The primary function of ACS is to transform input claims into output claims as follows:

1. Configure ACS and the identity provider to trust each other.
2. Configure ACS and your service (a.k.a., relying party) to trust each other with a signing key.
3. Configure ACS with rules for mapping input claims to output claims that your application expects. In the real world, these tasks are performed by system and/or security administrators.
4. When an application wants to consume the web service, it sends the required claims to ACS in a request for a token.
5. ACS transforms input claims into output claims based on the mapping rules you created while configuring ACS.
6. Next, ACS issues a token with output claims to the consumer application. The consumer application sends the token in the request header to the web service.

7. The web service validates the claims in the token and provides appropriate access to the end user.

In this model, if you are building your application using .NET Framework, the consumer and the relying party can both benefit from using the WIF APIs.

■ **Note** Active Directory itself doesn't support a claims-based identity model. You will need Active Directory Federation Services 2.0 (ADFS 2.0) to provide claims-based identity support to Active Directory. ADFS 2.0 is built using WIF. ACS is used for abstracting multiple identity providers from the relying party. If you are sure that you will be using one and only identity provider, then you can use WIF to directly federate with the relying party. The Identity Developer Training Kit available from Microsoft here (www.microsoft.com/download/en/details.aspx?id=14347) has several examples for using WIF to federate directly with the relying party, without using ACS.

The important information to take from this example is the fact that ACS abstracts multiple token providers from the relying party by always issuing the same type of token. The relying party has to only consider the output claims in its authorization logic to provide appropriate access to the end user. As a result, you as a developer only have to program your application or service against a set of output claims independent of input claims from multiple identity providers. You can reuse these output claims across multiple applications within the enterprise, cross enterprise, and even over the Internet. The relying party can be a web application or a web service. I cover each step discussed in Figure 7-1 in more detail later in the chapter.

■ **Note** Before diving deep into a technical discussion, you might want to review some key concepts and terminology that are extremely important in the design and architecture of ACS. The "Concepts and Terminology" section at the end of the chapter introduces some new terms and redefines some existing terms in the ACS context.

Claims-Based Identity Model

This section goes over the details of the claims-based identity model in ACS. Specifically, I expand on the discussion from Figure 7-1. Figure 7-2 illustrates the interaction between different components in a claims-based identity model. With the terminology defined, it will be much easier for you to understand the flow of information between different parties in this model.

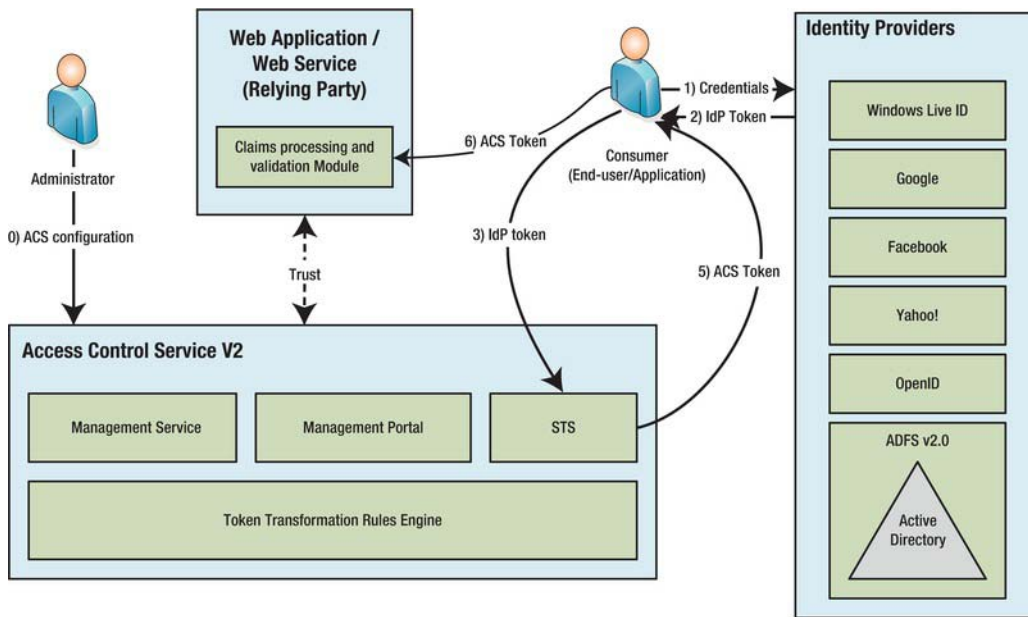


Figure 7-2. ACS claims-based identity message flow

As illustrated in Figure 7-2, several messages and tokens are passed back and forth between the key parties in a claims-based identity model. Before the interaction starts, prerequisites (listed as step 0) are required in order to make the end-to-end scenario work. The following steps describe the flow of information from the requesting user or application to the relying party:

Step 0: Two important prerequisites for claims-based identity to work are completed in this step. First, trust is established between the relying party (web service), ACS, and identity providers. The trust relationships are refreshed on a periodic basis. The trust between ACS and the relying party is established using a signing key. Second, an administrator creates an issuer to identify service consumers and defines the mapping rules between input claims and output claims in the form of rules in ACS. The issuer key material is distributed to the service consumer.

Step 1: When ACS, the relying party, and identity providers are configured for the claims-based identity model to work seamlessly, the service consumer must use the issuer key material to acquire a token from ACS in order to call the web service. In the current version (Version 1.0 production), the ACS supports the following three types of token requests:

- *Plain text:* The service consumer sends an issuer key directly to ACS to authenticate the request.

- *Signed*: The service consumer creates an SWT, signs the token, and sends the token to ACS for authentication. In this method, unlike with the plain text request, the service consumer doesn't need to send an issuer key directly to ACS. Typically, the signed token from the consumer includes input claims that are then mapped to output claims by ACS and included in the output token emitted by ACS.
- *SAML*: The service consumer acquires a signed SAML token from ADFS 2.0 or a similar identity provider that emits SAML tokens, and sends it to ACS for authentication. Intended primarily for ADFS 2.0 integration, this approach requires that a signed SAML bearer token be acquired and sent to ACS for authentication.

■ **Note** ACS Version 2.0 was released in April 2011. The following features were added:

Web-based administrative access to ACS configuration

An OData-based Management Service that provides programmatic access to ACS configuration

Support for the SAML 1.1, SAML 2.0, and Simple Web Token (SWT) token formats

Support for some web identity providers: Windows Live ID, Google, Yahoo, and Facebook

Support for Active Directory Federation Server v2.0

Support for OAuth 2.0 (draft 10), WS-Trust, and WS-Federation protocols

Step 2: Based on the claims-mapping rules configured in ACS, ACS maps the input claims received in the service consumer token to output claims specific to the web service. The ACS then issues an SWT¹ or a SAML token consisting of output claims to the service consumer. ACS signs the token using the key registered in Step 0. The mapping of input claims to output claims makes ACS an R-STs. ACS abstracts the token-issuing party from the token-consuming party by always emitting an SWT containing output claims the web service expects.

¹ Simple Web Token specification: <http://groups.google.com/group/oauth-wrap-wg>

Step 3: Regardless of the method used to acquire the input token, ACS creates an SWT or a SAML token and sends it to the service consumer. This token contains output claims that the web service expects.

Step 4: The consumer packages the token into an HTTP header and sends it to the web service along with the message payload.

Step 5: The web service validates the token based on the secret key exchange established in Step 0. The web service also validates the required claims and grants or denies access to the resource based on the validation outcome. There is no direct communication between the web service and ACS during the method invocation. The only communication happens during the periodic refresh of the secret key exchange. The token consists of all the information needed by the web service, therefore it is important to send the token the web service using a secure channel like HTTPS.

Figure 7-2 may look complex initially, but when you go through the steps, the claims-based identity is easy to understand. The next section puts the claims-based identity model into an enterprise scenario perspective.

■ **Note** In the future, ACS will support multiple tokens as input as well as output. Therefore, in the interest of keeping the text simple, when I mention the word “token” in the text, it can either be SWT, SAML, Facebook, Google Id, LiveID, or any token that is supported by ACS. If there is a need to be explicit, I will explicitly mention the type of token supported.

Access Control Service Usage Scenarios

Now that you understand the claims-based identity and ACS concepts, some real-world scenarios will provide more clarity about these concepts. This section presents three real-world scenarios. Scenario 1 shows how an enterprise cloud application can benefit from ACS. Scenario 2 illustrates the use of ACS in a cross-enterprise scenario, and finally scenario 3 shows an ISV cloud service using ACS across multiple customers.

■ **Note** A great resource for additional information is Alik Levin's blog, which is located at <http://blogs.msdn.com/b/alikl/>. One post provides a whitepaper that describes various cloud identity scenarios, with code samples. You can find that post at <http://blogs.msdn.com/b/alikl/archive/2011/09/30/cloud-identity-stories-for-developers-application-architecture-scenarios.aspx>.

Scenario 1: Enterprise Cloud Application

For this scenario, consider a news organization called T-Press Inc., similar to the Associated Press or Reuters. T-Press has a large workforce of journalists and technicians based around the world, who are busy investigating, planning, and creating news events. Usually, journalists and technicians can be either employees or contractors, but for the purpose of this scenario, assume that the journalists are employees and the technicians are contractors. Currently, T-Press has a newsroom-management web application called T-Room. T-Room is a globally deployed application and can be used by all journalists and technicians in the field. The T-Room web application is deployed in the cloud and federated with T-Press's Active Directory using ADFS 2.0. The deployment of T-Room in the cloud makes it accessible from anywhere in the world with Internet access. Journalists can view all the current and historical T-Press news items, but technicians can view only the news items to which they're assigned. The current pain point from an identity-management perspective is as follows.

Technicians are typically hired for a short period of time covering the lifetime of a single news event. After the contract expires, the technician rolls off and may or may not join the workforce for another news event. Currently, whenever a technician is hired, an account is created in T-Press's Active Directory. Due to the short contract periods, identity management has become expensive, and T-Press would really like to move away from creating short-term Active Directory accounts for technicians. Instead, T-Press wants to support any technician's existing digital ID (such as a Windows Live ID, Facebook ID, and the like) to access T-Room. T-Press needs help designing an access control system that can not only support public digital IDs but also give immediate access to the T-Room application from anywhere in the world. I recommend that T-Press design a claims-based identity model for T-Room and use ACS to abstract a technician's identity provider from T-Room. The design is illustrated in Figure 7-3.

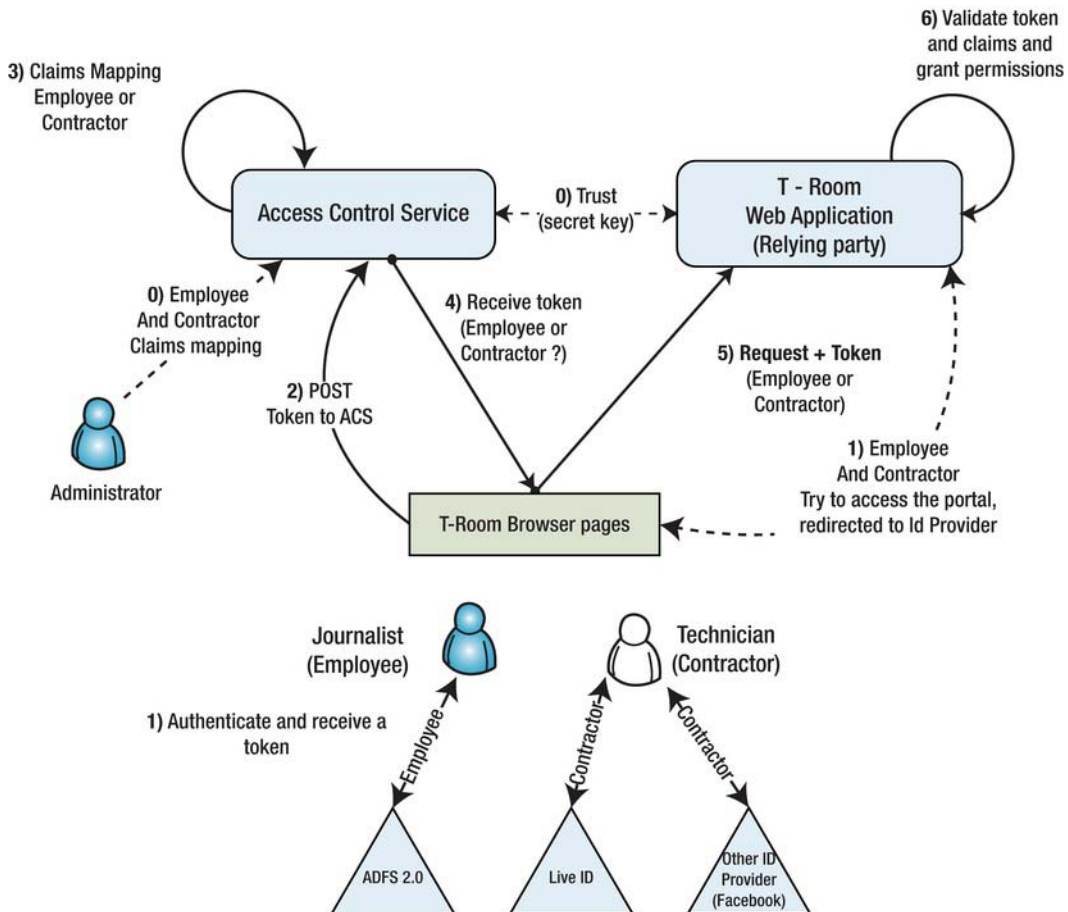


Figure 7-3. Enterprise cloud application scenario

The following steps describe the flow of information from the browser to the T-Room web application:

Step 0: The T-Room system administrator completes all the prerequisites to make ACS work for the T-Room application. In Figure 7-3, the important steps are establishing trust between the T-Room web application and ACS using a shared key, which is refreshed on a periodic basis; configuring ACS with the supported identity providers (such as ADFS 2.0, Windows LiveID, Facebook, and so on), and defining the mapping between input claims and output claims in the form of rules for employees and contractors. This is where the administrator can define different claims for employees and contractors. The ACS can be configured to process employee authentication with ADFS 2.0, whereas contractors can be authenticated using external identity providers. I have also seen real-world applications where we built custom STS for other identity providers like a simple SQL Server database,

Novell eDirectory, Computer Associates SiteMinder, and the like. WIF gives you the flexibility to build custom STS on top of any backend identity provider.

Step 1: First, the requestor goes to access the web application. The web application identifies there is no token in the request. Therefore, requestor is redirected to ACS and then to the login page of the appropriate identity provider. The requestor authenticates with the identity provider and acquires a token.

Step 2: The requestor posts the acquired token to ACS for claims mapping.

Step 3: ACS is an important piece of the identity federation orchestration because the token is sent to ACS to transform input claims to output claims the T-room application understands. T-Room application is not designed with any specific identity provider in mind, but only the claims. This is a very important concept to understand.

Step 4: ACS returns a token to the requestor. This token consists of the output claims that only the T-Room application understands.

Step 5: The requestor packages the token along with the payload and sends it to the relying party (the T-Room web application).

Step 6: The T-Room application processes these claims in a claims-processing module and determines the level of access the requestor is entitled to. The claims-processing module doesn't depend on any identity provider but only validates the claims from the requestor's token.

■ **Tip** The key concept to understand here is that if there was only one identity provider, you could validate the claims based on that identity provider, but every identity provider generates different claims and thus a different token structure. This forces the developer to update the relying party code for supporting specific identity providers. By using ACS, the developer can expect only one type of token emitted from ACS that is independent of the identity provider. This makes the relying party extensible. Therefore, when you design your relying party, make sure you consider the current and future requirements from the user identity perspective.

In Figure 7-3, the introduction of ADFS 2.0 and ACS into T-Press's existing infrastructure simplifies the identity management of a cloud application like T-Room that supports users from outside of the organization. The T-Press system administrators don't have to manage the identities of technicians in T-Press systems anymore. Technicians use their existing identities to access the T-Press application. Administrators can configure input to output claims mappings in ACS as per the business requirements. The T-Press developers only have to focus on building a claims-processing module for ACS-forwarded claims; they don't have to write separate modules for each identity provider as before. Thus, for T-Press, ACS successfully abstracts claims from multiple identity providers into a single coherent view of output claims for the T-Room web application.

Scenario 2: Cross-Enterprise Application

In this scenario, two partner enterprises would like to collaborate on several projects using each other's collaboration platforms. Enterprise A is a software company that manufactures operating system software. Enterprise A has partner companies (OEMs) that customize these operating systems, install those systems on their hardware, brand the integrated platform, and sell the product to consumers through their sales channels. The end product may be a personal computer, a laptop, or even a cell phone. For this example, Enterprise B is an OEM of Enterprise A. To launch a particular product in time, Enterprise B needs early access to some of the software releases and documentation associated with those releases. Enterprise A, on the other hand, needs information about sales of the end product to use for sales and revenue tracking of its own product.

Enterprise A has a dedicated web application named PartnerAccess in its extranet for OEM partners. The PartnerAccess web application supports role-based authorization for different roles in multiple partner enterprises. Some example partner roles are Partner_Manager, Partner_Employee, and Partner_Contractor.

Enterprise B has a list of users configured in Enterprise A's Active Directory who have access to the early release operating system software through the web application in Enterprise A's extranet. Enterprise B users log in to this application using their Enterprise A credentials. Enterprise A's administrators find it difficult to track and maintain users of Enterprise B and other partners, because the administrators have to delete or modify partner users when they quit their company or are promoted to another position. Remember that the information shared between these companies is top-secret and cannot be leaked under any circumstances. The total number of partner users' numbers totals hundreds of thousands across multiple OEM partners. Maintaining these partner user identities has become expensive and risky for Enterprise A, and the company is looking forward to mitigating its identity-management risks for the PartnerAccess web application.

On the other side of the equation, Enterprise B has a similar problem maintaining Enterprise A identities for thousands of Enterprise A employees and contractors in its SalesAccess software. The SalesAccess software provides sales information to Enterprise A. Enterprise A users log in to the SalesAccess software using their Enterprise B credentials, to acquire real-time sales information from Enterprise B. Other partner companies also have similar applications providing sales data access capabilities.

As an architect, I recommend Enterprise A and B to design a claims-based identity model for its web applications and use ACS to abstract the partner's identity providers. Each enterprise owns the responsibility of maintaining and authenticating its own employees. The recommended design is illustrated in Figure 7-4.

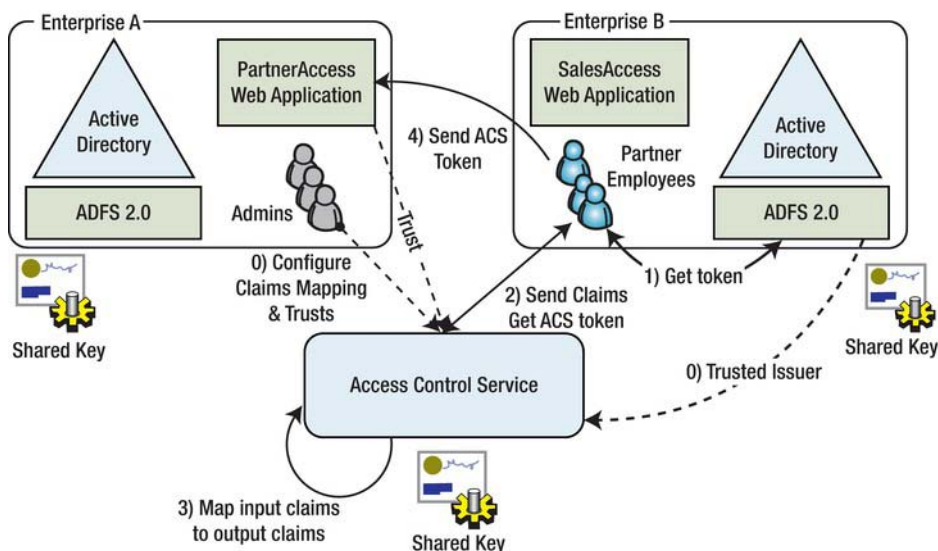


Figure 7-4. Cross-enterprise scenario

The following steps describe the claims model for the PartnerAccess web application:

Step 0: The PartnerAccess web application completes all the prerequisites required to make ACS work for the application. In Figure 7-4, the important steps are establishing trust between PartnerAccess and ACS using a shared key, which is refreshed on a periodic basis; having the PartnerAccess administrator and Enterprise B administrator configure ACS to trust Enterprise B's ADFS 2.0 identity provider to generate STS for Enterprise B users; and having PartnerAccess define the mapping between input claims from Enterprise B's ADFS 2.0-generated SAML tokens and output claims in the form of rules specific to the PartnerAccess application. This is where the administrator can define different claims for different roles for Enterprise B employees.

Step 1: When an Enterprise B employee wants to sign in to the PartnerAccess web application, the employee is authenticated with Enterprise B's Active Directory, and the ADFS 2.0 generates a SAML token for ACS. Because Enterprise B is in control of its employee identities, and Enterprise A trusts Enterprise B's authentication process, it makes sense to delegate the authentication of Enterprise B's employees to Enterprise B.

Step 2: The SAML token generated by Enterprise B's ADFS 2.0 is sent to ACS. The SAML token consists of input claims.

Step 3: ACS maps the input claims from the SAML token to output claims specific to the PartnerAccess web application and packages them into a token (SAML or SWT).

Step 4: The ACS token with output claims is sent to the PartnerAccess web application for processing. The PartnerAccess application validates the token, processes these claims in a claims-processing module and determines the level

of access the Enterprise B employee is entitled to. The PartnerAccess web application doesn't need to authenticate Enterprise B users in the Enterprise A environment because it trusts tokens generated by ACS.

The introduction of ACS into Enterprise A's environment and federating Enterprise B identities using ADFS 2.0 simplifies the management of the partner accounts. Enterprise A can reuse this configuration for all the partners accessing the PartnerAccess web application, whereas Enterprise B can reuse ADFS 2.0 to federate identities across multiple partner companies. Note that the PartnerAccess web application isn't dependent on the identity providers of partner companies. As long as a trust is established between ACS and the partner identity provider, the PartnerAccess application does the necessary claims processing for any partner.

For the SalesAccess web application, Enterprise B can implement the same pattern by introducing ACS in the architecture and letting Enterprise A employees authenticate and generate tokens using Enterprise A's own ADFS 2.0. With the same pattern implemented in Enterprise B's architecture, Enterprise A and Enterprise B can access each other's applications seamlessly by removing the identity-management burden from the partner company. Identity management remains with the company that owns the identities.

Scenario 3: ISV Cloud Service

In this scenario, an independent software vendor (ISV) named My Energy offers an energy-management cloud service to multiple utility companies. The service performs data collection from power meters on houses and commercial buildings and offers this data to utility companies for reporting and processing. Currently, the ISV service has its own identity-management database and for every utility company. Due to resource constraints, maintaining identities of all the utility partner companies has turned into an expensive process. Every time an employee of a utility company quits, My Energy has to remove the employee from the database. My Energy wants to reduce its identity-management costs because it's turning out to be a significant portion of the company's support operating expenses. Assuming that utility companies have an identity federation infrastructure, I recommend that My Energy implement a claims-based identity model using ACS as the claims-transformation engine. My Energy can use ACS to map claims issued by a utility company's identity federation server (such as ADFS 2.0) to claims required by the My Energy service.

The recommended design is illustrated in Figure 7-5.

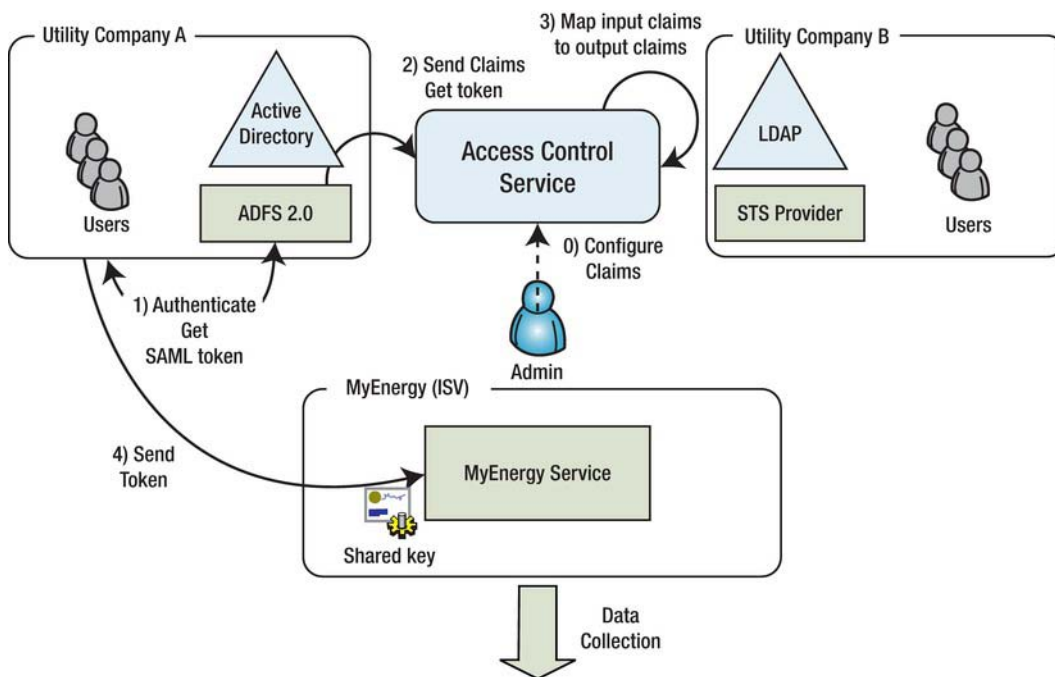


Figure 7-5. ISV cloud service scenario

The following steps describe the claims model for the My Energy web application:

Step 0: In this step, similar to previous scenarios, the My Energy administrator establishes trust relationships between My Energy, ACS, and the identity providers of utility companies. Then, the My Energy administrator configures ACS by mapping input claims from the identity providers to output claims specific to the My Energy application.

Step 1: When a utility company employee wants to sign in to the My Energy service, the employee authenticates with the utility company's identity federation server and receives a token.

Step 2: The token is sent to ACS. Because ACS is configured to trust the company's identity federation server, ACS can accept input claims from the issuer. The token consists of input claims to ACS.

Step 3: ACS maps the input claims from the token to output claims specific to the My Energy service and packages them into a secondary token.

Step 4: The ACS token with output claims is sent to the My Energy service for processing. The My Energy service processes these claims and determines the level of access to which the utility company's employee is entitled.

Using ACS, the My Energy service can support multiple utility companies without managing their identities in its own identity store. The identity management costs mainly involve claims mapping and

establishing trust between identity providers and ACS; but these are one-time efforts per utility company. After trust is established and claims are configured, the claims-based identity process will work seamlessly for My Energy. The My Energy service no longer maintains a separate identity-management store, because users are authenticated against the utility company's identity store. My Energy is configured only to process output claims coming from ACS.

The three scenarios discussed demonstrate the following ACS advantages:

- ACS federates between wide varieties of identity providers because of its standards-based interface.
- ACS abstracts identity management from your application or service.
- ACS abstracts out claims management from your application or service.
- ACS can help achieve single sign-on across diverse systems because of its standards-based API.
- ACS works with web browsers and web applications (passive participants) as well as smart clients and web services (active participants).
- ACS provides an STS for issuing SWT and SAML tokens containing output claims. Every mapping scope can be considered to have its own virtual STS.

Retrieving Tokens from ACS

The ACS version 2.0 supports a variety of identity providers out-of-the-box. These are, ADFS 2.0, Windows Live Id, Facebook Id, Yahoo Id, Google Id, and any SWT or SAML tokens generated by custom STS. ACS supports only SSL transmission of the tokens over HTTP POST. ACS also supports OAuth 2.0 draft (draft 10), WS-Trust, and WS-Federation protocols. Irrespective of the input token, ACS always issues an SWT or SAML output token that consists of output claims the relying party expects. Figure 7-6 illustrates these token-retrieving methods.

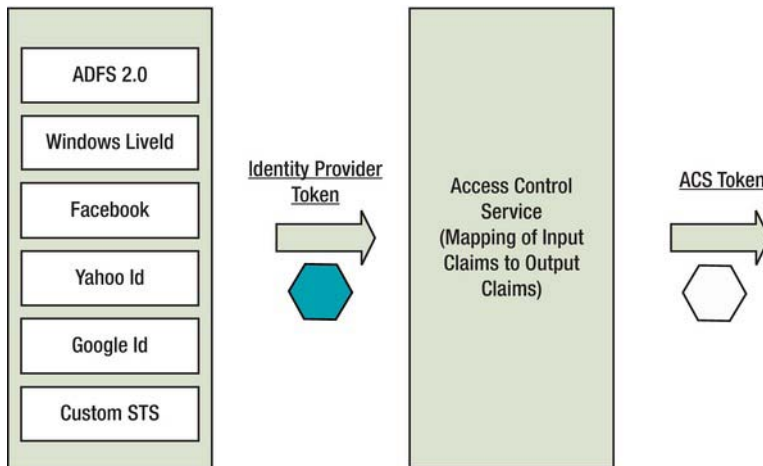


Figure 7-6. Retrieving tokens from ACS

ACS version 2.0 supports much more identity providers and protocols than version 1.0 which supported Plain Text, SWT, and SAML as input tokens and only SWT as the output ACS Token. In version 2, ACS supports multiple identity providers and token types.

Access Control Service Management Portal

The ACS version 2.0 Management Portal provides provisioning and configuration capabilities. As such it is your gateway to setting up ACS functionality. To set up an ACS-enabled application, you will perform the following steps:

1. Provision an ACS service namespace.
2. Configure the identity providers that will interact with this service namespace.
3. Configure the application(s) that will be authenticated by the service namespace.
4. Set up certificates, keys, or service identities
5. Modify the application to interact with ACS.

These steps are defined in detail in the following sections.

Provisioning Your ACS Service Namespace

The following are the steps you will typically take while provisioning your ACS service namespace.

1. Log in to the Windows Azure Management portal.
2. Select the ServiceBus, Access Control, and Caching option from the left pane.
3. On the Windows Azure AppFabric portal page, select Access Control and click the New Namespace button. Then choose a name for your service namespace that is unique, as shown in Figure 7-7.

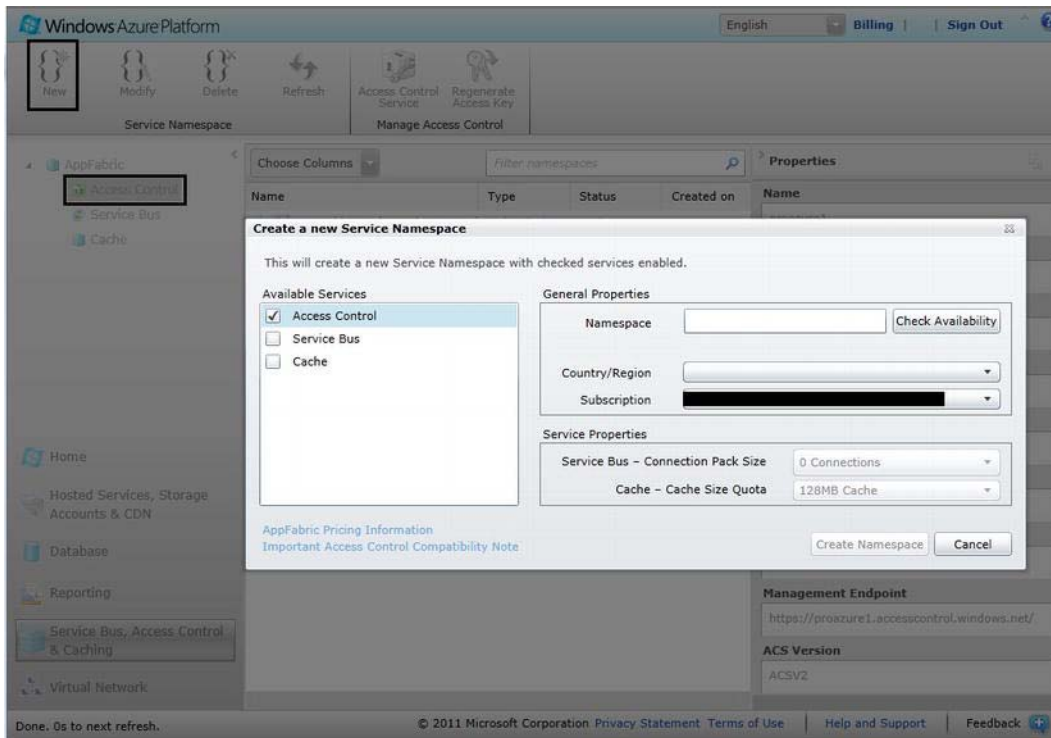


Figure 7-7. New namespace

4. Click OK to create the service namespace.
5. Select the newly created service namespace and click the Access Control Service button from the Manage Access Control group in the top menu to go to the service namespace management portal, shown in Figure 7-8

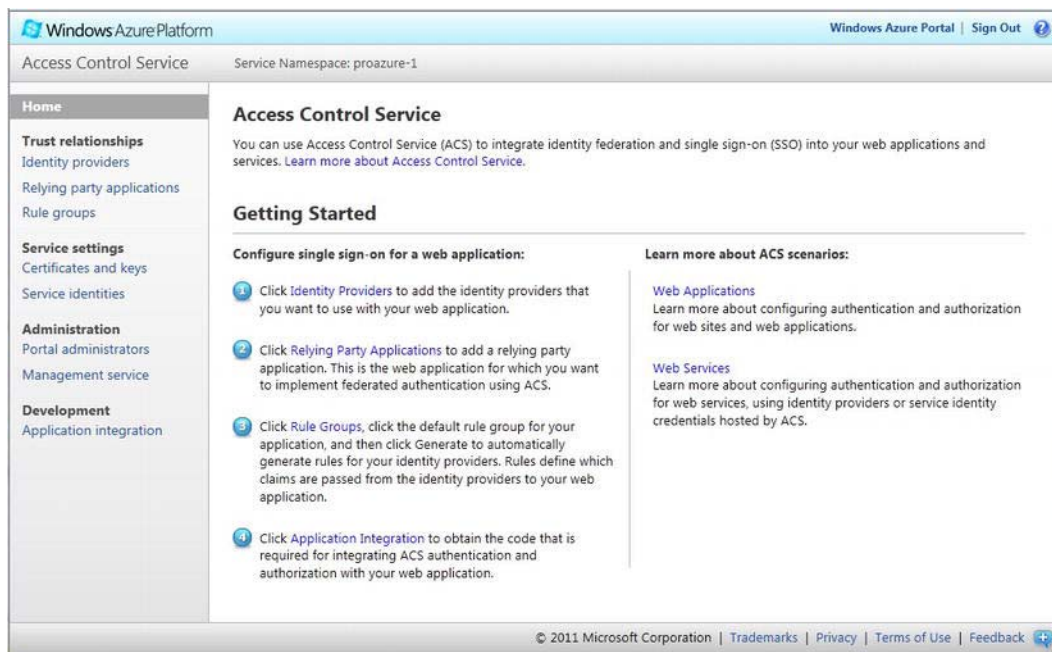


Figure 7-8. Manage Access Control Service

The service namespace management portal gives you the ability to manage the different entities that will participate in the claims-based identity orchestration involving ACS. A service namespace defines the namespace for your ACS resources. You must select a unique name for your service namespace and the region in which you want the service namespace to run. If you're building a Windows Azure distributed application, you can choose a common location for the Windows Azure services, SQL Azure database, and AppFabric service namespace so that all the Windows Azure components run in close proximity to each other and yield better performance. ACS also provides a management web service for provisioning and configuring ACS service namespaces.

■ **Tip** An ACS service namespace provides you with an out-of-the-box partitioning scheme for multi-tenant applications. You can create a separate service namespace for every customer and automatically partition the access per customer. This will help you isolate, debug, and scale the ACS access control to your application. By using the ACS management API, you can automate the provisioning of these namespaces whenever a new customer signs up for your service.

The service namespace page consists of four sections: Trust Relationships, Service Settings, Administration, and Development. The Trust Relationships section contains sub-services for managing the relying parties, identity providers, and the rules for configuring input claims (from the identity

provider's STS) to output claims (your relying party is expecting). The Service Settings section is a place for configuring certificates and keys for signing and encrypting tokens. This section also consists of section for configuring service identities. Even though ACS is not primarily designed for storing identities, it does provide a section for storing few identities you can use for testing purposes. The Administration section allows you to add ACS portal administrator accounts and management web service client accounts. The development section allows you to configure custom login pages and also lists the endpoints for the ACS service namespace.

Now that our service namespace is provisioned, we need to start setting up identity providers.

Identity Providers

Identity providers are repositories where user identities reside. For example, Active Directory, custom database with user name and passwords, and LDAP directories are all identity providers. In claims-based identity model, the identity providers are responsible for authenticating the user and then release a secure token (e.g., SAML and SWT) from its secure token service (STS). An STS that is tightly coupled with an identity provider is called an Identity Provider STS or IP-STS. ADFS 2.0 is IP-STS for Active Directory because it is tightly coupled with Active Directory and emits SAML tokens for Active Directory authentications. The Identity Providers section of the ACS service namespace management portal supports the following identity providers:

- Active Directory Federation Services 2.0
- Windows Live ID
- Facebook
- Google
- Yahoo!

Active Directory Federation Services 2.0 (ADFS 2.0)

ADFS 2.0 is a separate product from Microsoft that can be downloaded and installed independently of Active Directory. ADFS 2.0 is built using WIF and can be used as an STS for Active Directory and other identity providers. ADFS 2.0 can provide claims in the form of SAML tokens from Active Directory to ACS. ACS then maps these incoming claims to outgoing claims expected by the Relying Party application. Before configuring ADFS 2.0 as an Identity Provider, you need to install ADFS 2.0 and configure it for releasing claims in association with the backend identity provider.

■ **Note** For more information on installing, configuring, and deploying ADFS 2.0, please refer to the following TechNet documentation ([http://technet.microsoft.com/en-us/library/adfs2\(ws.10\).aspx](http://technet.microsoft.com/en-us/library/adfs2(ws.10).aspx)). The documentation includes step-by-step procedures for interoperating with various other federation and single sign-on providers like CA SiteMinder, Ping Identity Ping Federate, and Oracle Identity Federation.

The step-by-step procedure for configuring ADFS 2.0 as an identity provider in ACS is as follows:

1. Click the Add link on the Identity Providers as shown in Figure 7-9.

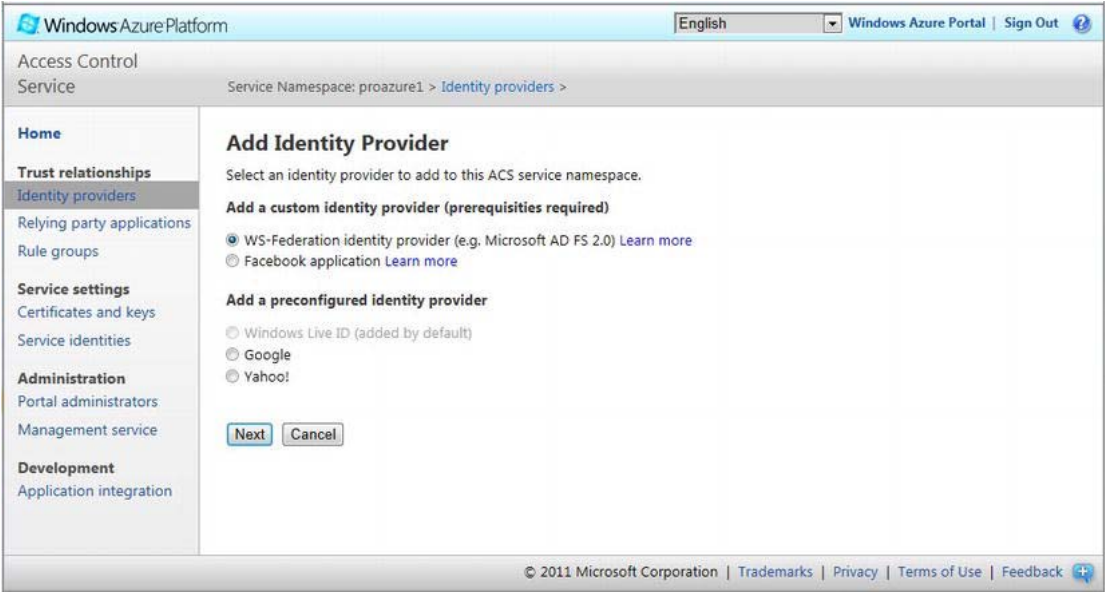


Figure 7-9. Add ADFS 2.0 identity provider

2. Click Next and fill up the form displayed on the Add ADFS 2.0 Identity Provider page as shown in Figure 7-10.

Windows Azure Platform | English | Windows Azure Portal | Sign Out

Access Control Service | Service Namespace: proazure1 > Identity providers >

Home

Trust relationships

Identity providers

Relying party applications

Rule groups

Service settings

Certificates and keys

Service identities

Administration

Portal administrators

Management service

Development

Application integration

Add WS-Federation Identity Provider

Use the following options to configure a WS-Federation identity provider in this ACS namespace, such as Microsoft Active Directory Federation Services (AD FS) 2.0. [Learn more about WS-Federation identity providers.](#)

Identity Provider Settings

Display name
Enter a display name for your identity provider. This name is used in the ACS Management Portal only.

WS-Federation metadata
Upload or enter the URL for the WS-Federation metadata document for your server. [Learn more](#)

☒ URL:

☐ File:

Example: <https://www.contoso.com/FederationMetadata/2007-06/FederationMetadata.xml>

Login Page Settings

Login link text
Enter the text to display for the login link for this identity provider. [Learn more](#)

Image URL (optional)
If you want to display an image as the login link for this identity provider, enter the URL of the image file that you want to use. [Learn more](#)

Email domain names (optional)
If you want to prompt users to log in using their email address, enter the email domain suffixes that this identity provider hosts. Otherwise, leave this box blank to display a direct login link. Use semicolons to separate the list of suffixes. [Learn more](#)

Example: [contoso.com](#); [fabrikam.com](#).

© 2011 Microsoft Corporation | [Trademarks](#) | [Privacy](#) | [Terms of Use](#) | [Feedback](#)

Figure 7-10. Add ADFS 2.0 details

■ **Note** The WS-Federation metadata represents the metadata definition of your ADFS 2.0 STS. When you upload the federation metadata, make sure you use HTTPS endpoint and upload metadata from the ADFS 2.0 service that you already know. In this configuration process, you are essentially establishing trust between ACS and ADFS 2.0; therefore, make sure you follow all the security best practices for accessing any remote service information.

3. If your ADFS 2.0 is encrypting tokens using a certificate, then you can also choose to add a token decryption certificate in the Certificates and Keys section as shown in Figure 7-11.

Certificates and Keys

Add or manage the certificates and keys used for token signing, encryption, and decryption in this Access Control Service namespace. For more information, see [help on certificates and keys](#).

[Add](#) | [Delete](#)

Token Signing			
Used For	Type	Effective Dates	Status
<input type="checkbox"/> Service Namespace	Symmetric Key	2/24/2011-12/31/9999	Primary
<input type="checkbox"/> Service Namespace	X.509 Certificate	2/24/2011-2/24/2012	Primary
<input type="checkbox"/> Web Service Certificate Auth	X.509 Certificate	9/16/2010-9/16/2011	Primary

[Add](#) | [Delete](#)

Token Encryption		
Used For	Type	Effective Dates
<input type="checkbox"/> Web Service Certificate Auth	X.509 Certificate	9/16/2010-9/16/2011

[Add](#) | [Delete](#)

Token Decryption

No token decryption certificates. Click [Add Token Decryption Certificate](#) to configure one.

Figure 7-11. Add certificate

- 4. In ADFS 2.0, add your ACS service namespace as a Relying Party, because from ADFS 2.0 perspective, it is sending SAML token to a Relying Party (RP), but essentially it is sending the SAML token to a claims mapping service that will in-turn transform these claims into another set of token and claims expected by the final Relying Party. Here ACS acts as a Relying Party and an STS, it is also called RP-STS.
- 5. Next, add claim rules for the Access Control Service namespace in ADFS 2.0.
- 6. In the Rule Groups section of the ACS service namespace management portal, you can add new rules for your Relying Party that map input claims to output claims as shown in Figure 7-12.

Add Claim Rule

Specify how an input claim is transformed into an output claim delivered to your relying party application. For more information, see [help on claim rules](#).

If

Claim issuer
Select the claim issuer that this rule applies to. The available input claim types listed below will filter if an identity provider is selected. Select Access Control Service to handle service identities and claims output from other rules. [Learn more](#)

☒ Identity Provider: Google

☐ Access Control Service

And

Input claim type
Select or enter an input claim type. An input claim type can be a string or a URL. [Learn more](#)

☒ Any

☐ Select type: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress

☐ Enter type:

Example: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name

And

Input claim value
Select or enter an input claim value. [Learn more](#)

☒ Any

☐ Enter value:

Then

Output claim type
An output claim type can be a string or a URL. [Learn more](#)

☒ Pass through input claim type

☐ Select type: http://docs.oasis-open.org/wsrf/authorization/200706/claims/action

☐ Enter type:

Example: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name

Output claim value
The input claim value can be passed through or replaced with a custom value. [Learn more](#)

☒ Pass through input claim value

☐ Enter value:

Rule Information

Description (optional)
Enter a description for this claim rule.

Figure 7-12. Add claim rule

Global Identity Providers

In ACS version 2.0, you can choose to use Windows Live, Google, Yahoo!, and Facebook as identity providers. For adding Windows Live, Google, or Yahoo! as Identity Providers, simply select the desired Identity Providers as shown in Figure 7-13.

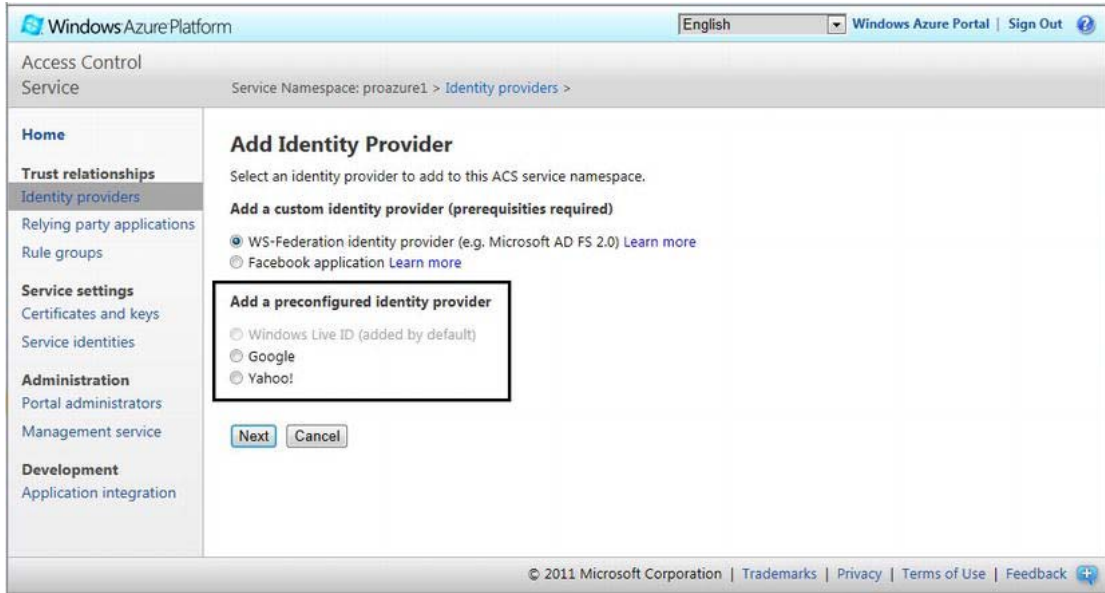


Figure 7-13. Add out-of-the-box identity providers

After you have added the desired Identity Providers, you can go to the Rule Groups section, add a new Rule Group, and click the Generate button to generate default claims for the Windows Live, Google, and Yahoo! Identity providers as shown in Figure 7-14.

Edit Rule Group

Use the fields below to specify how an input claim from a particular identity provider is transformed into an output claim that is delivered to your relying party application.

Rule Group Details

Name

Enter a name for the rule group.

Default Rule Group for simple80local

Used by the following relying party applications

simple80local

Save

Cancel

Generate

Add | Delete

Rules			
<input type="checkbox"/>	Output Claim	Claim Issuer	Rule Description
<input type="checkbox"/>	emailaddress	Google	Passthrough "emailaddress" claim from Google as "emailaddress"
<input type="checkbox"/>	emailaddress	Yahoo!	Passthrough "emailaddress" claim from Yahoo! as "emailaddress"
<input type="checkbox"/>	name	Google	Passthrough "name" claim from Google as "name"
<input type="checkbox"/>	name	Yahoo!	Passthrough "name" claim from Yahoo! as "name"
<input type="checkbox"/>	nameidentifier	Google	Passthrough "nameidentifier" claim from Google as "nameidentifier"
<input type="checkbox"/>	nameidentifier	Windows Live ID	Passthrough "nameidentifier" claim from Windows Live ID as "nameidentifier"
<input type="checkbox"/>	nameidentifier	Yahoo!	Passthrough "nameidentifier" claim from Yahoo! as "nameidentifier"

Figure 7-14. Edit Rule Group

The step-by-step process for adding Facebook application as an Identity Provider to ACS is as follows:

1. Create a Facebook account.
2. Install the Facebook Developer application (www.facebook.com/apps/application.php?id=2345053339) as shown in Figure 7-15.

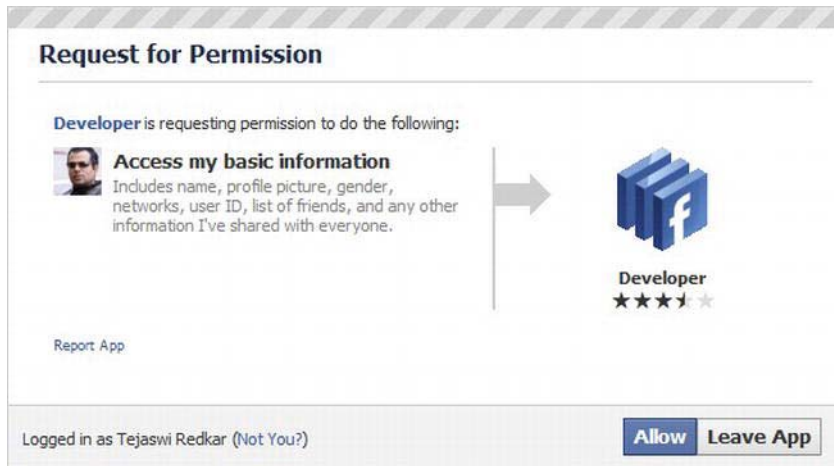


Figure 7-15. Create Facebook Developer application

3. Click Set Up New Application to create a new application.
4. In the Application Name field, enter a display name for your application.
5. Click Create Application.
6. Click the Web Site tab in the left panel.
7. In the Site URL field, enter the HTTPS URL of your ACS Service Namespace (e.g., `https://yourservicenamespace.accesscontrol.appfabriclabs.com/`).
8. From the resulting page, copy the Application ID and Application Secret as shown in Figure 7-16. This information will be used in the ACS portal when you configure Facebook as an Identity Provider.

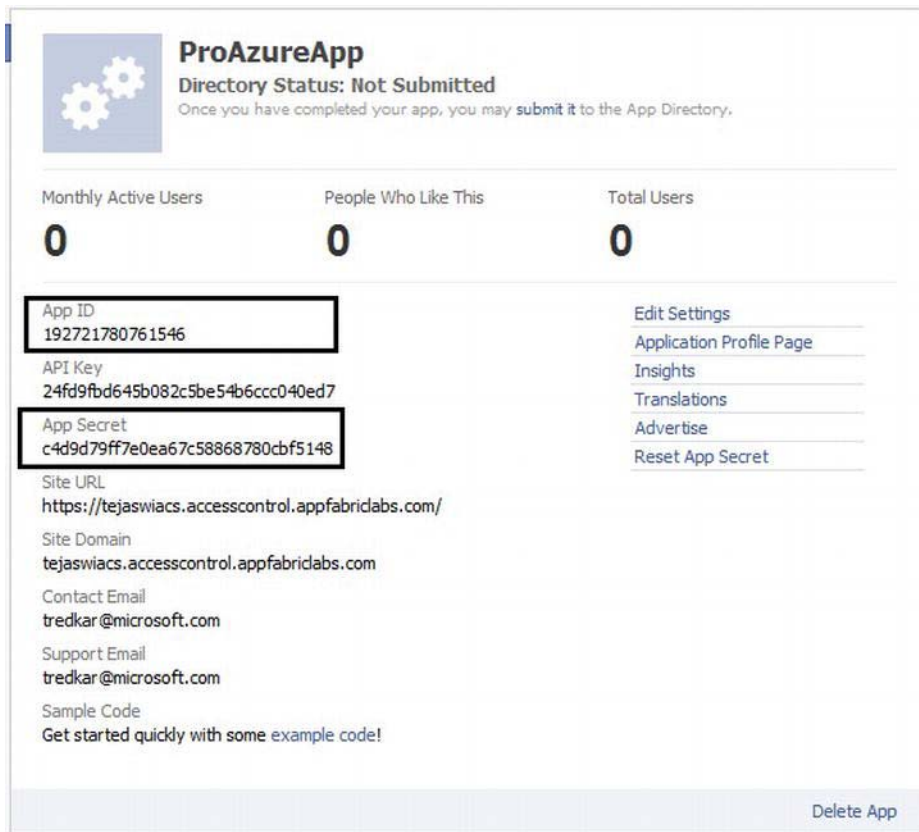


Figure 7-16. Facebook Application ID and Secret

9. On the ACS management portal main page, click Identity providers.
10. Click Add.
11. Select Facebook application, and click Next.
12. In the Application ID field, enter the App ID key copied from your Facebook application page.
13. In the Application secret field, enter the App Secret copied from your Facebook application page.
14. Optionally, in the Application permissions field, you can add any additional permission documented here <http://developers.facebook.com/docs/authentication/permissions>.
15. The Login link text field is used to customize the text displayed by the ACS login page.

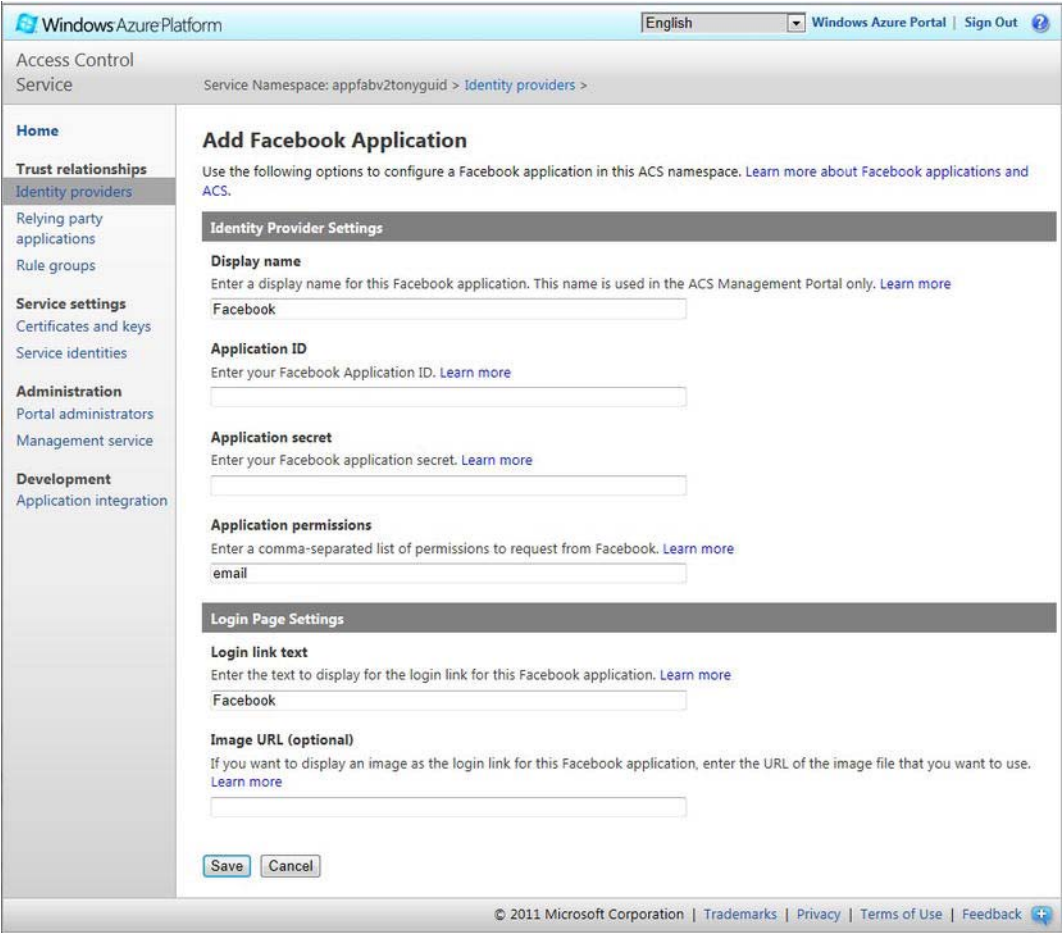


Figure 7-17. Add Facebook Identity Provider

- 16. In the Used By section, select any existing relying party applications with which you want to associate the Facebook identity provider. This causes the Facebook identity provider to appear on the login page for that application, and enables claims to be delivered from the identity provider to the application.

■ **Note** Make sure you generate the rules for your rule groups after you have added a new Identity Provider. ACS version 2.0 does not automatically generate rule groups when you add a new Identity Provider. See Figure 7-18.

Edit Rule Group

Use the fields below to specify how an input claim from a particular identity provider is transformed into an output claim that is delivered to your relying party application.

Rule Group Details

Name
Enter a name for the rule group.

Used by the following relying party applications
simple80local

[Generate](#) | [Add](#) | [Delete](#)

<input type="checkbox"/>	Output Claim	Claim Issuer	Rule Description
<input type="checkbox"/>	AccessToken	Facebook	Passthrough "AccessToken" claim from Facebook as "AccessToken"
<input type="checkbox"/>	emailaddress	Google	Passthrough "emailaddress" claim from Google as "emailaddress"
<input type="checkbox"/>	emailaddress	Yahoo!	Passthrough "emailaddress" claim from Yahoo! as "emailaddress"
<input type="checkbox"/>	emailaddress	Facebook	Passthrough "emailaddress" claim from Facebook as "emailaddress"
<input type="checkbox"/>	expiration	Facebook	Passthrough "expiration" claim from Facebook as "expiration"
<input type="checkbox"/>	name	Google	Passthrough "name" claim from Google as "name"
<input type="checkbox"/>	name	Yahoo!	Passthrough "name" claim from Yahoo! as "name"
<input type="checkbox"/>	name	Facebook	Passthrough "name" claim from Facebook as "name"
<input type="checkbox"/>	nameidentifier	Google	Passthrough "nameidentifier" claim from Google as "nameidentifier"
<input type="checkbox"/>	nameidentifier	Windows Live ID	Passthrough "nameidentifier" claim from Windows Live ID as "nameidentifier"

1 of 2 ▶

Figure 7-18. Facebook claims

17. Click Save.

Relying Party

Relying Party is the consumer of claims generated by the identity providers. Relying Party validates claims and provides appropriate privileges to the user of the application. Claims can consists of any information about the user, but when you are using ACS, it is important to provide a single view of all the incoming claims irrespective of the type of identity provider. You can acquire claims from Windows Live ID, Google Id and Facebook application, but ACS can transform these claims into a single view your relying party is configured to consume. Figure 7-19 illustrates the page for creating a new Relying Party on the ACS service namespace portal.

Trust relationships

Relying party applications

Identity providers

Rule groups

Service Settings

Certificates and keys

Service identities

Administration

Portal administrators

Management service

Development

Application integration

Add Relying Party Application

Use the fields below to configure a relying party application in this Access Control Service namespace.

Relying Party Application Settings

Name

Enter a display name for this relying party application.

ProAzureSimpleWeb

Example: fabrikam.com

Mode

Click one of these options to configure your relying party application settings manually or to upload a WS-Federation metadata document with the settings for your relying party application. [Learn more](#)

☒ Enter settings manually

☐ Import WS-Federation metadata

Realm

Enter the URI that the security token issued by Access Control Service will apply to. [Learn more](#)

http://localhost:9001/

Example: https://www.fabrikam.com (http://localhost is allowed)

Return URL (optional)

Enter the URL that Access Control Service will return the security token to. [Learn more](#)

Example: https://www.fabrikam.com/index.aspx (http://localhost is allowed)

Error URL (optional)

Enter the URL that Access Control Service will post to if an error occurs during sign-in. [Learn more](#)

Example: https://www.fabrikam.com/error.aspx (http://localhost is allowed)

Token format

Select a token format for Access Control Service to use when issuing security tokens to this relying party application. [Learn more](#)

SAML 2.0

SAML 2.0

SAML 1.1

SWT

None

Token lifetime (secs)

Specify the amount of time for a security token issued by Access Control Service to remain valid. [Learn more](#)

600

Authentication Settings

Identity providers

Select the identity providers to use with this relying party application. [Learn more](#)

☒ Google

☒ Windows Live ID

☒ Yahoo!

Rule groups

Select rule groups for this relying party application to use when processing claims. [Learn more](#)

☒ Create new rule group

Token Signing Options

Token signing

Select whether to sign SAML tokens using the default service namespace certificate, or using a custom certificate specific to this application. Only service namespace certificates are published in the WS-Federation metadata for this Access Control Service namespace. [Learn more](#)

Use service namespace certificate (standard)

Use service namespace certificate (standard)

Use a dedicated certificate

Save

Cancel

Figure 7-19. Create a new Relying Party application

356

While adding a new Relying Party application, make sure you pay attention to the Realm and Return URL. Realm is the URI for which ACS will generate tokens. This is usually the URI of your application or a URN. ACS matches this value to the Realm information (wtrealm in WS-Federation protocol and applies_to in OAuth WRAP protocol) in the token generated by the identity provider. ACS will issue a token for your Relying Party application only if the realm matches between the configured value and the token received from the identity provider. ACS can also do a prefix match like `www.tejaswiredkar.com` and `www.tejaswiredkar.com/home`, but will not match `www.tejaswiredkar.com` to `http://tejaswiredkar.com`.

Rule Groups

Rule Groups are used for mapping input claims from identity provider to the output claims for the Relying Party. The Rule Groups section in the ACS portal allows you to create mapping for all the Identity Providers you have added to your namespace, including the ACS itself. Figure 7-20 illustrates the Edit Claims page for the Facebook name claim.

Edit Claim Rule

Specify how an input claim is transformed into an output claim delivered to your relying party application. For more information, see [help on claim rules](#).

If

Claim issuer

Select the claim issuer that this rule applies to. The available input claim types listed below will filter if an identity provider is selected. Select Access Control Service to handle service identities and claims output from other rules. [Learn more](#)

Identity Provider:

Facebook

Access Control Service

And

Input claim type

Select or enter an input claim type. An input claim type can be a string or a URI. [Learn more](#)

Any

Select type:

http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name

Enter type:

Example: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name

And

Input claim value

Select or enter an input claim value. [Learn more](#)

Any

Enter value:

Then

Output claim type

An output claim type can be a string or a URI. [Learn more](#)

Pass through input claim type

Select type:

http://docs.oasis-open.org/wsrfed/authorization/200708/claims/action

Enter type:

Example: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name

Output claim value

The input claim value can be passed through or replaced with a custom value. [Learn more](#)

Pass through input claim value

Enter value:

Rule Information

Description (optional)

Enter a description for this claim rule.

Passsthrough "name" claim from Facebook as "name"

Save

Cancel

Figure 7-20. Edit Claim page

The Edit Claim Rule page is structured to guide you through the mapping of claims between input claims issuer and output claim receiver. You can chain the claims processing by choosing ACS as an input to itself. Complex claims mapping needs attention when you are working in multi-tenant authentication and authorization environments.

Certificates and Keys

In this section of the portal, you can manage the certificates and keys used for encryption, decryption, and token signing.

Token Signing

In ACS, tokens are signed using either an X.509 certificate or a symmetric key, depending on your token format. SAML tokens are signed with X.509 certificates, and SWT tokens are signed with a 256-bit symmetric key.

Which token format should you use? As usual, it depends. However, consider that SAML tokens are the default for Windows Identity Foundation applications, and are compatible with many protocols, including WS-Federation and WS-Trust. SWT tokens are also compatible with many protocols, including OAuth WRAP and WS-Federation.

When you create your namespace, a Symmetric Key and a X.509 certificate will be provisioned by default. You can use these to sign tokens for all relying party applications. However, if you wish to have a specific key or certificate for a specific relying party application, you can add that certificate or key as well, and specify the relying party application for which the key or certificate applies. See Figure 7-21.

Windows Azure Platform | Windows Azure Portal | Sign Out

Access Control Service | Service Namespace: proazure-1 > Certificates and keys >

Home

Trust relationships

- Identity providers
- Relying party applications
- Rule groups

Service settings

- Certificates and keys**
- Service identities

Administration

- Portal administrators
- Management service

Development

- Application integration

Add Token-Signing Certificate or Key

Add a new token-signing certificate or key. Token-signing certificates and keys are used to sign tokens that are issued to relying party applications. [Learn more about certificates and keys.](#)

Used for

Select the relying party application that this certificate or key is used for. ACS uses a Service Namespace certificate or key to sign tokens if no certificates or keys are present for a specific relying party application. Service Namespace certificates are also used to sign the WS-Federation metadata published by ACS. [Learn more](#)

☒ Relying Party Application: **ACSMachineInfo**

☐ Service Namespace (Fallback for all relying party applications)

Type

Select an option in the list to sign tokens with a 256-bit symmetric key or an X.509 certificate. [Learn more](#)

X.509 Certificate

Certificate

Browse for an X.509 certificate with a private key (.pfx file) for token signing. Enter the password for the .pfx file in the Password box.

If you do not have a certificate and you are running Windows, download MakeCert.exe as part of the Windows SDK. Run the following command to generate a self-signed certificate in your Personal certificate store, then export the private key and upload it. [Learn more](#)

MakeCert.exe -r -pe -n "CN=proazure-1.accesscontrol.windows.net" -sky exchange -ss my

File: **Browse...**

Password:

Primary

Select this option to set this certificate as the primary token-signing certificate for the selected relying party application or service namespace. [Learn more](#)

☒ Make Primary

Save **Cancel**

Figure 7-21. Adding a certificate for a specific Relying Party Application

CREATING YOUR OWN CERTIFICATES

In a claims-based identity model, X.509 certificates are used by all the participating parties: STS, ACS, and the relying party. X.509 certificates are used to encrypt and/or decrypt SAML tokens and also to validate

claims sent from one party to another. Most of the examples in MSDN and training kits use a predefined set of certificates that can cause conflicts when used by multiple developers and testers in the same environment. The following are the steps you can use to create your own certificates so you don't have to rely on the prepackaged certificates in sample applications:

1. Start the Visual Studio command prompt as an administrator.
2. Run the following command to create a temporary certificate:

```
makecert -n "CN=ProAzure" -r -sky exchange -sv ProAzure.pvk ProAzure.cer
```

3. Run the following command to create a certificate that is digitally signed and authorized by ProAzure:

```
makecert -sk ProAzureSignedCA -sky exchange -iv ProAzure.pvk -n "CN=ProAzureSignedCA" -ic ProAzure.cer ProAzureSignedCA.cer -sr localmachine -ss My
```

4. Use MMC to Import the ProAzure.cer certificate into the Trusted Root Certificate Authorities folder of the local machine certificate store. You can start MMC from Start ► Run ► mmc.exe. Then, choose File ► Add/Remove Snap-In ► Certificates.
5. From MMC, import ProAzureSignedCA.cer into the certificates personal folder of the local machine certificate store.
6. Export the certificate to distribute it to the outside world, using the pvk2pfx.exe tool from the Visual Studio .NET\Tools\bin folder:

```
pvk2pfx.exe -pvk ProAzure.pvk -spc ProAzure.cer
```

7. If you're hosting your service in IIS and would like to give permissions to certificates to specific accounts, see the WinHttpCertCfg.exe certificate configuration tool at [http://msdn.microsoft.com/en-us/library/aa384088\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa384088(VS.85).aspx).

Token Encryption

For web services that use proof-of-possession tokens over WS-Trust, token encryption is required. ACS can encrypt any SAML 1.1 or 2.0 token to send to your relying party application. Tokens are encrypted using X.509 certificates, so you will need to upload these through the portal.

Token Decryption

If you have a WS-Federation identity provider that sends encrypted tokens to ACS (such as ADFS 2.0), ACS will need to decrypt those tokens. Once again, we turn to X.509 certificates. The certificate is uploaded to ACS via the portal. The identity provider will then obtain the public key from the ACS federation metadata endpoint, and use that public key to encrypt the token. ACS will then decrypt the token using the private key.

Service Identities

Service Identities allow an authentication request to be made directly against ACS instead of an identity provider. Essentially, they are hard-coded identities inside ACS that will produce a token if presented with the right credentials. These are useful when an autonomous application or service needs access to a

relying party application or service. Typical scenarios are REST web services or OAuth WRAP protocol, where a client would request a token from ACS to present to an ACS-integrated service.

■ **Note** Service identities are not intended to be used as end-user credentials.

Three types of credentials can be associated with a service identity: Symmetric key, Username/Password, and X.509 Certificate.

To create a Service Identity, navigate to the Service Identities section of the portal, and then click Add. Set up the name, credential type, and credential information. Once saved, distribute the credential information along with the reference endpoint to the necessary application or service.

Portal Administrators

In ACS, you have the capability of assigning users administrative rights for specific ACS namespaces. The administrator can then access ACS through the ACS Management Portal.

In order to add a user, the identity provider that contains that user's credential must be added to the namespace. For example, if a user is authenticated using ADFS 2.0, then that identity provider must be added.

Once the identity provider is added, navigate to the Portal Administrators area and click Add. Then select the identity provider, claim type, and claim value. See Figure 7-22.

■ **Note** it is critically important to pick a unique claim type. If the claim type contains values that could be used by more than one user, then access to the portal will be granted to all users who present this claim value.

Access Control Service Service Namespace: proazure-1 > Portal administrators >

Home

Trust relationships

- Identity providers
- Relying party applications
- Rule groups

Service settings

- Certificates and keys
- Service identities

Administration

- Portal administrators**
- Management service

Development

- Application integration

Add Portal Administrator

Use the following options to grant administrative access to the ACS Management Portal to users from selected identity providers. [Learn more about ACS portal administrators.](#)

Identity provider

Specify the identity provider of the user account that you want to make a portal administrator. After the administrative account is created, this identity provider appears on the ACS Management Portal login page. [Learn more](#)

Windows Live ID ▼

Identity claim type

Specify a claim type that represents a unique identifier of the user account that you want to make a portal administrator. An email address is the default claim type for identifying individual users. [Learn more](#)

http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress ▼

Identity claim value

Enter a value for the claim type, for example: user1@contoso.com [Learn more](#)

Description (optional)

Type a description for this administrator.

Figure 7-22. Adding a portal administrator

Management Service

This section of the portal allows you to administer the accounts that can access the management service for your namespace. You provide a name and option description, then can define one of three credential types: Symmetric Key, Username/Password, or X.509 Certificate.

You can then distribute this credential information to users/applications that need to access the Management Service API via the reference endpoint provided in the Application Integration section.

Application Integration

This section provides you with the means necessary to integrate ACS into your application. There are three sections: login pages, SDKs and documentation, and endpoint references.

Login Pages

This is where you define what users see when they login to your application through ACS. There are 2 options: link to an ACS-hosted login page or host the login page as part of your application.

Linking to an ACS-Hosted Login Page

In this case, ACS hosts a default login page, as shown in Figure 7-23. The URL to this page is provided on the screen. Link to that page anywhere you expect users to login to your application. The screen is very

simple and not pretty, but it is easy and effective. The Global identity providers will appear as buttons, as will WS-Federation providers.

The image shows a web form titled "Sign in to My Application". Below the title is the instruction "Sign in using your account on:". Underneath this instruction are five rectangular buttons, each containing the name of an identity provider: "Windows Live ID", "Yahoo!", "Facebook", "Google", and "Contoso Corp.". The buttons are stacked vertically.

Figure 7-23. ACS-hosted login page

However, consider the scenario where an application is serving many enterprise customers using WS-Federation. It is more than likely that you will not want to show all of the WS-Federation providers. The solution for this is to define e-mail suffixes for the WS-Federation providers in the Management Portal. Once you have done this, the screen will look different, as shown in Figure 7-24.

The image shows a web form titled "Sign in to My Application". It is divided into two main sections by a vertical line. The left section is titled "Sign in using your account on:" and contains four buttons: "Windows Live ID", "Yahoo!", "Facebook", and "Google". The right section is titled "Sign in using your e-mail address:" and contains a text input field and a "Submit" button. Between the two sections, centered vertically, is the word "Or".

Figure 7-24. ACS-hosted login page with e-mail suffixes defined

Once the user logging in enters their email address, ACS will use the e-mail domain suffix to route the authentication request to the appropriate identity provider.

Hosting the login page as part of your application

You have the option to download a sample login page from this screen, which will provide you with an HTML page identical to the default login page provided by ACS. It uses AJAX to call a JSON-encoded feed to acquire the token. You can simply modify the look and feel of this page to provide a better-looking screen, or build your own page that integrates with the ACS-hosted JSON feed. The URL for the JSON feed is provided on the screen.

Here is an example URL from the proazure-1 namespace:

```
https://proazure-1.accesscontrol.windows.net:443/v2/metadata/IdentityProviders.js?protocol=wsfederation&realm=https%3a%2f%2flocalhost%2fACSWebApp&reply_to=https%3a%2f%2flocalhost%2fACSWebApp%2fDefault.aspx&context=&request_id=&version=1.0&callback=
```

Let’s take a look at the individual parts of this URL listed in Table 7-1.

Table 7-1. ACS JSON feed URI elements

Feed URI	Required	Change ‘proazure-1’ to the name of your Windows Azure AppFabric service namespace.
protocol	Required	ACS requires the communication protocol to be ‘wsfederation.’
Realm	Required	The realm specified for the relying party application in the Management Portal.
version	Required	ACS requires a value of 1.0. If you are confused about ACS v1 and v2, note that ‘v2’ is in the path of the URL.
reply-to	Optional	Return URL. If omitted, will use default Return URL you specified for the relying party application in the Management Portal.
context	Optional	Simple pass-through of additional context information. ACS does not use the data in this parameter.
callback	Optional	JavaScript callback function to execute upon receiving a response. The JSON feed is passed into this function as an argument.

The response to this call will be a JSON array containing an array for each provider, similar to the following:

```
[{"Name":"Windows Live ID","LoginUrl":"https://...","LogoutUrl":"https://...","ImageUrl":"https://...","E-mailAddressSuffixes":[]}, {"Name":"My ADFS 2.0 Provider","LoginUrl":"https://...","LogoutUrl":"https://...","ImageUrl":"","E-mailAddressSuffixes":["contoso.com"]}]
```

Each array item represents an identity provider, and will contain the information listed in Table 7-2.

Table 7-2 – JSON ACS Response elements

Name	Display name you entered in the Management Portal.
LoginUrl	URL to use when making requests of the identity provider.
LogoutUrl	Allows for signing out from the identity provider. Currently only supported for ADFS 2.0 and Live ID. Will return empty for other providers.
ImageUrl	An image that should be displayed for that provider. Entered into the Management Portal.
E-mailAddressSuffixes	E-mail domain suffixes added for this identity provider.

If you download the sample login page from the ACS portal, you will see the following HTML embedded in the page:

```
<!-- This script gets the HRD metadata in JSON and calls the callback function which renders the links -->
<script src="https://proazure-1.accesscontrol.windows.net:443/v2/metadata/IdentityProviders.js?protocol=wsfederation&realm=https%3a%2f%2flocalhost%2fACSWebApp&reply_to=&context=&request_id=&version=1.0&callback=ShowSignInPage" type="text/javascript"></script>
```

Note the callback. This means that the JSON generated from this call will be passed to a JavaScript function that will render the UI elements for each identity provider:

```
// This function will be called back by the HRD metadata, and is responsible for displaying the sign-in page.
function ShowSignInPage(json) {
    // Code to iterate through identity provider array and render UI elements for each
}
```

SDKs and Documentation

Self-explanatory, this contains links to SDKs and documentation that will help you create ACS-integrated applications.

Endpoint References

This section is critical, it contains the endpoint references you will need to incorporate your application. Depending on whether you are using OAuth or WS-Federation, you will need to reference one or more of these endpoints in your application. See Figure 7-25.

Endpoint Reference	
Management Service	https://proazure-1.accesscontrol.windows.net/v2/mgmt/service
Management Portal	https://proazure-1.accesscontrol.windows.net/
OAuth WRAP	https://proazure-1.accesscontrol.windows.net/WRAPv0.9
WS-Federation Metadata	https://proazure-1.accesscontrol.windows.net/FederationMetadata/2007-06/FederationMetadata.xml
WS-Metadata Exchange	https://proazure-1.accesscontrol.windows.net/v2/wstrust/mex

Figure 7-25. ACS endpoint references

Programming Access Control Service Applications

This section discusses some end-to-end examples of configuring and using ACS in web applications and web services. I will cover a relatively simple scenario that should demonstrate the power of ACS. The example is a web application that will use multiple identity providers, both global (LiveID, Google, Yahoo!) and WS-Federation. In addition, we will define rules that will allow us to use the claims presented by the tokens to perform authorization.

■ **Note** Before proceeding in this section, it is important to have the pre-requisites installed for creating the development environment.

Visual Studio 2010 (any edition) (www.microsoft.com/visualstudio/en-us/products)

.NET Framework 4.0 or .NET Framework 3.5 SP1 with KB's 976126 or 976127 applied

Windows Identity Foundation Runtime (<http://support.microsoft.com/?kbid=974405>)

Windows Identity Foundation SDK (www.microsoft.com/downloads/en/details.aspx?familyid=C148B2DF-C7AF-46BB-9162-2C9422208504&displaylang=en)

In a typical ACS solution, the development workflow is as follows:

1. Create a service namespace.
2. Define the identity providers.
3. Define the relying party applications.
4. Create rule groups and rules for mapping input claims to output claims.

5. Modify the relying party application to integrate with ACS.

The first four topics have already been covered, so in this section I will focus on the steps required to modify your application integrate with ACS.

■ **Note** Some of the examples in this section require Windows Identity Foundation (WIF). Before running the examples, please install WIF and the WIF SDK.

Passive Federation with ACS

In Passive Federation, a web browser is the access point for the requestor for interacting with a relying party, ACS and the identity provider. The web browser loads web pages from the relying party web application and the requestor interacts with the relying party only through the web browser. The relying party, ACS and the identity provider redirects the requestor's browser through several stages of the claims-based identity model.

Web Application: Multiple Identity Providers using ACS

In this example, we will create a simple web application that uses ACS to federate identity management to multiple providers. We will federate with the “out-of-the-box” providers: Windows Live, Google, and Yahoo!, as well as a custom WS-Federation STS (simulating ADFS 2.0).

The sample code for this example can be found in the ProAzureACSFederation solution in the code samples for this chapter.

Configure Access Control Service Using Management Portal

Configuring ACS in this manner requires a series of steps, which are outlined in the following sections.

Add Identity Providers

Per the steps described in the Identity Providers section earlier in this chapter, add Identity providers to your namespace for Windows LiveID, Google, and Yahoo!

■ **Note** If you don't already have existing accounts with these providers, you will need to set up accounts for testing,

Add Relying Party Application

Navigate to Relying Party Applications. Click the Add link to add a new relying Party application, and fill in the form with the following information:

1. **Name:** ProAzureACSFederation
2. **Mode:** Enter settings manually
3. **Realm:** https://127.0.0.1:81 (this may change based on how the Compute emulator starts your application)
4. **Return URL:** https://127.0.0.1:81/ProAzureACSWeb/Default.aspx (again, this may change)
5. **Error URL:** leave the field empty
6. Token format: SAML 2.0
7. Token encryption policy: None
8. Token lifetime (secs): 600
9. Identity providers: Select all
10. **Rule groups:** Create New Rule Group
11. **Token signing:** Use service namespace certificate (standard)

Create Rule Groups and Rules for Mapping Claims

Follow these steps to create rule groups:

1. Click Rule Groups. There will be a default rule group created for your application, in this case names Default Rule Group for ProAzureACSWeb (Figure 7-26).
2. To create, modify, and delete rules, click the link for this rule group. You will notice upon first look that no rules will have been created.
3. Click the Generate link to generate a set of base rules for each provider.

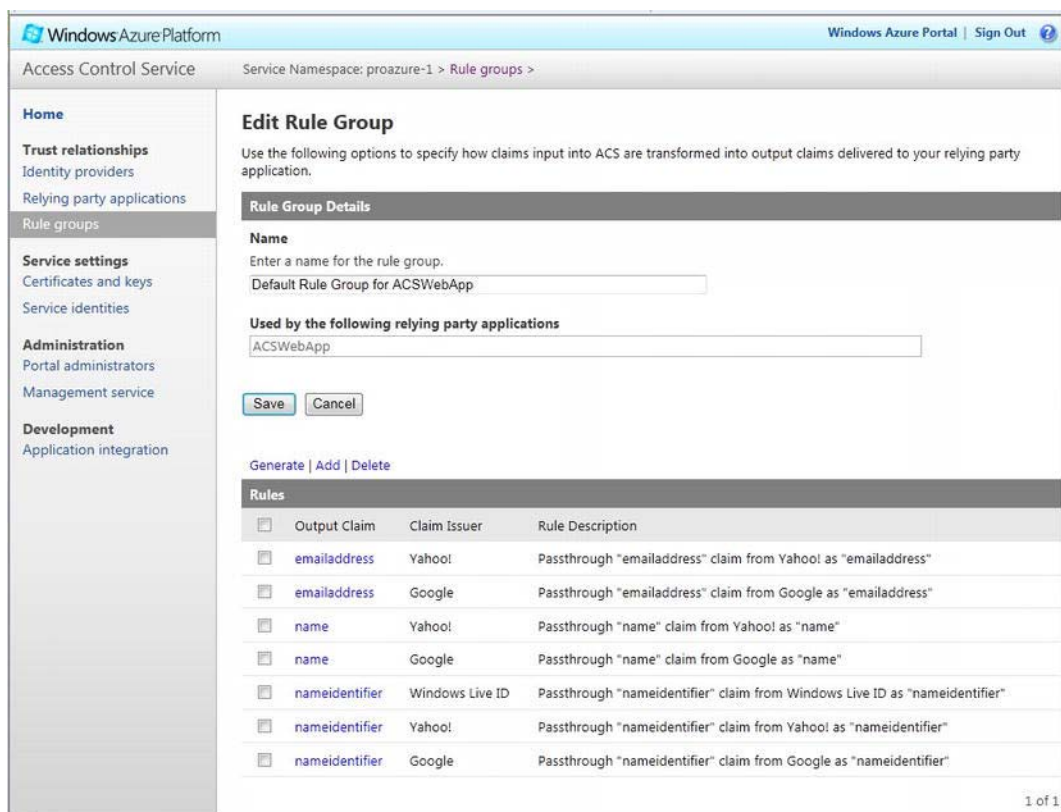


Figure 7-26. Default rule group

Modify Relying Party Application to Integrate with ACS

To integrate with ACS, a web application needs to forward all unauthenticated requests to ACS. We will use WS-Federation to communicate from our web application to ACS. Therefore, the application will receive a consistent authentication token to use for authorization purposes.

The first thing we need is the WS-Federation Endpoint from your ACS namespace:

1. Click Application Integration on the left-hand navigation in the portal, and you will be provided with this endpoint.
2. Copy the endpoint text from the WS-Federation Metadata section as shown in Figure 7-27.

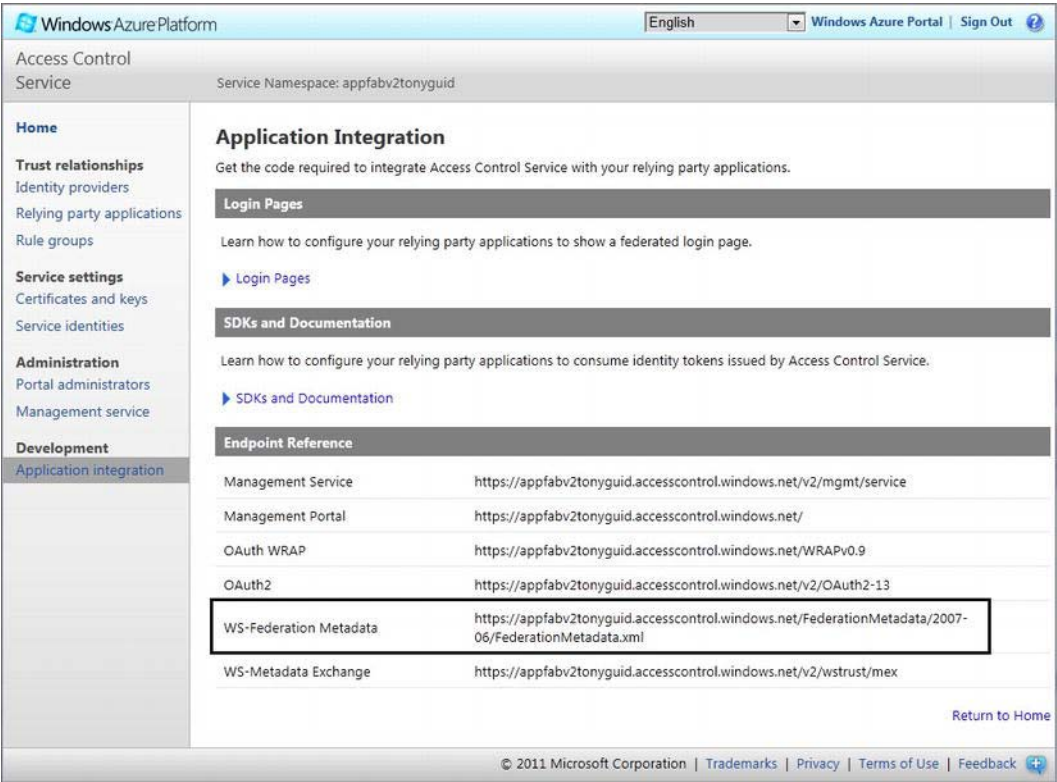


Figure 7-27. WS-Federation Metadata endpoint text

Once we have the endpoint, we need to create an STS reference in our web application.

3. Right-click on the application and choose Add STS Reference.
4. Use the pre-populated field on the welcome screen. Click Next to go to the Security Token Service page.
5. Choose Use an Existing STS, and enter the endpoint reference you copied into the textbox for STS WS-Federation metadata document location, as shown in Figure 7-28.

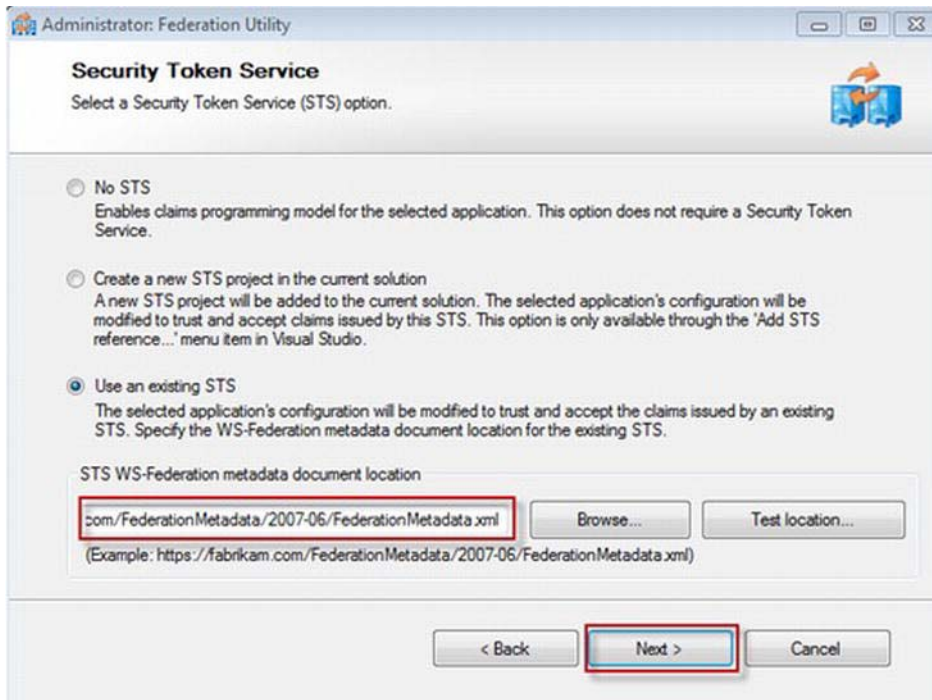


Figure 7-28. Entering WS-Federation metadata location information

Accept the defaults through the rest of the screens, and Visual Studio will configure your web application to authenticate against ACS. It will add the WIF configuration to your web.config by adding a <microsoft.identitymodel> section, as well as configuration to restrict access to only authenticated users, as well as adding a configuration that allows all users to access the FederationMetadata.xml file. This is required so the metadata can be downloaded prior to authenticating.

When you start your application now, you should see the ACS-hosted page, with buttons for your identity providers, which will require you to authenticate with them before accessing your application.

Adding a WS-Federation Provider

Now that we have configured global providers, let's say that administrators of your application must be authenticated from your own identity provider. Inside your domain, these users are assigned to an admin group. When the claim is presented to ACS, users who enter from this group will have a claim of type Group with a value of admin. However, our application is using role-based security, and is expecting a claim of type 'role' with a value of 'admin' or 'user.' Based on this claim, administrative access to the application will be granted or denied.

■ **Note** For this example, I have used the 'SelfSTS' tool, provided as part of the Windows Azure Platform Training Kit. This tool will simulate a running WS-Federation provider, saving you the trouble of setting up and configuring

one yourself. The latest version as of the release date of this book can be found here:

<http://msdn.microsoft.com/en-us/gg271268>. Once the kit is downloaded and installed, the SelfSTS tool can be found at <install directory>\Labs\ACS2Federation\Source\Assets\SelfSTS1.

Run the SelfSTS utility. The first thing we are going to do is add the role claim to the set of claims that are presented by the provider. Click Edit Claim Types and Values, ensure the claims match the values shown in Figure 7-29.

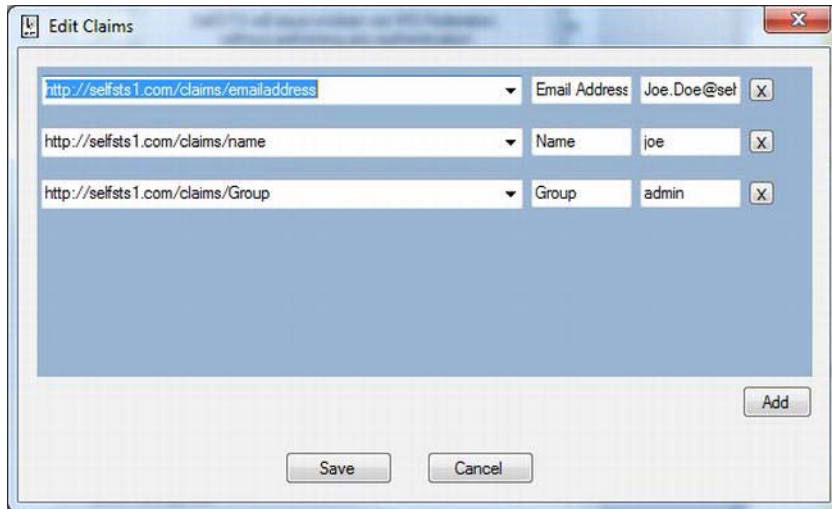


Figure 7-29. Claims to be presented by WS-Federation provider

Once that is completed, click Start so that SelfSTS will be running when we add it to ACS as an identity provider. Navigate to the ACS portal, add a new identity provider, and choose “WS-Federation identity provider.” Then enter the following information to complete the request:

- Display Name: SelfSTS1
- WS-Federation metadata: Choose File, then click the ‘Browse’ button and navigate to the SelfSTS1 directory on your file system, and select FederationMetadata.xml
- Login link text: SelfSTS1
- Image URL: leave blank
- E-mail domain names: selfsts1.com
- Relying Party Applications: ACSWebApp

Designing the Relying Party Claims

When you design a claims-based identity model, one of the important design tasks you must complete is designing claims for the relying party. The relying party is the web service or web application that you want to protect using a claims-based identity model. Most web services and web applications already have some kind of role-based authorization model that defines privileges for end users. In most cases, the role-based authorization model can be easily transformed into a claims-based model by converting the roles to claims; you can keep the privileges the same as in the role-based authorization model. One advantage of moving to the claims-based identity model is that you can remove the end user authentication from your web application. Your web service or web application processes the tokens issued by ACS and validates the claims issued by ACS regardless of the authentication method used to authenticate the end user.

■ **Note** In the interest of keeping the example conceptual to ACS, it's very simple. You can enhance this example to provide more complex scenarios.

Designing ACS Rules

After you design the claims for your web service, you need to design the input and output claims for ACS. I mentioned earlier that our application grants administrative access based on the role claim having a value of “admin.” However, in the SelfSTS tool, we configured the SelfSTS tool to present a “group” claim. So we will need to use the mapping capabilities of ACS to map the group claim to a role claim. In this example, only the SelfSTS is issuing this claim, so the mapping is simple; complex scenarios can have multiple input claims from multiple issuers that need to be mapped to a single set of output claims expected by the relying party. For this example, let's also assume we want to still keep the original claim, and add a second claim that is the mapped claim, so that we can see both the original and the mapped claim. Table 7-3 lists the input claim types and values with their corresponding output claim types and values. Figure 7-30 shows the view in the management portal.

Table 7-3. SelfSTS Claims Mapping

Rule Name	Input Claim Type	Input Claim Value	Output Claim Type	Output Claim Value
Group	Group	admin	group	admin
role	Group	admin	role	admin

Windows Azure Platform Windows Azure Portal | Sign Out

Access Control Service Service Namespace: proazure-1 > Rule groups >

Home

Trust relationships

- Identity providers
- Relying party applications
- Rule groups**

Service settings

- Certificates and keys
- Service identities

Administration

- Portal administrators
- Management service

Development

- Application integration

Edit Claim Rule

Specify how an input claim is transformed into an output claim delivered to your relying party application. [Learn more about claim rules.](#)

If

Claim issuer

Select the claim issuer that this rule applies to. Select Identity Provider to process input claims from various identity providers, or select Access Control Service to process input claims from a service identity or another claim rule. [Learn more](#)

☒ Identity Provider: SelfSTS1

☐ Access Control Service

And

Input claim type

Select or enter an input claim type. The available input claim types will change based on the identity provider specified above. An input claim type can be a string or a URL, and is case-sensitive. [Learn more](#)

☐ Any

☒ Select type: http://selfsts1.com/claims/Group

☐ Enter type:

Example: `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name`

And

Input claim value

Select or enter an input claim value. Input claim values are case-sensitive. [Learn more](#)

☒ Any

☐ Enter value:

Then

Output claim type

An output claim type can be a string or a URL. [Learn more](#)

☐ Pass through input claim type

☒ Select type: http://schemas.microsoft.com/ws/2008/06/identity/claims/role

☐ Enter type:

Example: `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name`

Output claim value

The input claim value can be passed through or replaced with a custom value. [Learn more](#)

☒ Pass through input claim value

☐ Enter value:

Rule Information

Description (optional)

Enter a description for this claim rule.

Figure 7-30. Creating a claim that maps from Group input claim to Role output claim

Claims-Based Authorization

The final step in integrating our application is to add the authorization code. For our sample application, we have a few simple rules: we want administrators to see a link to an administrator page, and we want to prevent unauthorized access to the administrator age. The code for this is actually quite simple. In our `Page_Load` method, we add the following code:

```
// check to see if the user is in the admin role
if(User.IsInRole("admin"))
    EnableAdminAccess();
```

And in the EnableAdminAccess() method:

```
//Attribute protects method from unauthorized access
[PrincipalPermission(SecurityAction.Demand, Role="admin")]
private void EnableAdminAccess()
{
    lnkAdmin.Visible = true; // show link to admin page
}
```

Also, we want to ensure someone cannot access the admin page by entering the URL in the browser, so we add an attribute to the Page_Load method of the Admin.aspx page:

```
// Attribute protects against unauthorized access.
// User must belong to the "admin" role, no matter how they are authenticated.
// In this example, the role access was presented as a claim in the token from ACS.
[PrincipalPermission(SecurityAction.Demand, Role="admin")]
protected void Page_Load(object sender, EventArgs e)
{
}
```

Once you run your application in debug mode, you will be taken to the ACS-hosted login screen, which is shown in Figure 7-31. Note all three global providers are available, as well as a place to enter your e-mail address.

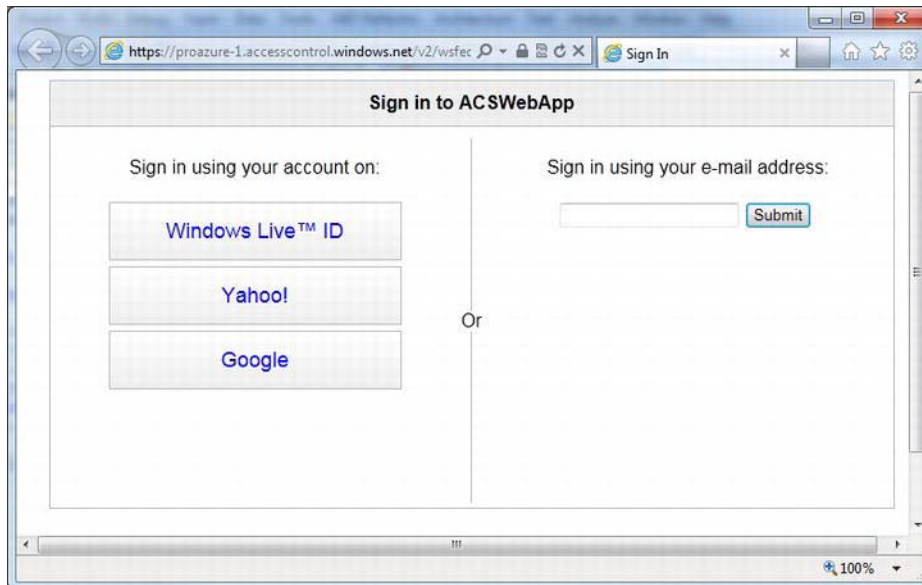


Figure 7-31. ACS-hosted login page, with e-mail domain suffix entry for WS-Federation providers

Make sure SelfSTS is running, then enter <something>@selfsts1.com and click Submit. It should take you to the default.aspx page shown in Figure 7-32, with the admin link enabled and visible:

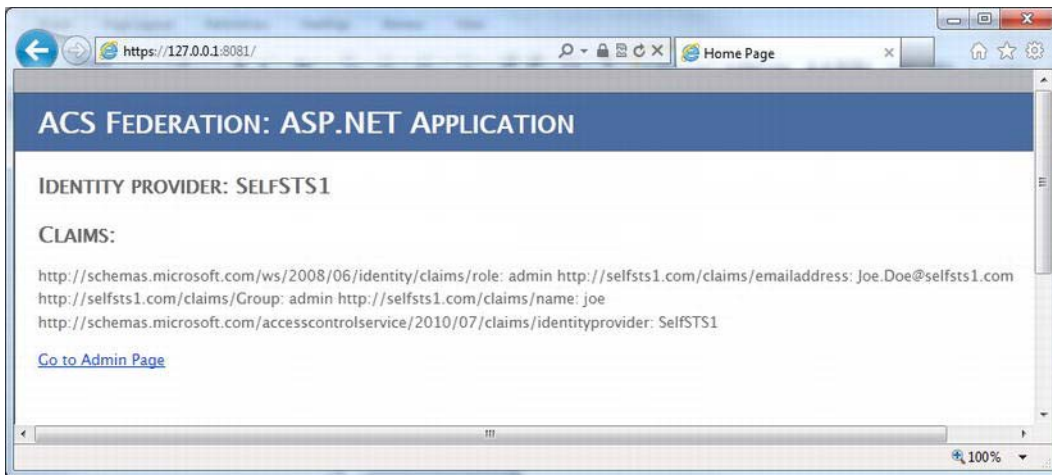


Figure 7-32. Default.aspx, with administrative access via claims presented through ACS mapping

Clicking on the Admin link will take you to the admin page. Now, stop and restart the application and choose another provider. Now the admin page link will be hidden. Try to navigate to it by appending 'admin.aspx' to the URL in the address bar. You should see the screen shown in Figure 7-33.

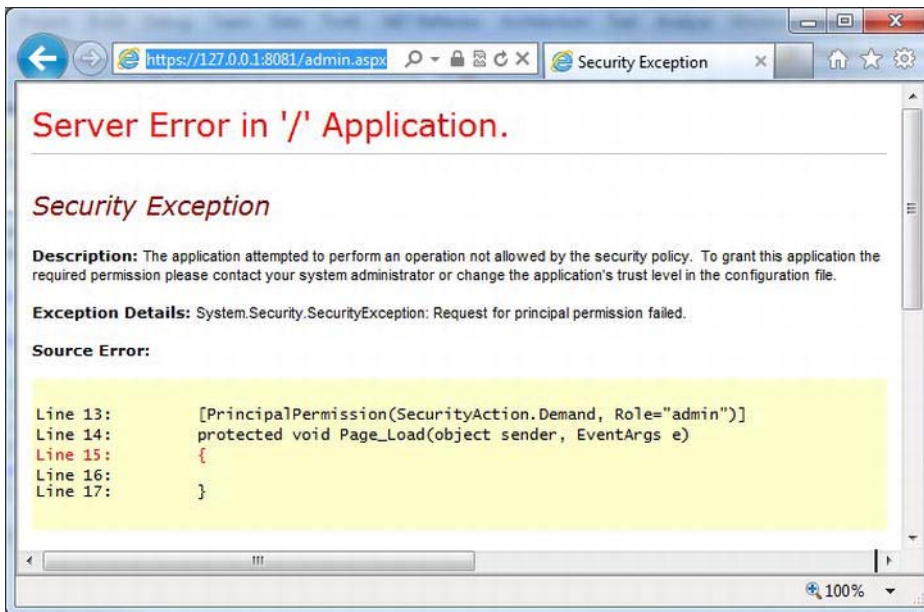


Figure 7-33. Access to admin page denied for non-admin users

In this example, we've managed to federate our identity management to four providers (both global and WS-Federation), and allow for administrative user management through one of those providers. Additionally, we secured our application using authorization policies that have no specific dependency on ACS or partner/customer-specific policies or protocols. We can add as many new providers as we want without needing to make changes to the application.

Finally, remember that the protocol is REST-based and so can be easily used from multiple platforms. The core functionality of ACS is to map input claims to output claims by abstracting multiple input claims from multiple sources to a consistent set of output claims expected by the relying party. The relying party doesn't have knowledge of the input claim source; it trusts the output claims issued by ACS. The ACS management service API provides functions to create these mappings.

Summary

Microsoft is investing heavily in its products to support a claims-based identity model. The Windows Identity Foundation SDK, ADFS v2.0, WCF, and ACS are good evidence of the direction Microsoft is taking. In cloud applications, currently there is no unified programming model for managing authentication and authorization across multiple applications and platforms. Enterprises rely on identity federation services like ADFS, and consumer applications build custom identity providers within the service. ACS fills this gap by abstracting the claims-transformation logic in the cloud and presenting a unified view of claims issued by identity providers to the claims required by applications.

In this chapter, you learned how ACS achieves this through simple configurations of input and output claims. You also examined different scenarios that ACS supports. Through examples, you gained hands-on knowledge about implementing claims-based identity models using ACS. ACS is a core piece of the overall Azure Services Platform and is actively used in other Azure technologies like the AppFabric Service Bus. In the next chapter, you learn about the communication and messaging possibilities offered by the AppFabric Service Bus in the cloud.

Concepts and Terminology

Before diving deep into a technical discussion, you should understand some key concepts and terminology that are extremely important in the design and architecture of ACS. This section introduces some new terms and redefines some existing terms in the ACS context.

Identity Provider

An identity provider manages your identity and provides an authentication service for client applications. Identity providers authenticate users and issue Security Assertions Markup Language (SAML) tokens (defined in a moment). SAML tokens contain user IDs and other identity properties of the user (claims). Examples of some identity providers are Windows Live ID, ADFS 2.0, Google Accounts, Yahoo ID, and Oracle.

Relying Party

The relying party is the application that validates the claims issued by ACS to authorize a user and release appropriate access to the user. As a developer, you're primarily concerned with developing a relying party application that receives a SAML token filled with claims from ACS. You can then process these claims in the relying party to provide appropriate access to the end user.

Security Token (SAML Token)²

A SAML token is an XML message consisting of sets of claims digitally signed by the issuing authority. The token is issued by a Secure Token Service (STS). The ACS and relying party both process claims from SAML tokens.

Secure Token Service (STS)

An STS is a subsystem responsible for building, signing, validating, cancelling, renewing, and issuing SAML tokens. An STS may support one or more of these features. It typically implements the protocol defined in the WS-Trust specification. Identity providers and ACS both have STS capabilities.

An R-STS is a resource STS that acts as an intermediate claims-transformation service to transform input claims from a partner STS to output claims specific to your application. This model is popular in extranet and cross-enterprise applications. ACS is an R-STS because it transforms input claims from identity providers to output claims. You can build your own STS and R-STS using Windows Identity Framework.

Request for Security Token (RST)

Every relying party requires a unique set of claims it can process. RST is the request made to an STS to acquire these claims to an STS. For example, a requestor may make this request to ACS to acquire claims for a relying party.

Request Security Token Response (RSTR)

The response sent by an STS to the RST is called an RSTR. This request contains the SAML token with claims signed by the STS.

Claim

A claim consists of information about the user or role interested in accessing an application (or relying party). A claim can have any information about the user depending on the configuration of the identity provider and ACS. A typical ACS scenario involves three kinds of claims:

- **User claims:** When a user sends a request for a security token to ACS, the request contains claims like the username, password, domain name, and so on, which are usually required for authentication.
- **Input claims:** When the user is authenticated with the identity provider, the identity provider issues a SAML token. The SAML token usually contains input claims to the ACS. These input claims may contain user claims as well as additional claims introduced by the identity provider in the SAML token.

² You can find the SAML token profile at the WS-I web site: www.ws-i.org/deliverables/workinggroup.aspx?wg=samltoken.

- **Output claims:** ACS examines the input claims from the SAML token issued by the identity provider and maps them to output claims. ACS translates input claims into application- (or relying party-) specific output claims and includes them in the token that the relying party can use to authorize users and give appropriate access. For example, an input claim “Username: tejaswi_redkar” may map to an output claim “Role: Domain User.” The relying party reads the Role as Domain User and provides Domain User privileges to the user. Input claims are mapped to output claims as a part of ACS configuration exercise covered later.

Identity Federation

Identity federation is a set of mechanisms, standards, and patterns that define different ways of sharing identity information between domains. It reduces identity-maintenance costs and also simplifies software development because you don’t have to design and maintain a separate identity store within the application. Federated identities also ease single sign-on between applications running in different domains and/or enterprises.

Windows Identity Foundation (WIF)

The Windows Identity Foundation is a Microsoft product used to create claims-based applications and services in .NET Framework. You can build your own STS using the Windows Identity Framework if existing products don’t fulfill your application requirements. WIF simplifies the development of cross-domain security scenarios. It provides a framework for building passive (web browser-based) as well as active (Windows Communications Foundation) clients that support identity federation across a wide variety of clients and servers. ACS uses WIF to provide STS capabilities. ADFS 2.0, Microsoft’s next-generation claims-based identity federation server, is also built using WIF.

Active Directory Federation Server (ADFS 2.0)

ADFS 2.0 is a Microsoft product that provides STS functionality to Active Directory. It’s the next version of Active Directory Federation Services (ADFS) and supports a claims-based identity model. It enables the creation of single sign-on between on-premises and cloud applications using the claims-based identity model. By definition, ADFS 2.0 implements the protocol defined in the WS-Trust specification and so provides interoperability with other products like Sun OpenSSO and Novell Access Manager. ADFS 2.0 supports not only passive clients like web browsers but also active stand-alone clients built using the Windows Communications Foundation (WCF).

Web Resource Authorization Protocol (WRAP) and Simple Web Token (SWT)

Version 1 of ACS implements the REST-friendly Web Resource Authorization Protocol (WRAP) that defines the Simple Web Token standard. The token issued by ACS adheres to the SWT specification, which you can find in the WRAP profiles on the OAuth web site at <http://groups.google.com/group/oauth-wrap-wg>. SWT tokens are HTTP form encoded key-value pairs signed with an HMAC-SHA256 cryptographic key. ACS always emits either an SWT or SAML for different types of input tokens (such as SAML, SWT, Facebook tokens, LiveID tokens, and so on), so the relying party can always expect either SWT of SAML from ACS. SWT is typically designed for REST-based web services. SAML is supported by a wide range of software vendors like IBM, Microsoft, Oracle, and

Computer Associates. You can find more information on SAML 1.1 and SAML 2.0 here http://en.wikipedia.org/wiki/SAML_2.0.

Bibliography

Federated Identity Primer. (n.d.). Retrieved from sourceid.org: www.sourceid.org/content/primer.cfm.

Microsoft Corporation. (2009, 11 17). *Identity Developer Training Kit (PDC 2009)*. Retrieved from Microsoft Download Center: www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=c3e315fa-94e2-4027-99cb-904369f177c0.

Microsoft Corporation. (n.d.). *Identity Management (Geneva Framework)*. Retrieved from MSDN: <http://msdn.microsoft.com/en-us/security/aa570351.aspx>

Microsoft Corporation. (n.d.). *MSDN .NET Services Center*. Retrieved from MSDN: <http://msdn.microsoft.com/en-us/azure/netservices.aspx>

Microsoft Corporation. (2009, 11 16). *Windows Identity Foundation*. Retrieved from Microsoft Download Center: www.microsoft.com/downloads/details.aspx?familyid=EB9C345F-E830-40B7-A5FE-AE7A864C4D76&displaylang=en

Microsoft Corporation. (2009, 11 16). *Windows Identity Foundation SDK*. Retrieved from Microsoft Download Center: www.microsoft.com/downloads/details.aspx?familyid=C148B2DF-C7AF-46BB-9162-2C9422208504&displaylang=en

Smith, J. (2009, November 14). *ACS SAML / ADFS v2 Sample*. Retrieved from Justin Smith's Blog: <http://blogs.msdn.com/justinjsmith/default.aspx>

Microsoft Corporation (n.d) *Windows Azure Platform Training Kit*. Retrieved from Microsoft Download Center: <http://msdn.microsoft.com/en-us/wazplatformtrainingcourse.aspx>

AppFabric Service Bus

In this chapter, you will learn details of the AppFabric Service Bus architecture. We will cover the concept of an Enterprise Service Bus, and introduce you to the AppFabric Service Bus. We will then cover the various ways of programming applications that use the Service Bus, both from the .NET Client API and a REST-based API. We will also look at the newest functionality addition, a new robust messaging system consisting of Queues and Topics. After reading this chapter, you should be able to use the AppFabric Service Bus in your own architectures.

■ **Note** This is a large chapter. If you're already familiar with the concepts of an Enterprise Service Bus, you might want to skip ahead to "Introduction To the AppFabric Service Bus" section.

First, a Little Background. . .

Over the past decade, enterprises have invested heavily in upgrading their enterprise architecture by implementing several enterprise software patterns like Service Oriented Architecture and Enterprise Service Bus (ESB). These software patterns make application infrastructure loosely coupled and compatible across software boundaries. For example, Microsoft SharePoint server can integrate with Lotus Domino or EMC Documentum. You can also build custom business applications that can take advantage of these loosely coupled architectures. To make such integrations possible, Microsoft has defined four tenets¹ as guidance:

- Services have explicit boundaries.
- Services are autonomous and deployed, versioned, and managed independently.
- Services share schema and contracts.
- Service compatibility is achieved by appropriate policy configuration.

These tenets are by no means comprehensive, but they give a good high-level framework for service-oriented enterprise architectures.

¹ John Evdemon. *The Four Tenets of Service Orientation*. Business Architectures and Standards, Microsoft Architecture Strategy Team, Thursday, May 19, 2005.

The ESB pattern is designed to offer service-oriented brokered communications of enterprise objects across enterprise applications. The design and implementations of ESBs varies in different organizations because by definition, ESB is a pattern and not a product. For example, I consulted with an enterprise where the ESB had an FTP interface. You could configure and schedule the ESB on the kind of data the subscriber systems needed from a publisher system. The ESB then queried the publisher system and provided an FTP endpoint to the subscriber systems. The architecture worked like a charm because the contract at the data level was defined in a set of enterprise schema, and the data communication medium was FTP, a well-known public protocol.

Even though these architectures work well in an enterprise environment, they can't easily cross enterprise boundaries and aren't designed for Internet scale. As applications move into the cloud, they still need to decouple themselves to keep the architectural tenets of the enterprise intact and make applications seamlessly accessible not only in the cloud but also on-premises.

Microsoft's attempt to create an Internet-scale Service Bus is an Azure Platform Service called AppFabric Service Bus. AppFabric Service Bus runs in the cloud and seamlessly connects cloud, enterprise, and consumer applications.

Enterprise Service Bus (ESB)

There is no generic architecture for an ESB, because it's a pattern and can be built as an add-on for already-existing Microsoft products like BizTalk Server, MSMQ, Windows Communications Foundation (WCF), and SQL Server. Every company that makes a product conforming to the ESB pattern has a different definition of ESB. I define the ESB pattern as follows: "ESB is an enterprise architecture pattern that defines the connectivity, contracts, and communication of business objects across enterprise applications."

The definition is depicted in Figure 8-1.

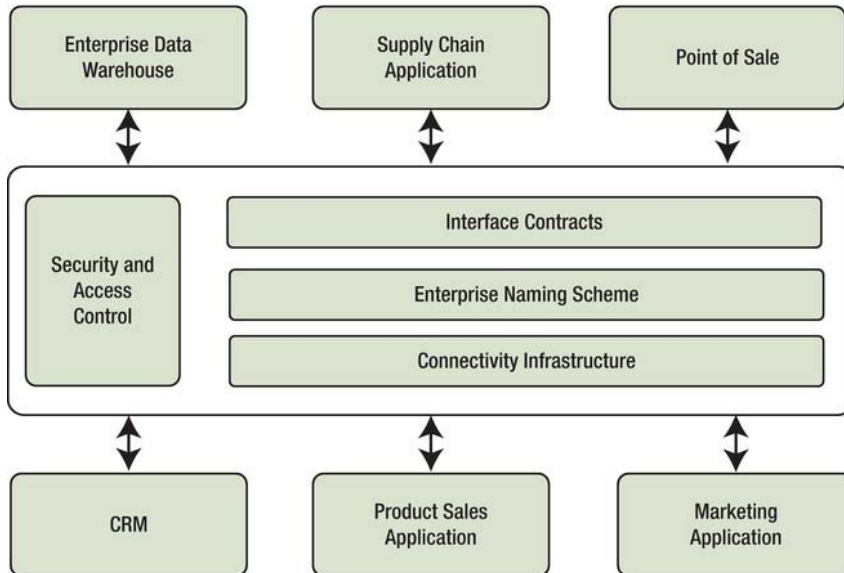


Figure 8-1. Enterprise Service Bus pattern

As in my definition, an ESB offers the following four core services:

- Security and Access Control
- Connectivity Infrastructure
- Enterprise Naming Scheme
- Interface Contracts

These are discussed in the sections that follow.

Security and Access Control

The Security and Access Control service offers communication as well as message-level security for interacting with ESB endpoints. An ESB usually integrates with the enterprise identity providers but may have an integrated identity provider. All applications must pass through this layer before interacting with the ESB.

Connectivity Infrastructure

The connectivity infrastructure defines the mechanisms and endpoints of an ESB to communicate business objects across enterprise applications. These endpoints may be any public or private protocols conforming to enterprise standards. In enterprises, I have seen ESBs with a connectivity infrastructure based on protocols like FTP, HTTP, TCP-Sockets, SOAP, and even REST.

Enterprise Naming Scheme

To communicate business objects across enterprise applications, you need an enterprise standard for defining naming schemes for objects. For example, a Product object must have a single schema across the enterprise. The URI scheme for accessing these objects in an ESB should also be standardized.

ESB can define the URI scheme for accessing business objects. For example, the URI of a specific product object may be of the format `/MyEnterprise/MyProducts/T-Shirts["ProductId"]`. ESB can translate this schema and make it usable across any connectivity infrastructure. For example, in an HTTP-based interface, you can access the product using the URI `http://mysystem/MyEnterprise/MyProducts/T-Shirts["ProductId"]`, whereas in an FTP-based interface, you can access the serialized object in a file `/MyEnterprise/MyProducts/T-Shirts/["ProductId"].xml`. Enterprise schemes not only define a uniformed way of accessing business object, but also offer simple business rules and filters within the scheme.

Interface Contracts

ESB acts as a broker of business objects across business applications. One business application can access the methods and objects of another business application in a loosely coupled manner. The ESB interface contracts define the standard contracts for invoking methods on the ESB as well as other business systems. For example, I have a marketing reporting application that needs access to daily sales data on a periodic basis, but sometimes I also want to know real-time sales figures by accessing real-time sales data on demand. ESB can define interface contracts that the source and destination systems can adhere to while making asynchronous and synchronous invocations.

Evolution of the Internet Service Bus (ISB)

ESB clearly has challenges in the cloud as well as in cross-organization scenarios. Current ESBs aren't designed to offer the scalability and availability required by cloud applications. In cross-organization scenarios, ESB may somehow integrate the connectivity infrastructure and interface contracts, but it faces significant challenges in integrating security and enterprise naming schemes. Porting enterprise schemes becomes difficult across enterprises, and most applications need to be rewritten to work with different enterprise schemes. To make the security service in ESB work across organizations, ESB needs to integrate with the security provider of another enterprise. ESBs aren't designed to work across security realms and thus usually aren't recommended to be used across enterprises. With the enterprise push toward cloud services, it's important to offer a Service Bus in the cloud that can be used by enterprises as well as consumer applications at an Internet scale.

Some years back, I designed an Internet Service Bus (ISB) specifically to collect data from energy devices in homes and commercial buildings. At that time, I called it Energy Bus, but essentially it was an ISB with some limitations. I deployed this ISB as part of an overall service in a data center. The service was designed for high scalability and availability with multiple clustered nodes at the infrastructure as well as database level. The business purpose of the service was to collect energy data from thousands of homes and commercial buildings and offer energy-management services to end users through utility companies. For example, you as a homeowner could control your home devices like lighting, security, HVAC, and coffee maker over the Internet. At the same time, devices in the house could call the energy service in the cloud to send energy usage logs (kWh values) and alarms (fire alarm, burglar alarm, and so on). The entire architecture was built around the concept of an Internet Service Bus with Microsoft Message Queuing (MSMQ) as its backbone communications engine. Figure 8-2 illustrates the high-level architecture of the ISB.

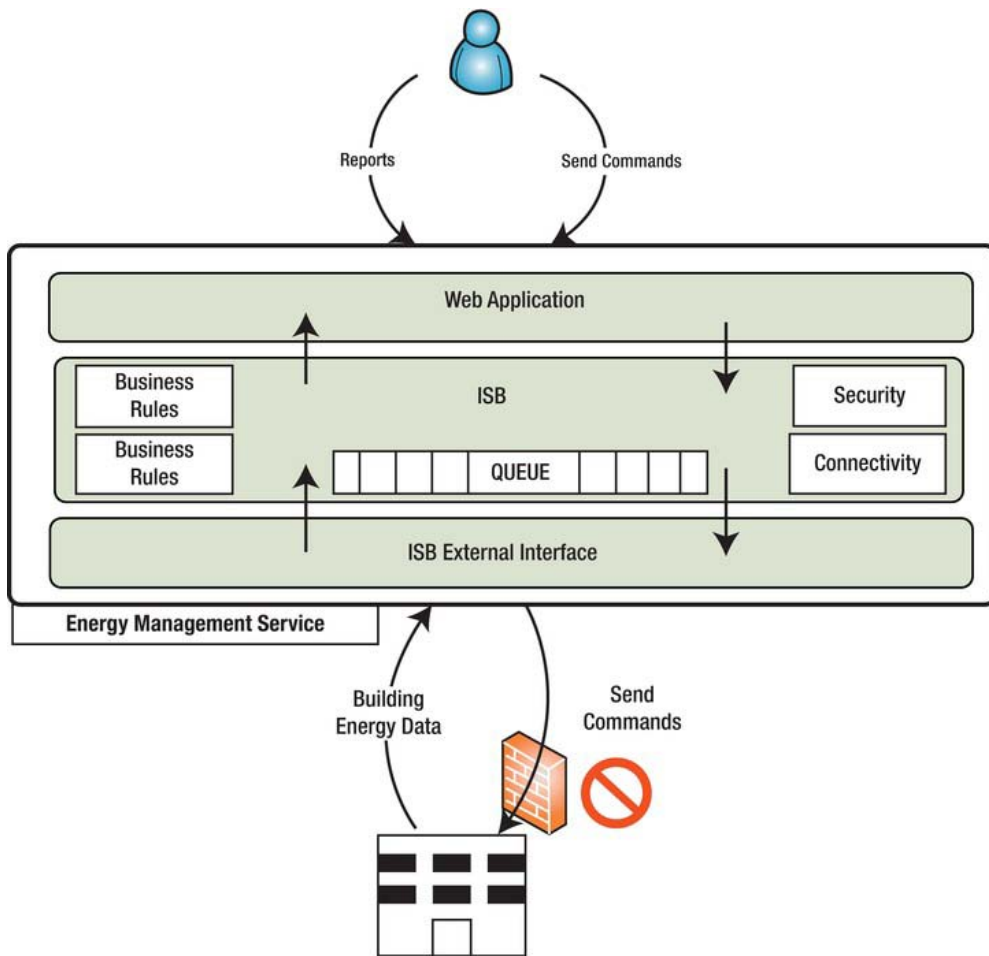


Figure 8-2. Energy management service ISB

As shown in Figure 8-2, end users could generate reports on their energy data and also get and set values of energy-consuming devices in buildings and apartments. The ISB provided the connectivity and interfaces between the devices in the buildings and cloud. Two of the biggest challenges I faced in designing the service were as follows:

- *Connectivity:* Because of the nature of the service, one of its core functions was providing real-time connectivity between devices and the service. Most often, devices were behind firewalls or network address translation (NAT) routers. Even though communication from the device to the service was seamless, communication from the service to the device was always challenging. Opening firewall ports to the devices wasn't an option in many cases due to customers' security policies. So, ISB communication couldn't penetrate the firewall, and communications failed. As a workaround, the developers had to tunnel communications through only the ports that were allowed through the firewall, or build a proxy server on the customer site that polled the cloud service on a periodic basis to receive commands from ISB.
- *User profiles:* Customers wanted their existing user profile stores to synchronize with the cloud securely, rather than creating all the user profiles from scratch. As a workaround, I ended up building a profile import and synchronization server that periodically synchronized the user profiles from the customer's Active Directory with the service database in the cloud. Because the service was deployed in the cloud and was available for multiple customers, it couldn't directly integrate with any identity providers.

If Microsoft's AppFabric Service Bus had been available at the time, both these challenges would have been non-existent because the Service Bus is designed to address these exact challenges. The AppFabric Service Bus provides access control, naming, service registry, messaging, and connectivity services at Internet scale. It enables bidirectional communications between on-premises and cloud application through relay service capabilities. The relay service runs in the cloud, and interested parties register themselves with it to communicate with each other. The Service Bus determines the best connectivity method by either using outbound bidirectional sockets connections from the service to the Service Bus when a firewall is present, or establishing a direct connection between the client and the service when there is no firewall.

Some of the applications you use today may already support bidirectional communication through NAT traversal. Internet client applications like Windows Live Messenger, Kazaa, BitTorrent, Xbox Live, and some Universal Plug and Play clients (UPnP) can traverse through firewalls using Relay Service.

Relay Service

A *relay service* is a central service running in the cloud that provides a rendezvous connection point between the client and the service. In networking terms, the *rendezvous address* is a common meeting point for two connections. Figure 8-3 shows typical relay service communications between client and service.

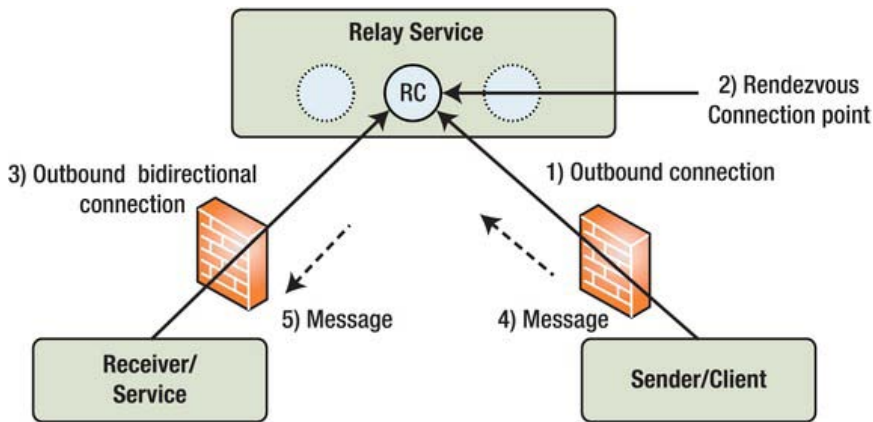


Figure 8-3. Relay service

As shown in Figure 8-3, the relay service runs in the cloud and offers connection endpoints to the message client and service. A client opens an outbound connection to the relay service. The service opens a bidirectional outbound connection to the relay service and receives a rendezvous connection endpoint that is shared with the service. The outbound bidirectional connection from the service makes it possible for the service to receive messages on an outbound connection without opening inbound ports in the firewall or NAT routers. The client sends a message to the relay service that is routed by the rendezvous connection point to the service over the outbound connection. Thus, the relay service makes it possible for clients and services to communicate through firewalls.

With the advancements in networking APIs in frameworks like the .NET Framework, it isn't difficult to build a relay service and bidirectional sockets in your applications. The real challenge is to build a relay service at Internet scale for applications around the world. In this chapter, you see how the AppFabric Service Bus provides an Internet scale Service Bus with relay capabilities.

Introduction to the AppFabric Service Bus

Microsoft's AppFabric Service Bus is an Internet-scale Service Bus that offers scalable and highly available connection points for application communication. The AppFabric Service Bus is designed to provide connectivity, queuing, and routing capabilities not only for the cloud applications but also for on-premises applications. It also integrates with the Access Control Service (ACS) to provide secure relay and communications. Figure 8-4 illustrates the architecture of the AppFabric Service Bus.

■ **Note** To see a Field note describing how Service Bus was used to connect an Azure application to a FAST search engine on-premises, go to <http://www.microsoft.com/windowsazure/learn/real-world-guidance/field-notes/integrating-with-service-bus-and-port-bridge/>.

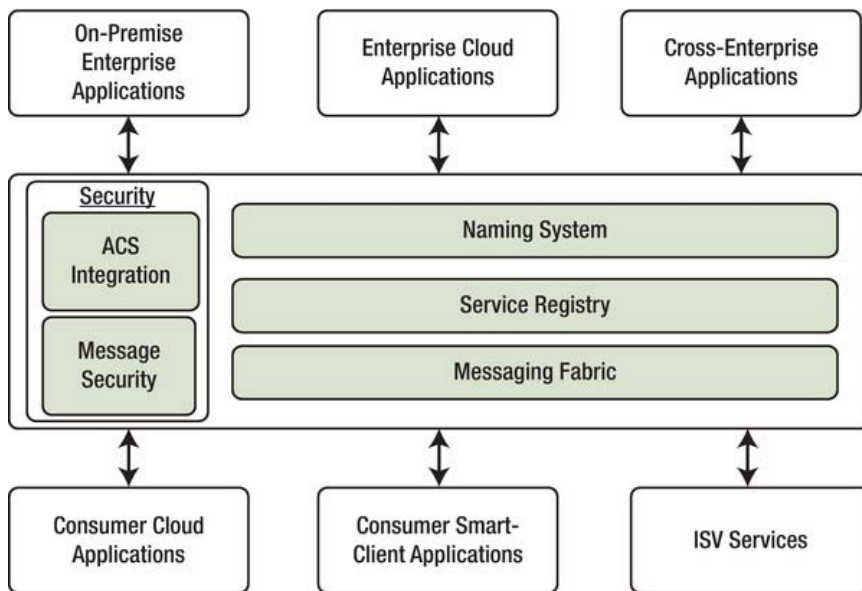


Figure 8-4. AppFabric Service Bus architecture

As shown in Figure 8-4, the AppFabric Service Bus consists of four main services that can be used by different kinds of on-premises as well as cloud services. They are as follows:

- Security
- Naming service
- Service registry
- Messaging fabric

Security

As you read in Chapter 1, one of the biggest concerns of enterprises in moving applications to the cloud is security. At Internet scale, where millions of frauds and hacks occur on a daily basis, secure communication across applications is absolutely necessary for enterprises. An on-premises environment is governed and controlled by corporate policies, and prevention is preferred to cure. In the cloud, systems, applications, and data are exposed and prone to not only external but also internal threats. To overcome this barrier, the AppFabric Service Bus offers the following two main options for securing the transport of messages from clients to services:

- Access Control Service (ACS) integration
- End-to-end security

ACS Integration (Relay Authentication)

Microsoft has integrated the AppFabric Service Bus with ACS to provide relay authentication and authorization. The message sender and message receiver have to pass security checks before connecting to the AppFabric Service Bus. Services (or receivers) must be authenticated either by ACS or an identity provider trusted by ACS before establishing a connection to the AppFabric Service Bus. By default, the clients (or senders) require relay authentication but can be optionally exempted from authentication by services. The client authentication type may be different than the service authentication type. For example, a client can authenticate using a shared secret, whereas a service can authenticate using a SAML token. Three types of authentication are currently available with ACS: shared secret, SAML token, and simple web tokens (SWTs). Figure 8-5 illustrates the Service Bus integration with ACS.

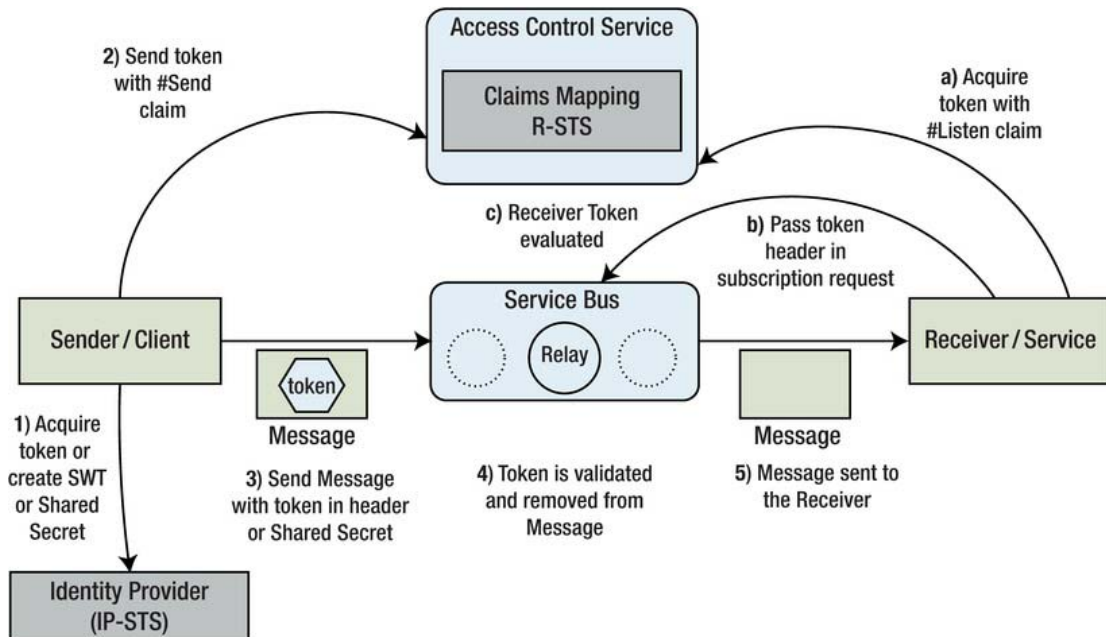


Figure 8-5. AppFabric Service Bus and ACS integration

As shown in Figure 8-5, the client and service both have to be authenticated with ACS before connecting to the Service Bus. The authentication for client and service takes place separately and isn't dependent on the other. The client authentication process is as follows:

1. The client acquires a SAML token from a SAML token provider or creates an SWT token or uses a shared secret to authenticate with Service Bus.
2. The client sends an authentication request to ACS and acquires a #Send claim from ACS. After it is authenticated, the client receives a token containing the #Send claim. AppFabric Service Bus is preconfigured to validate only the #Send claim from a client application.

■ **Note** For more information about ACS, please refer to Chapter 7.

3. The token with the #Send claim is embedded into the header of the message sent to the Service Bus relay service.
4. The relay service validates the token and removes it from the message header. Because AppFabric Service Bus is the relying party in this scenario, as seen in the previous chapter, ACS encrypts the token with a public key, and Service Bus decrypts the token with a private key. During solution provisioning, trust between ACS solution and Service Bus is already established by the AppFabric portal.
5. The relay service sends the message (without the token) to the service.

The service also has to authenticate itself with ACS before connecting to the AppFabric Service Bus. The service authentication process is as follows:

6. The service sends an authentication request to ACS and acquires the #Listen claim from ACS. Similar to the client, the service can authenticate with any identity provider trusted by ACS.
7. The token with the #Listen claim is embedded in the subscription request to the AppFabric Service Bus relay service.
8. The relay service validates the token and lets the service open a bidirectional outbound connection to the relay service.

Optionally, you can turn off the client authentication by specifying it in the service-binding configuration as shown in Listing 8-1.

Listing 8-1. Turning Off Client Authentication

```
<binding name="default">
  <security relayClientAuthenticationType="None" />
</binding>
```

The `RelayClientAuthenticationType.None` value specifies that clients of the service aren't required to present any token issued by the ACS. Usually, you set the `RelayClientAuthenticationType.None` value if you want the service to authenticate and authorize the clients and the AppFabric Service Bus authentication is adding unnecessary overhead to the service without adding any value. The default value for the `relayAuthenticationType` attribute is `RelayAccessToken`.

`TransportClientEndpointBehavior` is a class in the `Microsoft.ServiceBus` namespace that describes the WCF behavior of a particular endpoint registered with the Service Bus. The `CredentialType` property of the `TransportClientEndpointBehavior` class specifies the type of authentication you use for the endpoint. AppFabric Service Bus API offers `TransportClientCredentialType` enumeration with four different values for relay authentication, as shown in Table 8-1.

Table 8-1. *TransportClientCredentialType Values*

TransportClientCredentialType option	
Saml	Suggests that the client is authenticated using a Security Assertions Markup Language (SAML) tokens. The SAML token is sent over the SSL protocol, and you're required to create your own SSL credential server.
SharedSecret	Refers to the issuer and issuer key created in ACS from the ACS management portal. The Service Bus has a dedicated issuer name and issuer key created by default when you create a service namespace in ACS.
SimpleWebToken	Suggests that the client is authenticated using an SWT token that is self-issued by the client and registered with ACS.
Unauthenticated	Doesn't require clients to authenticate to connect to the Service Bus. Client must set this option explicitly in code if it's exempted from authentication to the service. When this option is used, the AppFabric Service Bus sends the message without acquiring a token from ACS.

The services and clients can choose to authenticate using any of the configured types. In the examples later in the chapter, I show you how to implement these options in your code.

As you read in the previous chapter on ACS, ACS creates dedicated Service Bus endpoints in your service namespace. Figure 8-6 shows the Service Bus section from the service namespace page of your account.

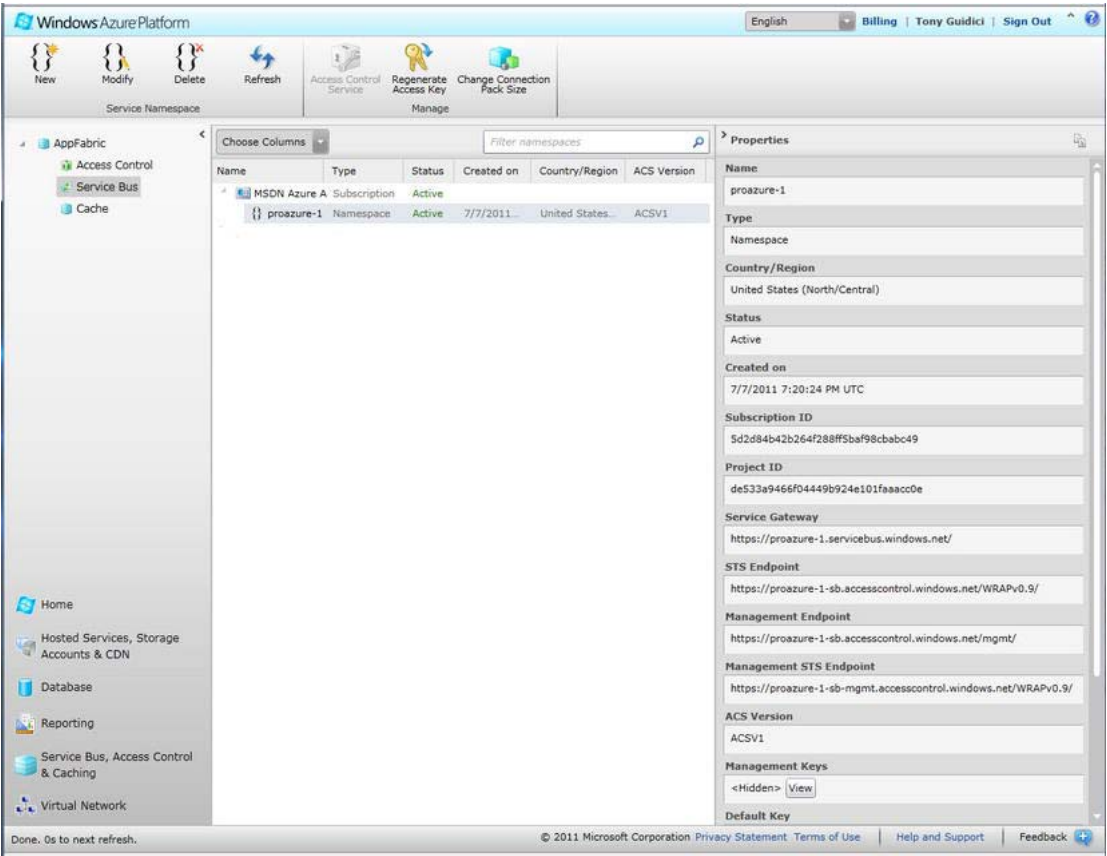


Figure 8-6. Service Bus solution in ACS

You can map incoming and outgoing claims in ACS to authenticate your clients and/or services. Thus, ACS integration provides The AppFabric Service Bus with the ability to authenticate with any identity provider and participate in a claims-based identity model for authorization.

Message Security

Relay authentication is geared toward authenticating clients and services to communicate with the AppFabric Service Bus. But a true enterprise solution is incomplete without security of the message that travels between the communicating parties. Message security refers to the security of the message that travels from the source through the AppFabric Service Bus to the destination. The AppFabric Service Bus offers four options for securing messages between the clients and services. The enumeration `Microsoft.ServiceBus.EndToEndSecurityMode` in the AppFabric Service Bus API defines four security modes, as shown in Table 8-2.

Table 8-2. Message Security Values

Message Security Type	Description
None	Security for the message is disabled. The message is sent as-is from the client to the service.
Transport	The message is sent through a secure channel (such as HTTPS) to and from the relay service. The movement of the message within the AppFabric Service Bus isn't secure. The message doesn't contain any client credentials. This is the recommended and the default mode for most applications where messages don't contain sensitive information.
Message	In this security type, you can encrypt the body of the message using an X.509 certificate provided by your service. Because the message is encrypted, the movement of the message within the .NET Service Bus is secure. The message may contain client credentials, and the service must authenticate the client credentials if present in the message. Use this option only if you need client credentials in your service for authorization purposes.
TransportWithMessageCredential	This security type is a combination of the Transport and Message security types. The transport between the relay service and applications is secured using a secure channel, and the message is moved from the client all the way to the service in encrypted format. The message is secure as it travels through the AppFabric Service Bus. The message may also contain client credentials. This security type is recommended only when sending sensitive messages over the Internet.

■ **Note** Message security is independent of relay security. Relay security is used to connect with the AppFabric Service Bus, whereas message security refers to the security of the message that traverses through the AppFabric Service Bus.

Naming Service

The Naming service allows you to assign DNS-capable names to your service, which makes the service easily resolvable over the Internet. The Internet is based on the Domain Name System (DNS) where every resource on the Internet can be resolved using the DNS name and relative path. For example, in the URL `www.microsoft.com`, `microsoft.com` is the registered domain name for Microsoft's web site. HTTP is the protocol used for accessing the web site. Similarly, `http://msdn.microsoft.com` is the registered domain name for MSDN site. The `msdn` part of the URL is called a subdomain of

microsoft.com, and microsoft.com itself is called a root domain. DNS follows a hierarchical structure where one root domain can consist of many subdomains to form a tree structure. For example, social.msdn.microsoft.com adds one more level (social) under msdn to the microsoft.com domain hierarchy.

The Internet DNS system was designed for reference to static resources like web pages and web sites where the application may change but the domain name remains the same. In the cloud services world, there can be multiple unique cloud services and subservices that can register and unregister themselves from the DNS depending on the cloud service requirements. Companies can use the AppFabric Service Bus on-premises as well as off-premises. In case of on-premises services, companies can register unique domain names for services; but for off-premises services, companies must invest in infrastructure and internal naming schemes for identifying these services uniquely on the Internet.

The AppFabric Service Bus offers a DNS-compatible naming system for assigning unique Internet URIs to cloud as well as on-premises services. The AppFabric Service Bus defines a root domain name that can be resolved through the Internet DNS, but offers a service namespace-based naming hierarchy below the root. For example, in the Service Bus naming system, servicebus.windows.net is the root domain of the Service Bus. If you have ten service namespaces you want to register with the Service Bus, all ten service namespaces automatically receive URIs for cloud as well as on-premises services. If you name your namespaces solution1, solution2, ..., solution10, then each solution has its own URI name:

solution1.servicebus.windows.net

solution2.servicebus.windows.net

....

solution10.servicebus.windows.net

Figure 8-7 shows an example hierarchical naming tree structure in the AppFabric Service Bus.

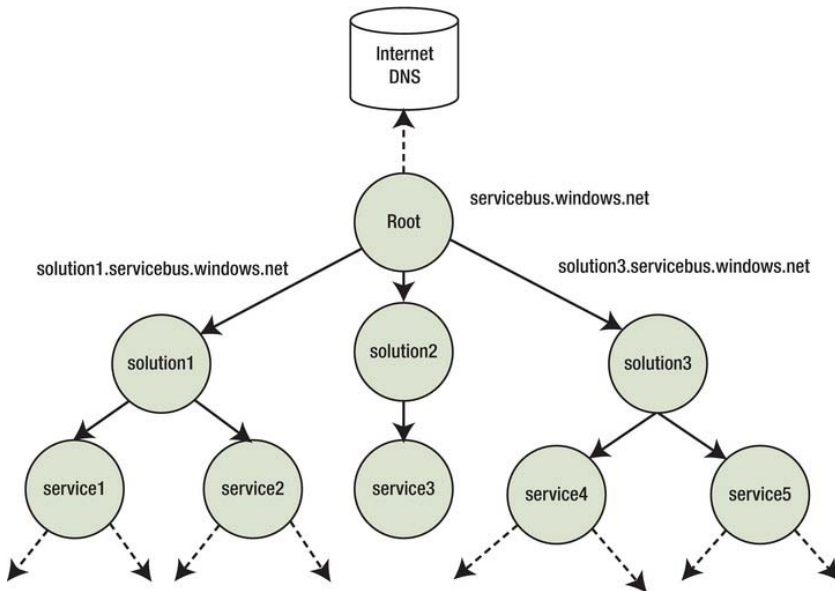


Figure 8-7. Hierarchical naming structure

You, the service namespace owner, have complete control over the naming hierarchy under the Service Bus root node. The naming scheme for the URI formation is

```
[scheme]://[solution-name].servicebus.windows.net/[name]/[name]/...
```

where [scheme] is the protocol for accessing the service. AppFabric Service Bus supports two URI schemes: http and sb. http is used for all HTTP-based communications between clients and services, whereas sb is used for all TCP-based communications between clients and services. [solution-name] is the unique solution name across the entire AppFabric Service Bus namespace. Because this name is the subdomain under the AppFabric Service Bus root domain, this needs to be unique across the entire AppFabric Service Bus namespace. You can choose any solution name while creating the account. For example, the solution I use in this chapter is the ProAzure solution. The name ProAzure is unique across the entire AppFabric Service Bus namespace. You can reference the ProAzure namespace in AppFabric Service Bus as `http://proazure.servicebus.windows.net` or `sb://proazure.servicebus.windows.net`.

[name] is the user-defined virtual name for a service or a hierarchical structure pointing to a service. You can create any hierarchical structure using the user-defined namespace. For example, if you're offering an energy management service in different cities around the world, and you have deployed different instances of your service, you can assign unique names to these service instances based on the names of the cities as follows:

```
http://proazure.servicebus.windows.net/sanfrancisco/energy
```

```
http://proazure.servicebus.windows.net/newyork/energy
```

```
http://proazure.servicebus.windows.net/london/energy
```

```
http://proazure.servicebus.windows.net/singapore/energy
```

```
http://proazure.servicebus.windows.net/mumbai/energy
```

You can also further extend the hierarchy by offering subservices like

```
http://proazure.servicebus.windows.net/sanfrancisco/energy/reports
```

```
http://proazure.servicebus.windows.net/sanfrancisco/energy/realtime
```

```
http://proazure.servicebus.windows.net/sanfrancisco/energy/logs
```

All these URIs point to endpoints of services hosted in these cities. The physical location of these URIs is transparent not only to applications but also to each other. The

`http://proazure.servicebus.windows.net/sanfrancisco/energy/reports` service may be hosted in a totally separate location from the

`http://proazure.servicebus.windows.net/sanfrancisco/energy/realtime` service. The AppFabric Service Bus internally resolves the actual location of the service endpoints at runtime. Thus, the AppFabric Service Bus allows you to create an infinitely deep hierarchical naming structure referencing endpoints of cloud as well as on-premises services. It also abstracts the DNS registration and resolution for your services and applications calling these services.

Service Registry

The AppFabric Service Bus provides a registration and discovery service for service endpoints called the service registry. The service endpoints can be in the cloud or on-premises. The service registry offers an Atom feed to your solution. You can register a service endpoint into the Atom Feed using either the Atom

Publishing Protocol (APP)² or WS-Transfer³ references. APP and WS-Transfer both support publishing, listing, and removing the service endpoints. The client application can then discover your service endpoint references by simply navigating Atom 1.0 feed of your solution. The Atom 1.0 feed exposes a tree-like structure you can manually or programmatically navigate to get to the leaf node of the service endpoint. You can also programmatically register a service endpoint for public discovery by setting the `DiscoveryMode` property of the `Microsoft.ServiceBus.ServiceRegistrySettings` object to `Public` and associating it with the service endpoint behavior as shown in Listing 8-2. In this approach, the AppFabric Service Bus relay service automatically registers the service endpoint for you in the service registry.

Listing 8-2. *Associating ServiceRegistrySettings*

```
class Program
{
    static void Main(string[] args)
    {
        ServiceHost host = new ServiceHost(typeof(EnergyManagementService));
        ServiceRegistrySettings settings = new ServiceRegistrySettings();
        settings.DiscoveryMode = DiscoveryType.Public;
        foreach (ServiceEndpoint s in host.Description.Endpoints)
            s.Behaviors.Add(settings);
        host.Open();
        Console.WriteLine("Press [Enter] to exit");
        Console.ReadLine();
        host.Close();
    }
}
```

The default setting for the public discovery is set to private, so if you don't set the discovery type to public, your service won't be discoverable publicly. After you register the service endpoint, you can view the Atom feed of your Service Bus registry by navigating to the AppFabric section of the Azure management portal, and clicking the Service Bus item under AppFabric in the navigation tree. You'll see the endpoint information on the right, as shown in Figure 8-8.

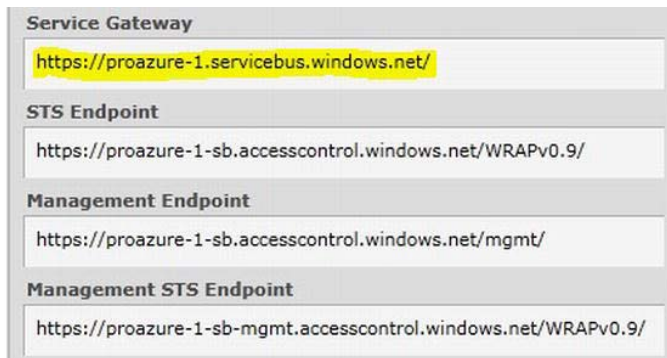


Figure 8-8. *Service Bus registry link*

² Atom Publishing Protocol Reference: www.ietf.org/rfc/rfc5023.txt.

³ WS-Transfer Specification: www.w3.org/Submission/WS-Transfer/.

Figure 8-9 shows the Atom feed for the publicly listed services in the ProAzure solution.



Figure 8-9. Service Bus registry for the ProAzure solution

The Service Bus registry shows only one registered service. I revisit the Service Bus Registry later in the examples in this chapter.

Messaging Fabric

The messaging fabric enables the relaying and communication of messages between clients and services. The messaging fabric makes it possible to expose your service endpoints into the cloud for on-premises as well as cloud deployed services. The messaging fabric also integrates with ACS to provide message level security.

The relay service is the core component of the AppFabric Service Bus messaging fabric. The relay service makes it possible for the client and services to communicate behind firewalls and NAT routers. As the name suggests, the relay service plays the role of relaying messages from clients to the services by assuming the responsibility of receiving the messages from the clients and delivering it to the services. The services can be running in the cloud or on-premise. As long as the endpoints of the services are registered in the service registry of the AppFabric Service Bus and are reachable, the relay service forwards the message. In simple terms, the relay service is like a postman who delivers the message from the client to the service. As long as the services address is valid and in the USPS registry, the postman delivers the mail. The only difference is that the postman is an asynchronous communication whereas the relay service defines a synchronous communication. This means the relay service requires the server to be available in most of the cases when the client sends a message.

The relay service supports the following types of communications between the clients and the services:

One-way communications

- Publish/Subscribe messaging
- Peer-to-peer communications
- Multicast messaging
- Direct connections between clients and services

Figure 8-10 illustrates the communication process that takes place between the client, the service, and the AppFabric Service Bus relay service.

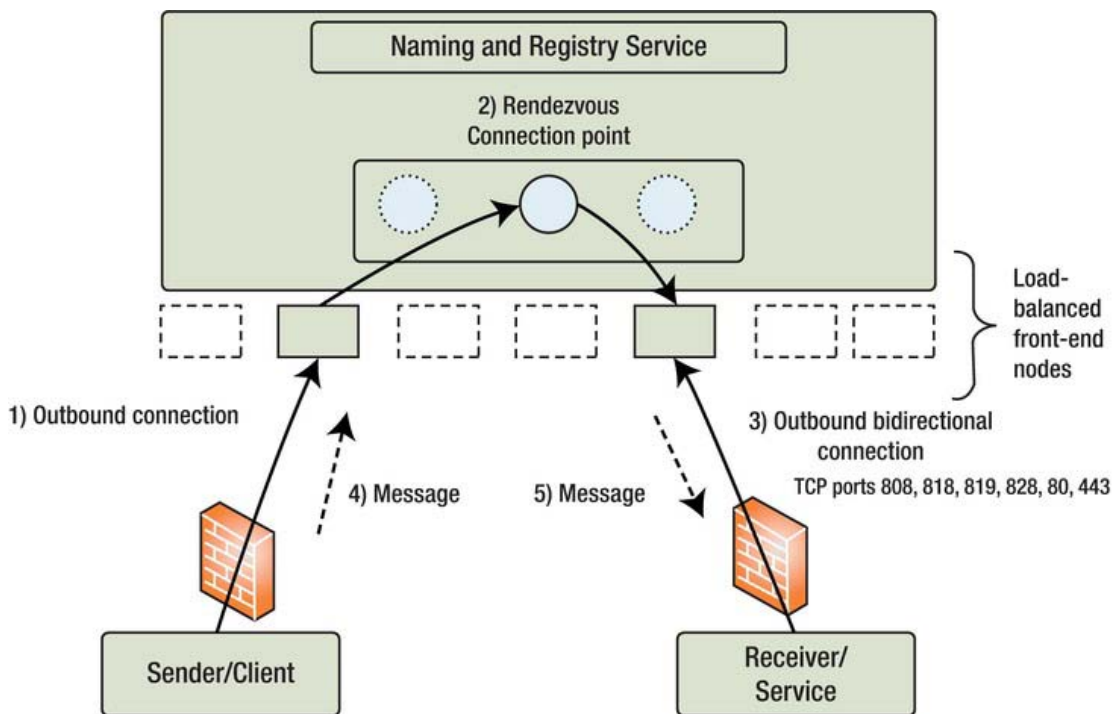


Figure 8-10. AppFabric Service Bus relay service

As shown in Figure 8-10, the service opens an outbound connection with a bidirectional socket to the AppFabric Service Bus relay service. The service registry registers the listener's endpoint in its naming tree for client applications to resolve. Most the AppFabric Service Bus listener bindings require the following TCP ports opened on firewall or the NAT router for outbound communication: 808, 818, 819, 828, 80, and 443.⁴

Note that you don't need to open any inbound ports in your firewall or NAT router for the end-to-end communication to work when using the AppFabric Service Bus. Therefore, the listener application can be running behind a firewall, NAT router, and even with a dynamic IP address. The client application initiates an outbound connection to the relay service with the appropriate service address that can be resolved from the service registry. The AppFabric Service has a load-balanced array of nodes that provide the necessary scalability to the client and service communications. When the client sends a message to the service, the message is relayed by the relay service to the appropriate node that is holding reference to the listener's endpoint. Finally, the relay service sends the message to the service over the listener's outbound bidirectional socket.

The AppFabric Service Bus URI naming scheme restricts listeners from registering more than one listener on a URI scope. For example, if you have a service with the URI `/energy/california`, you can't register any listener with a URI suffix of `/energy/California—/energy/california/sanfrancisco`, `/energy/california/sanramon`, and so on. You can register a service with the same URI root address,

⁴ Port information available in the AppFabric SDK: <http://msdn.microsoft.com/en-us/library/dd582710.aspx>.

such as /energy/sanfrancisco or /energy/sanjose. The AppFabric Service Bus uses the longest-prefix match algorithm to relay messages to the services. The longest URI under URI scope is evaluated and used to relay the message. So, in your service, you can process the entire URI suffix directory for query processing or filtering.

AppFabric Service Bus Bindings

The AppFabric Service Bus SDK comes with an API for programming AppFabric Service Bus applications. The namespace for AppFabric Service Bus classes is Microsoft.ServiceBus. The AppFabric Service Bus supports bindings similar to Windows Communications Foundation (WCF) bindings. Microsoft architected the AppFabric Service Bus with the vision of supporting the existing WCF programming model so that WCF developers can design and develop services for AppFabric Service Bus with their existing skill sets. The fundamental difference between AppFabric Service Bus bindings and WCF bindings is at the transport level, which is completely opaque to the programming model. The AppFabric Service Bus API provides binding classes that can be used in your WCF applications for binding to the AppFabric Service Bus relay service.

In traditional WCF applications, the service runs with specified bindings on local or remote servers, and client applications connect to the services directly. In traditional WCF, the notion of a relay service doesn't exist. Most of the standard WCF bindings have a direct match in the AppFabric Service Bus bindings. Table 8-3 lists the WCF bindings and the AppFabric Service Bus bindings side by side.

Table 8-3. *WCF and AppFabric Service Bus Bindings*

WCF Binding	AppFabric Service Bus Relay Binding	Description
BasicHttpBinding	BasicHttpRelayBinding	Both bindings use simple HTTP transport. BasicHttpRelayBinding uses the HTTP transport channel to the relay service.
WebHttpBinding	WebHttpRelayBinding	Both bindings support HTTP, XML, and raw binary encodings like base64. Popularly used in REST-style interfaces.
WS2007HttpBinding	WS2007HttpRelayBinding	Both bindings support the Organization for the Advancement of Structured Information Standards (OASIS) standard versions of ReliableSession and Security. WS2007HttpRelayBinding doesn't support atomic TransactionFlow protocols because the MSDTC isn't available between your service and the AppFabric Service Bus.
WSHttpContextBinding	WSHttpRelayContext Binding	Both bindings support context-enabled binding. WSHttpRelayContextBinding enables context-enabled binding between your service and the relay service. You can use SOAP headers for exchanging context.

WCF Binding	AppFabric Service Bus Relay Binding	Description
NetTcpBinding	NetTcpRelayBinding	These bindings are the TCP counterpart of the WSHttp bindings you saw earlier. The NetTcpRelayBinding uses binary message encoding and TCP for message delivery between your service and the relay service.
NetTcpContextBinding	NetTcpRelayContext Binding	NetTcpRelayContextBinding binding uses a context-enabled binding between your service and the relay service. You can use SOAP headers to exchange context.
N/A	NetOnewayRelayBinding	The NetOnewayRelayBinding is available only in the AppFabric Service Bus and doesn't have any corresponding binding in WCF. This binding supports only one-way messages between your service and the relay service.
N/A	NetEventRelayBinding	The NetOnewayRelayBinding is available only in the AppFabric Service Bus and doesn't have any corresponding binding in WCF. The NetEventRelayBinding enables one-way multicast eventing between multiple publishers and subscribers. This binding is used in Internet-scale publish-subscribe scenarios.

The AppFabric Service Bus Relay bindings offer you a complete spectrum of choices when you're selecting a high-performance binding like the NetTcpRelayBinding or a more interoperable and flexible binding like the WSHttpRelayBinding. All the bindings depend on the relay service to decide the message communication path between the clients and the services.

Message Buffer

The AppFabric Service Bus bindings for the WCF-style communications are designed for synchronous communications between the sender and the receiver. This means the receiver must be running to receive the message sent by the sender; otherwise, the message will get lost. The relay service doesn't contain a message store for storing and forwarding messages sent by senders to receivers. At Internet scale, the existence of the senders and receivers 100% of the time is an unrealistic expectation because senders and receivers depend on external and internal dependencies like server availability, on-premises network resources, network availability, bandwidth, and so on, that pose a significant availability risk for synchronous communications.

The AppFabric Service Bus offers a message buffer service for storing messages in a temporary cache for asynchronous communication between clients and servers. The AppFabric Service Bus buffers expose the REST API for applications to create a message buffer, send messages to the message buffer, and retrieve messages from the message buffer. The messages stored in a message buffer on the server

don't survive server reboots. The message buffers themselves are replicated across multiple servers to provide redundancy, but messages stored in message buffer are stored in the server memory and are lost when the server reboots or crashes. When you design your application to use a message buffer, you have to design redundancy into the application. If you need redundancy for your messages in the server, you should consider using either Windows Azure Queue storage or SQL Azure. The message buffer also uses ACS authentication to authenticate client applications.

Queues and Topics

Released with AppFabric SDK 1.5 are two new features called Queues and Topics. Queues provide a durable messaging mechanism, and Topics builds upon the queuing structure by adding the ability to create topics for which consumers can create rules by which to filter messages. This provides a robust pattern for delivering messages to multiple subscribers with a publish-subscribe pattern. These will be discussed in-depth below in the section "AppFabric Messaging Queues and Topics."

■ **Note** Queues and Topics are replacing Message Buffers, which will be deprecated in future releases.

Programming with the AppFabric Service Bus

This section dives into programming applications with the AppFabric Service Bus. The AppFabric Service Bus API provides WCF-like bindings for senders to send messages to receivers via the relay service. The job of the relay service is to receive messages from the sender(s) and relay those messages to the appropriate receiver(s). The AppFabric Service Bus bindings you saw in the previous sections consist of all the communication logic to communicate with the relay service. From a programmer's perspective, you must understand the limitations of and differences between WCF bindings and AppFabric Service Bus bindings in order to program AppFabric Service Bus applications. The WCF-like programming model reduces the barriers to entry for .NET developers and also enables easy porting of existing WCF applications to the AppFabric Service Bus.

The steps to create an AppFabric Service Bus application are as follows:

1. Create an AppFabric Service Bus namespace by doing the following:
2. Navigate to <http://windows.azure.com> in your browser, and log in using your LiveID.
3. Choose Service Bus, Access Control, & Caching from the lower left hand pane.
4. If not already expanded, expand the AppFabric tree node by clicking the Arrow/triangle next to it.
5. Select the Service Bus node by clicking on it.
6. Click the New button on the upper left to create a new namespace. To modify an existing namespace, select it in the middle pane, and then click Modify. You should see the dialog shown in Figure 8-11.

7. Provide a name for your namespace, a region/Country (i.e., data center) in which it will be located and, if desired, a Connection Pack size. Connection packs provide discounts if you are willing to subscribe to a certain number of connections every month. Be sure you need them because you will be charged for them every month whether they are used or not.
8. Design AppFabric contracts between the servers and the clients.
9. Implement the service contracts.
10. Design a bindings plan between the servers and clients for complex services using multiple bindings. This plan lists the AppFabric Service Bus bindings used for every message communication.
11. Create a security plan for relay- and message-level security between the clients and the servers. Some of the popularly used security scenarios include the following:
 - X.509 certificates for message security
 - ACS integration with a third-party identity provider (Windows Identity Foundation, ADFS v2.0, LiveID, and so on)
 - ACS integration with a client generated SWT token
 - ACS integration with a shared issuer key
 - Custom message security
12. Design endpoints for the service.
13. Design service hosting. This design includes whether the service will be hosted on-premises or in the cloud.
14. Design the scalability and availability for the service.
15. Design client applications for the service contract.

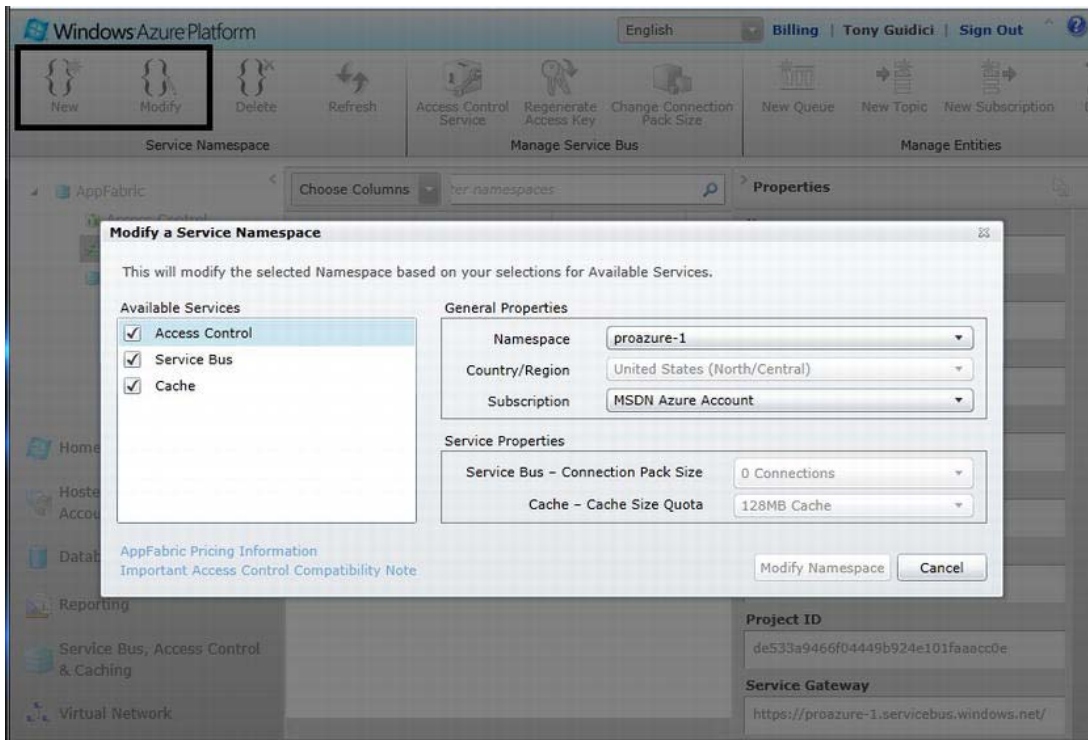


Figure 8-11. Modifying an existing Service Bus namespace.

The relay bindings are the core concepts for programming AppFabric Service Bus applications. This section covers relay bindings, queues, and routers. For the purpose of demonstration, I use a simple energy-management service that puts the AppFabric Service Bus’s capabilities in a business context.

ProAzure Energy Service Example

ProAzure Energy is a sample service I use in most of the demonstrations in this chapter. The ProAzure Energy service offers utility companies energy-related data from consumer and commercial buildings. For the purpose of this demo, assume that the ProAzure Energy service offers the following three services to the utility companies:

- *Energy meter monitoring:* A control gateway device monitors the energy meters in buildings and sends energy meter values to the ProAzure Energy head-end software periodically. The utility companies can then access these values through a subscription service.

■ **Note** Assume the head-end software is either in the cloud or on-premises at the ProAzure Energy company site. It definitely isn't on the customer's site where the actual device monitoring takes place. Also assume there is one gateway per building that can monitor different types of energy devices.

- *Lighting monitoring and control:* A control gateway device in buildings monitors the light switches using a control network protocol. The gateway device accepts ON/OFF commands from the ProAzure head-end software to turn the lights on and off, respectively. The gateway device also send real-time light-switch values to the head-end software when an ON or OFF switch event takes place on the switch either manually or programmatically.

■ **Note** Assume that the control gateway device is control-network-protocol agnostic. That means it supports all control network protocols over power lines to communicate with the energy devices. The gateway has Internet connectivity on one side and control network connectivity on another.

- *Heating Ventilation Air Conditioning (HVAC) monitoring and control:* A control gateway device in buildings monitors the HVAC devices using a control network protocol. The gateway device accepts the following HVAC commands:
 - *SETPOINT:* Changes the set point of the HVAC to the specified value in degrees Fahrenheit (°F).
 - *HEAT:* Sets the HVAC value to heating mode.
 - *COOL:* Sets the HVAC value to the cooling mode.
 - *OFF:* Sets the HVAC value to the OFF mode.
- The control gateway device also sends the set-point value, temperature, and heat/cool HVAC value to the ProAzure head-end software when it changes locally or on a periodic basic.

Figure 8-12 illustrates the high-level architecture of the ProAzure Energy service.

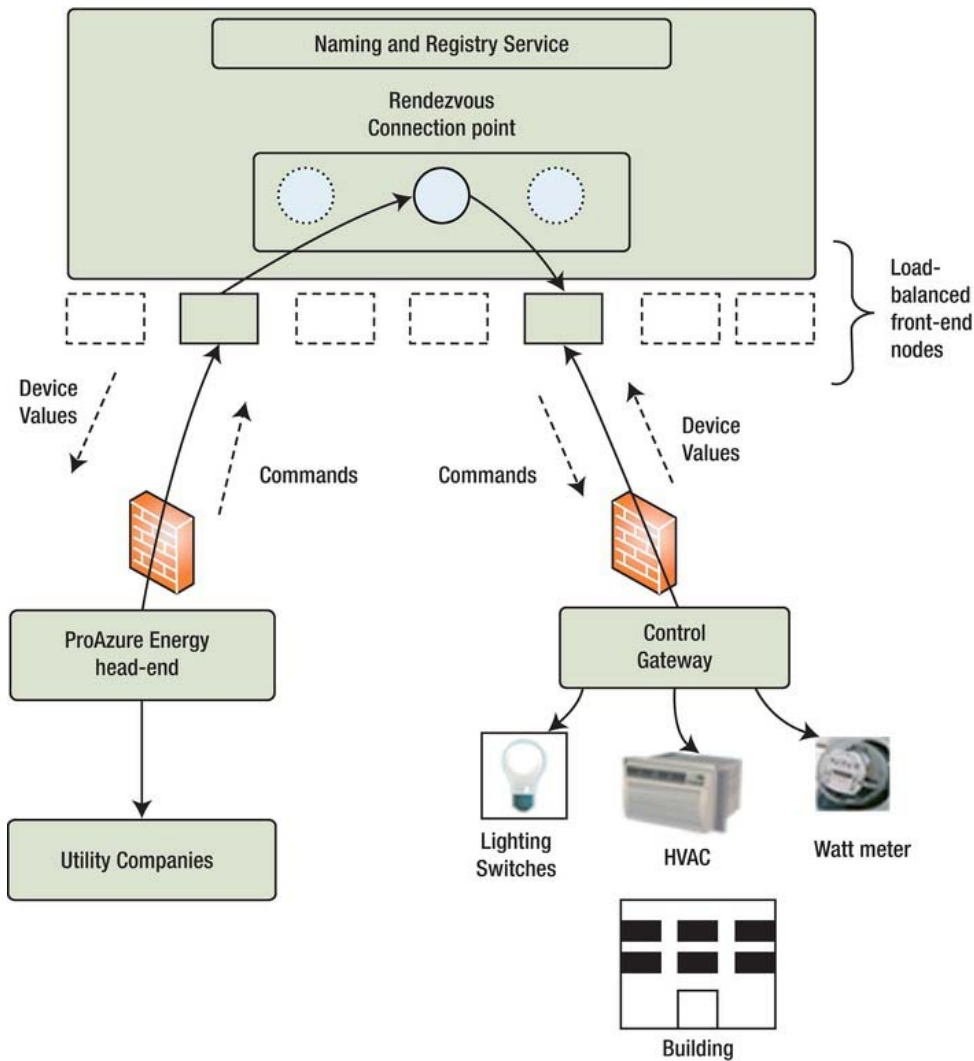


Figure 8-12. ProAzure Energy service architecture

Some of the important characteristics of the ProAzure Energy Service are as follows:

- The service monitors thousands of control gateways, which in turn manage energy devices in buildings.
- The control gateways communicate with the AppFabric Service Bus relay service to send and receive messages.

- The control gateways are clients of the head-end server as well as servers to receive commands from the head-end server.
- The control gateways may be behind firewalls.
- The ProAzure head-end server can be hosted either in the cloud in Windows Azure or on-premises at the ProAzure Energy service data center.
- The ProAzure Energy service head-send server uses the AppFabric Service Bus relay service to send and receive messages.

In addition to the device commands, the control gateway also supports the following commands for its own configuration and monitoring:

- *ONLINE*: Periodically, the control gateway sends an online message to let the head-end know of its continued availability.
- *UPLOAD_SOFTWARE*: This command is used to upload the software on the control gateway.

In the following sections, you learn how to leverage different AppFabric Service Bus bindings, queues, and routers to implement the ProAzure Energy service.

NetOnewayRelayBinding

NetOnewayRelayBinding supports one-way messages from client to the server. The method signatures for one-way methods in the service contract must not return any values. One-way methods are optimized for one-way TCP communications between the senders to the relay service and then to the receivers. The default size of the message is set to 65,536 bytes. The receiver using the NetOnewayRelayBinding opens a bidirectional TCP connection on outbound TCP port 828 for an SSL connection and TCP port 808 for a non-SSL connection. If the TCP outbound ports are unavailable due to environmental policies or port conflicts, you can configure the AppFabric Service Bus to use the HTTP protocol instead. The HTTP protocol polls the relay service through outbound ports 443 for SSL and 80 for non-SSL communications.

Figure 8-13 illustrates the workings of NetOnewayRelayBinding.

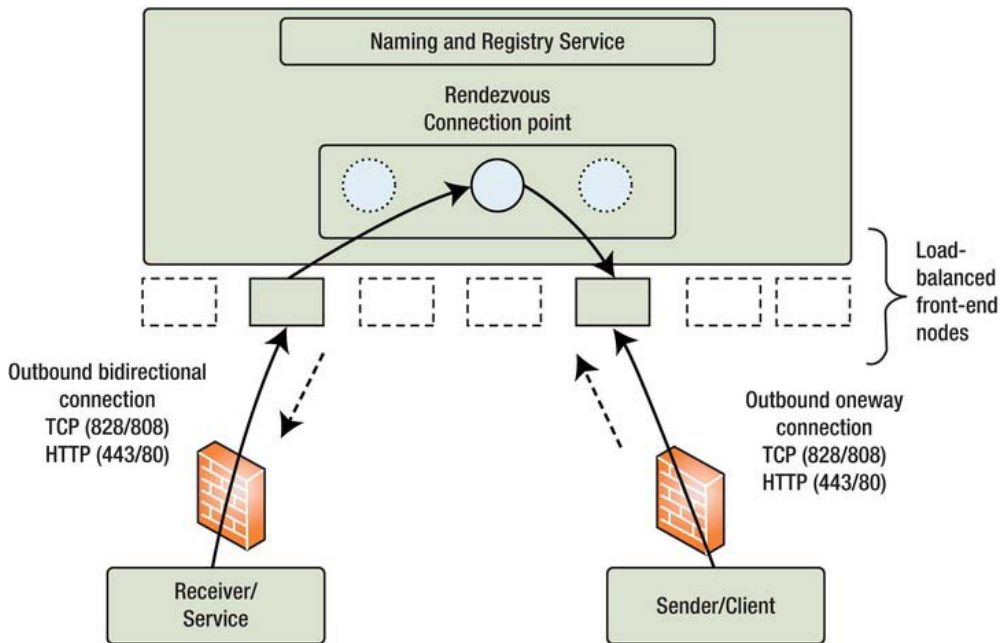


Figure 8-13. *NetOnewayRelayBinding*

For the purpose of demonstrating `NetOnewayRelayBinding`, in this section you design part of the ProAzure Energy sample service. Based on the requirements discussed in the previous section, you use the following communications from the control gateway to the head-end server to use the `NetOnewayRelayBinding`:

Sending energy meter value (kWh) to the head-end server periodically

- Sending light switch value (ON/OFF) to the head-end server when the state of the switch changes.
- Sending the HVAC set-point value to the head-end server when the set point changes.
- Sending the HVAC mode value (OFF/COOL/HEAT) to the head-end server when the HVAC mode changes.

The service project for this example is `NetOnewayRelayServer`, and the client project is `NetOnewayRelayClient`.

AppFabric Contract

The service contract represents the interface contract between the client and the server. The contract abstracts the interface of the server from its implementation. For the four communication requirements

defined in the previous section, you design four methods in a service contract interface named `IOnewayEnergyServiceOperations`, as shown in Listing 8-3.

Listing 8-3. *Service Contract `IOnewayEnergyServiceOperations`*

```
[ServiceContract(Name = "IOnewayEnergyServiceOperations.",
Namespace = "http://proazure/ServiceBus/energyservice/headend")]
public interface IOnewayEnergyServiceOperations
{
    [OperationContract(IsOneWay=true)]
    void SendKwhValue(string gatewayId, string meterId,
double kwhValue, DateTime utcTime);
    [OperationContract(IsOneWay = true)]
    void SendLightingValue(string gatewayId, string switchId,
int lightingValue, DateTime utcTime);
    [OperationContract(IsOneWay = true)]
    void SendHVACSetPoint(string gatewayId, string hvacId,
int setPointValue, DateTime utcTime);
    [OperationContract(IsOneWay = true)]
    void SendHVACMode(string gatewayId, string hvacId,
int mode, DateTime utcTime);
}

public interface IOnewayEnergyServiceChannel : IOnewayEnergyServiceOperations,
IClientChannel { }
```

The `IOnewayEnergyServiceOperations` define four operations you implement in the head-end server for the control gateway to call to send the values. Note the `IsOneWay=true` property of the `OperationContract` attribute, and also note that none of the one-way methods return any values. This is a requirement for all one-way methods in the AppFabric Service Bus.

■ **Tip** Always explicitly define the name and namespace for the service contract as a best practice. Doing so ensures a unique namespace for your contract and avoids any conflicts with default values.

The `IOnewayEnergyServiceChannel` defines a channel for client communications that inherits from the `IOnewayEnergyServiceOperations` and `IClientChannel` interfaces.

■ **Note** All the code for the interfaces is available in the `EnergyServiceContract` project in the `Ch8Solution.sln` Visual Studio solution. Before opening the solution, download the latest Windows Azure AppFabric SDK, also known as the AppFabric SDK.

Service Implementation

After the contract is designed, the next step is to implement the contract in the head-end server. In the interest of keeping the book conceptual, you create a simple implementation of the contract that prints out the received messages to the console. Listing 8-4 shows the implementation of the `IOnewayEnergyServiceOperations` interface.

Listing 8-4. IOnewayEnergyServiceOperations Implementation

```
[ServiceBehavior(Name = "OnewayEnergyServiceOperations",
Namespace = "http://proazure/ServiceBus/energyservice/headend")]
public class OnewayEnergyServiceOperations :
EnergyServiceContract.IOnewayEnergyServiceOperations
{
    public void SendKwhValue(string gatewayId, string meterId,
double kwhValue, DateTime utcTime)
    {
        Console.WriteLine(String.Format
("{0}>Energy Meter {1} value:{2:0.00} kWh @ {3}",
gatewayId, meterId, kwhValue, utcTime.ToString("s")));
    }
    public void SendLightingValue(string gatewayId, string switchId,
int lightingValue, DateTime utcTime)
    {
        Console.WriteLine(String.Format
("{0}>Changed lightbulb state of switch {1} to {2}",
gatewayId, switchId, ((lightingValue == 1) ? "ON" : "OFF")));
    }
    public void SendHVACSetPoint(string gatewayId, string hvacId,
int setPointValue, DateTime utcTime)
    {
        Console.WriteLine(String.Format
("{0}>HVAC {1} has SETPOINT value:{2:0} F @ {3}",
gatewayId, hvacId, setPointValue, utcTime.ToString("s")));
    }
    public void SendHVACMode(string gatewayId, string hvacId,
int mode, DateTime utcTime)
    {
        Console.WriteLine(String.Format
("{0}>HVAC {1} MODE is set to {2} @ {3}", gatewayId,
hvacId, GetHVACModeString(mode), utcTime.ToString("s")));
    }
}
```

Note that all the concepts applied until now are the same as any WCF service implementation.

Service Binding

Bindings define the transport, encoding, and protocol required by the WCF services and clients to communicate with each other. A binding configuration is applied to the endpoint to represent the transport, encoding, and protocol used for communication between client and services. `NetOnewayRelayBinding` is an AppFabric Service Bus binding that defines one-way communication

between the client, relay server, and service. Listing 8-5 shows the binding configuration in App.config for the OnewayEnergyServiceOperations service implementation.

Listing 8-5. Service Binding for OnewayEnergyServiceOperations

```
<bindings>
  <netOnewayRelayBinding>
    <binding name="default" />
  </netOnewayRelayBinding>
</bindings>
```

The bindings section defines the netOnewayRelayBinding. You can define multiple bindings in the bindings section and then later apply one to the service endpoint. The netOnewayRelayBinding makes a TCP outbound connection on port 828 by default, which is on a secure connection. For a non-secure TCP connection, it uses port 808. In most enterprises, no outbound connections other than HTTP on port 80 or SSL on port 443 are allowed due to corporate security policies. In such scenarios, you can configure netOnewayRelayBinding to establish an HTTP connection with the relay service over port 80 or 443. The AppFabric Service Bus environment supports a ConnectivityMode property you can set to one of these enum values: AutoDetect, TCP, or HTTP, as listed in Table 8-4.

Table 8-4. ConnectivityMode Values

ConnectivityMode	Description
AutoDetect	This option automatically detects the communication options between TCP and HTTP depending on the availability of TCP and HTTP. If both are available, it chooses TCP over HTTP.
TCP	This is the default mode of communication. If the TCP option is selected, the application opens a TCP connection with the relay service on outbound TCP port 828 for secure TCP connections.
HTTP	If the HTTP option is selected, the application opens an HTTP connection with the relay service on port 443 for SSL communications and port 80 for non-SSL communications. The HTTP connection on the receiver side polls the relay service for messages. Use this option only if you don't have TCP outbound connections restricted, because the HTTP option has a performance impact due to the polling mechanism.

You can set the ConnectivityMode of the netOnewayRelayBinding using `ServiceBusEnvironment.SystemConnectivity.Mode = ConnectivityMode.AutoDetect;` `SystemConnectivity.Mode` sets the value of `ConnectivitySettings` that represents the AppFabric Service Bus connectivity. The default connectivity mode between the AppFabric Service Bus and the service is TCP. If you're running your service behind a firewall, you can use the HTTP binding. If you aren't sure about the network constraints, use AutoDetect mode, where the Service Bus selects TCP by default but automatically switches to HTTP if TCP connectivity isn't available. You can configure end-to-end security between the client and the server as shown in Listing 8-6.

Listing 8-6. Binding Security for netOnewayRelayBinding

```

<netOnewayRelayBinding>
<binding name="default" >
<security mode="Transport" relayClientAuthenticationType="None" />
</binding>
</netOnewayRelayBinding>

```

The mode attribute supports four values, as listed in Table 8-5.

Table 8-5. End to End Security Values

Security Mode Value	Description
Message	Provides SOAP message security.
Transport	Provides transport-level security like SSL.
TransportWithMessageCredential	Provides transport-level security like SSL along with message-level client security.
None	Provides no security between client and server.

Relay Security

The AppFabric Service Bus integrates with ACS to provide the authentication and authorization required for accessing and creating service endpoints in the AppFabric Service Bus. Even though ACS can be configured to use an external identity provider like ADFS v2.0 or Windows Live ID, this example uses a shared secret to authenticate with ACS for both the service and the client. Listing 8-7 shows the code to pass an issuer name and issuer key as credentials to authenticate with the AppFabric Service Bus.

Listing 8-7. Shared Secret Authentication

```

TransportClientEndpointBehavior sharedSecretServiceBusCredential =
new TransportClientEndpointBehavior();
    sharedSecretServiceBusCredential.CredentialType =
TransportClientCredentialType.SharedSecret;
    sharedSecretServiceBusCredential.Credentials.SharedSecret.IssuerName =
    issuerName;
    sharedSecretServiceBusCredential.Credentials.SharedSecret.IssuerSecret =
    issuerKey;
ServiceHost Host = new ServiceHost(serviceType);
Host.Description.Endpoints[0].Behaviors.Add(behavior);

```

In Listing 8-7, you create a `TransportClientEndpointBehavior` object and select the credential type `SharedSecret` to use the issuer and issuer key as the authenticating credentials.

Figure 8-14 shows the service namespace page with Service Bus credentials. You can use the default issuer name and default issuer key in the shared secret values while connecting to the Service Bus.

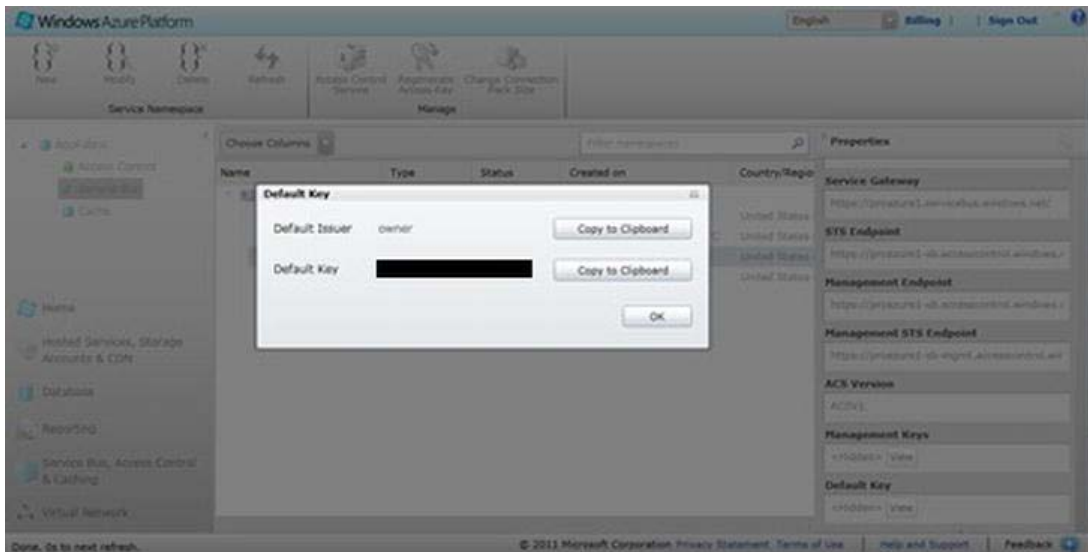


Figure 8-14. Credentials Management page

You can also define the shared secret in `app.config`, as shown in Listing 8-8. If you define credentials as your service behavior and assign it to the service endpoint, then you don't need to initialize transport client credentials in the code.

Listing 8-8. SharedSecret Declaration

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="sharedSecretClientCredentials">
          <transportClientEndpointBehavior credentialType="SharedSecret">
            <clientCredentials>
              <sharedSecret issuerName="owner"
issuerSecret="wJBJaobUmarWn6kqv7QpaaRh3ttNvr3w10jiotVE0L4=" />
            </clientCredentials>
          </transportClientEndpointBehavior>
        </behavior>
      </endpointBehaviors>
    </behaviors>
    <bindings>
      <!-- Application Binding -->
      <netOnewayRelayBinding>
        <binding name="default" />
      </netOnewayRelayBinding>
    </bindings>
```

```

<services>
  <service name="EnergyServiceContract.OnewayEnergyServiceOperations">
    <endpoint address="sb://proazure-
1.servicebus.windows.net/OnewayEnergyServiceOperations/"
      binding="netOnewayRelayBinding"

      behaviorConfiguration="sharedSecretClientCredentials"

      bindingConfiguration="default"
      name="RelayEndpoint"
      contract="EnergyServiceContract.IOnewayEnergyServiceOperations" />
    </service>
  </services>
</system.serviceModel>
</configuration>

```

In Listing 8-8, the transport client behavior is defined under the `sharedSecretClientCredentials` element, which is assigned as the `behaviorConfiguration` of the service endpoint.

Message Security

Message security refers to the security of the message as it travels from client to service via the AppFabric Service Bus. As discussed earlier, the AppFabric Service Bus API offers four options for message security in the enumeration `Microsoft.ServiceBus.EndToEndSecurityMode`: `None`, `Transport`, `Message`, and `TransportWithMessageCredentials`. `netOnewayRelayBinding` doesn't support `TransportWithMessageCredentials`. If you want to use a certificate in the client, you have to explicitly configure the service certificate in the client; in a one-way message, there is no direct connection between the client and service. When the client sends a message, the service may not be available, and so the client can't negotiate the certificate with the service.⁵

The `netOnewayRelayBinding` example provides configuration files for default (`AppBasic.config`), `Transport` (`AppTransport.config`), `Message` without client credentials (`AppMsgSecNoClientCreds.config`), and `Message` with username credentials (`AppMsgSecUsernameClientCreds.config`). Figure 8-15 shows the client (`NetOnewayRelayClient`) and service (`NetOnewayRelayServer`) projects.

⁵ Juval Lowy. Securing The .NET Service Bus. MSDN. <http://msdn.microsoft.com/en-us/magazine/dd942847.aspx>.

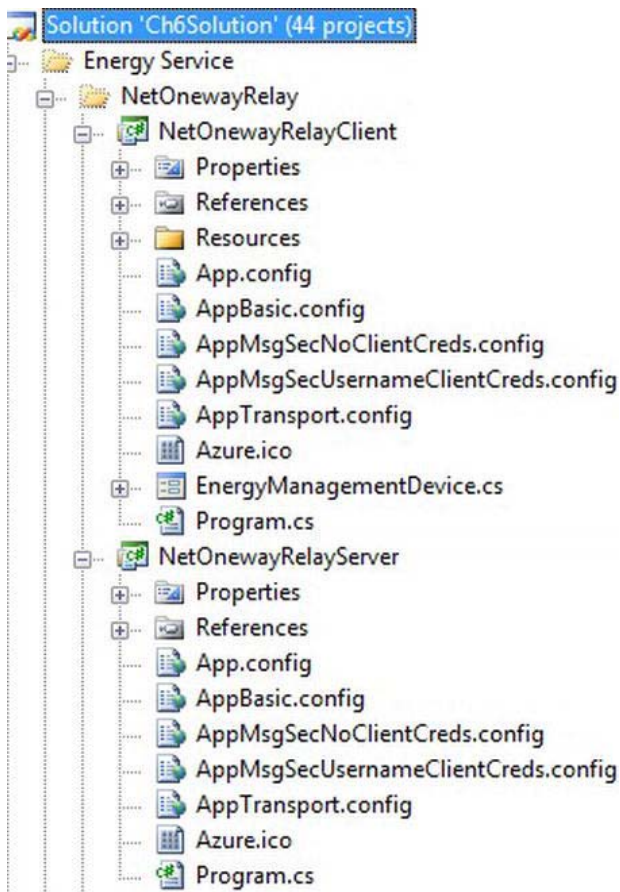


Figure 8-15. *NetOnewayRelayBinding example*

To use any particular message security, copy and paste the contents of the appropriate configuration file into `App.config` for the project in both client and service, and recompile the project. The examples use the `TempCA.cer` X.509 certificate for the service identity, which you can find in the code directory of `Ch8Solution`. Listing 8-9 shows the contents of `AppMsgSecNoClientCreds.config` for the service, and Listing 8-10 shows the contents of `AppMsgSecNoClientCreds.config` for the client.

Listing 8-9. *AppMsgSecNoClientCreds.config for the Service*

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <!--Configure certificate for service identity-->
```

```

    <behavior name = "CertificateProtection">
      <serviceCredentials>
        <serviceCertificate
          findValue      = "TempCA"
          storeLocation  = "LocalMachine"
          storeName      = "My"
          x509FindType   = "FindBySubjectName"
        />
      </serviceCredentials>
    </behavior>
  </serviceBehaviors>
  <endpointBehaviors>
    <behavior name="sharedSecretEndpointBehavior">
      <transportClientEndpointBehavior credentialType="SharedSecret">
        <clientCredentials>
          <sharedSecret issuerName="ISSUER_NAME" issuerSecret="ISSUER_SECRET" />
        </clientCredentials>
      </transportClientEndpointBehavior>
    </behavior>
  </endpointBehaviors>
</behaviors>
<bindings>
  <!-- Application Binding -->
  <netOnewayRelayBinding>
    <binding name = "OnewayMessageSecurity">
      <security mode = "Message">
        <message clientCredentialType = "None"/>
      </security>
    </binding>
  </netOnewayRelayBinding>
</bindings>
<!--Configure certificate for message security-->

<services>
  <service name="EnergyServiceContract.OnewayEnergyServiceOperations"
    behaviorConfiguration = "CertificateProtection">
    <endpoint address=
"sb://proazure.servicebus.windows.net/OnewayEnergyServiceOperations/"
    binding="netOnewayRelayBinding"
    bindingConfiguration="OnewayMessageSecurity"
    name="RelayEndpoint"
    contract="EnergyServiceContract.IOnewayEnergyServiceOperations"
    behaviorConfiguration="sharedSecretEndpointBehavior" />
  </service>
</services>
</system.serviceModel>
</configuration>

```

Listing 8-10. *AppMsgSecNoClientCreds.config for the Client*

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <netOnewayRelayBinding>
        <binding name = "OnewayMessageSecurity">
          <security mode = "Message">
            <message clientCredentialType = "None"/>
          </security>
        </binding>
      </netOnewayRelayBinding>
    </bindings>
  </system.serviceModel>
  <behaviors>
    <endpointBehaviors>
      <behavior name = "ServiceCertificate">
        <transportClientEndpointBehavior credentialType="SharedSecret">
          <clientCredentials>
            <sharedSecret issuerName="ISSUER_NAME" issuerSecret="ISSUER_SECRET" />
          </clientCredentials>
        </transportClientEndpointBehavior>
        <clientCredentials>
          <serviceCertificate>
            <scopedCertificates>
              <add targetUri = "sb://{your service
namespace}.servicebus.windows.net/OnewayEnergyServiceOperations/"
                findValue      = "TempCA"
                storeLocation  = "LocalMachine"
                storeName     = "My"
                x509FindType   = "FindBySubjectName"
                />
            </scopedCertificates>
          </serviceCertificate>
        </clientCredentials>
      </behavior>
    </endpointBehaviors>
  </behaviors>
  <client>
    <!-- Service Endpoint -->
    <endpoint name="RelayEndpoint"
      contract="EnergyServiceContract.IOnewayEnergyServiceOperations"
      binding="netOnewayRelayBinding"
      bindingConfiguration="OnewayMessageSecurity"
      address=
"sb://proazure.servicebus.windows.net/OnewayEnergyServiceOperations/"
      behaviorConfiguration = "ServiceCertificate"
    >
    <identity>
      <dns value = "TempCA"/>
    </identity>
  </client>

```

```

        </endpoint>
    </client>
</system.serviceModel>
</configuration>

```

The TempCA X.509 certificate is configured in the service as well as the client in the behavior section of the configuration file. In production applications, you have to use a production certificate issued by a certificate authority. Note that the behavior elements in both the client and server configuration include the transport client endpoint behavior set to shared secret. You can also initialize the `TransportClientEndpointBehavior` class in the client and server code. In production applications, you should encrypt the issuer credentials wherever they're stored. The X.509 certificate is used.

Service Endpoints

A WCF service endpoint defines how a client can communicate with the WCF service. The endpoint consists of four main attributes: the address of the endpoint, a binding that defines what protocol a client can use to communicate with the endpoint, a service contract that defines the operations available for the client to call, and a set of behaviors defining the local behavior of the endpoint. AppFabric Service Bus endpoints are similar to WCF endpoints. The only difference is the specific bindings used to communicate with the relay service.

Endpoints can be configured in application configuration files or programmatically. For the `netOnewayRelayBinding` example, Listing 8-11 shows the service endpoint definition from the `App.config` file.

Listing 8-11. *netOnewayRelayBinding Endpoint*

```

<!-- Service Endpoint -->
<endpoint
    address="sb://{your service namespace}
.servicebus.windows.net/OnewayEnergyServiceOperations/"
    behaviorConfiguration="sharedSecretClientCredentials"
    binding="netOnewayRelayBinding"
    bindingConfiguration="default"
    name="RelayEndpoint"
    contract="EnergyServiceContract.IOnewayEnergyServiceOperations" />

```

In Listing 8-11, the binding is set to `netOnewayRelayBinding`, and the `bindingConfiguration` and `behaviorConfiguration` are pointers to the sections within the same configuration file. The address refers to the URI of the service endpoint. You can also create the URI of the service using the static method call

```
ServiceBusEnvironment.CreateServiceUri("sb", serviceNameSpace, servicePath);
```

where `servicePath` is the part of the URI after `sb://proazure.servicebus.windows.net`. In this example, it's `OnewayEnergyServiceOperations`. The “sb” represents the scheme used to communicate with the AppFabric Service Bus. The scheme can be either “http” or “sb” depending on the binding you're using. For `netOnewayRelayBinding`, you must use the “sb” scheme.

Service Hosting

After you've defined the service contract, service implementation, bindings, and endpoints, you can create a host for the service, as shown in Listing 8-12.

Listing 8-12. Hosting the AppFabric Service Bus Service

```
TransportClientEndpointBehavior behavior =  
ServiceBusHelper.GetUsernamePasswordBehavior(issuerName, issuerKey);  
Host = new ServiceHost(typeof(OnewayEnergyServiceOperations));  
Host.Description.Endpoints[0].Behaviors.Add(behavior);  
Host.Open();
```

As shown in Listing 8-12, the `System.ServiceModel.ServiceHost` is used to host the service. The `TransportClientEndpointBehavior` object is created from the issuer name/issuer key and passed to the defined endpoint. Finally, the `Host.Open()` method opens the service for communication. If you define the issuer name and issuer key in the configuration file, then you don't have to initialize it programmatically. In this example, you define the transport client endpoint behavior in the configuration file.

Client Design

You can find the client application in the `NetOnewayRelayClient` Visual Studio project. From the business requirements perspective, the client application is the control gateway application that connects to the head-end server to send messages. Figure 8-16 illustrates the user interface for the `NetOnewayRelayClient` client application.

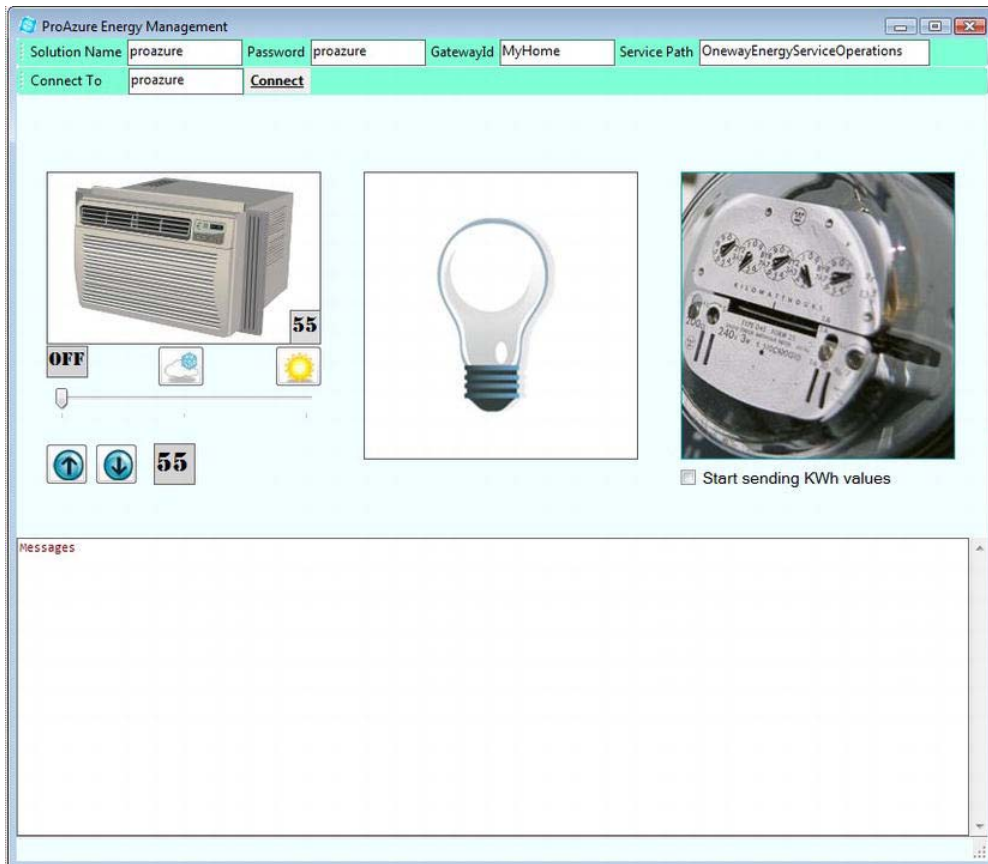


Figure 8-16. *NetOnewayRelayClient application Design View*

The client user interface has four main sections: configuration, HVAC operations, light switch operations, and meter reading, as discussed in the original requirements of the application. In the configuration section at the top of the form, you should enter your solution name and solution password. The Connect button establishes a connection to the AppFabric Service Bus. Any change to the HVAC set point or mode is sent to the head-end server by calling the `SendHVACSetPoint()` and `SendHVACMode()` methods on the server. Clicking the light bulb button turns the light switch on and off. Any change to the state of the light switch is sent to the server by calling the `SendLightingValue()` method on the head-end server. If you click on the energy meter button, a random kWh value is sent to the server by calling the `SendKwhValue()` method on the head-end server. If you check the “Start sending kWh values” check box, a random kWh value is sent to the head-end server every 10 seconds.

Listing 8-13 shows the code to initialize the channel to communicate with the server. The credentials are defined in the `app.config` file so they don’t need to be initialized in the code.

Listing 8-13. Client Communication Initialization

```
Uri address = ServiceBusEnvironment.CreateServiceUri
("sb", serviceNamespaceDomain, "OnewayEnergyServiceOperations");

ChannelFactory<IOnewayEnergyServiceChannel> netOnewayChannelFactory = new
ChannelFactory<IOnewayEnergyServiceChannel>("RelayEndpoint", new EndpointAddress(address));

IOnewayEnergyServiceChannel netOnewayChannel = channelFactory.CreateChannel();

channel.Open();
```

After the channel is opened successfully, you can call the methods on the service as follows:

```
netOnewayChannel.SendLightingValue(gatewayId, switchId, lightingValue, DateTime.UtcNow);
netOnewayChannel.SendKwhValue(gatewayId, meterId, kwhValue, DateTime.UtcNow);
```

■ **Note** In a real-world application, the control gateway polls the actual energy meter and sends kWh values to the head-end server. This example uses random numbers to simulate a real-world environment.

Running the Application

The steps required to run the end-to-end application are as follows:

1. Open App.config for the server and client, and configure them to represent your service namespace and issuer credentials.
2. Open a command prompt as Administrator, and navigate to the bin\Debug directory of the NetOnewayRelayServer project.
3. Run NetOnewayRelayServer.exe.
4. Enter the service namespace to start the service.
5. Open Windows Explorer and navigate to the bin\Debug directory of the NetOnewayRelayClient project.

■ **Note** Make sure the configuration for the server and the client match in terms of address and security.

6. Double-click NetOnewayRelayClient.exe to start the client application.
7. Click the Connect button to connect to the relay service. If the connection is successful, the text box displays success messages.

8. You can interact with the application by changing the state of HVAC, Light switch or the meter reading button. The client application calls the appropriate methods on the head-end server, and as a result the NetOnewayRelayServer.exe command prompt displays the received method calls.

Figure 8-17 illustrates a running instance of the client application, and Figure 8-18 illustrates the messages received on the server command prompt.

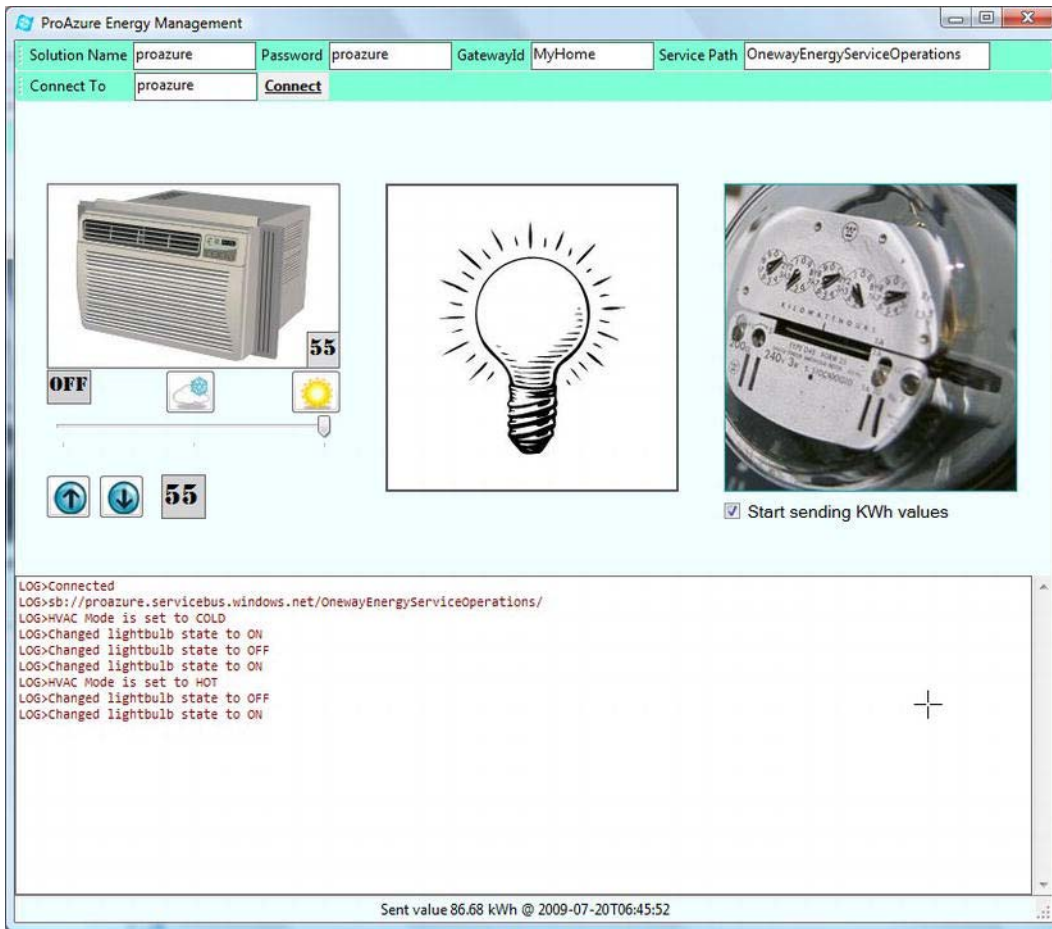


Figure 8-17. NetOnewayRelayClient application

```

C:\Users\tredkar\Desktop\Publications\Azure Services\Content\Code\6\NETServices July2009 CTP\...
Please enter the Solution name to use for this sample: proazure
Your Solution Password: *****
ServiceUri: sb://proazure.servicebus.windows.net/OnewayEnergyServiceOperations/
Service registered for public discovery.
Scheme: sb
Security Mode: Transport
Security RelayAuthType: RelayAccessToken
Security Transport.ProtectionLevel: EncryptAndSign
Press [Enter] to exit
MyHome>Changed lightbulb state of switch LightSwitch-1 to OFF
MyHome>Changed lightbulb state of switch LightSwitch-1 to ON
MyHome>HVAC HVAC-1 MODE is set to HOT @ 2009-07-20T06:44:48
MyHome>Energy Meter Meter-1 value: 30.80 kWh @ 2009-07-20T06:44:50
MyHome>Changed lightbulb state of switch LightSwitch-1 to OFF
MyHome>Energy Meter Meter-1 value: 92.84 kWh @ 2009-07-20T06:45:02
MyHome>Energy Meter Meter-1 value: 18.15 kWh @ 2009-07-20T06:45:12

```

Figure 8-18. NetOnewayRelayServer application

■ **Tip** If you want to observe the ports open or trace messages sent back and forth between the client and the service, you can use Microsoft's Network Monitor (netmon.exe), available at www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=983b941d-06cb-4658-b7f6-3088333d062f.

Figure 8-19 illustrates the Microsoft Network Monitor conversation tree of the interaction between NetOnewayRelayClient.exe and NetOnewayRelayServer.exe. Note the TCP outgoing port 828 and SSL connection in the conversation tree.

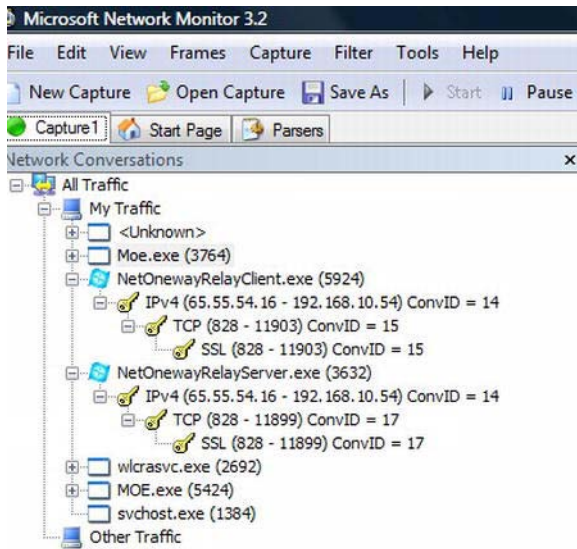


Figure 8-19. Microsoft Network Monitor capture

netEventRelayBinding

`netEventRelayBinding` extends the `netOnewayRelayBinding` by providing multicast messaging between multiple subscribers and publishers listening on the same rendezvous service endpoint. The `netEventRelayBinding` class inherits from `netOnewayRelayBinding`. This is the only binding that supports multiple receivers on the same service URI. Figure 8-20 illustrates the architecture of `netEventRelayBinding`.

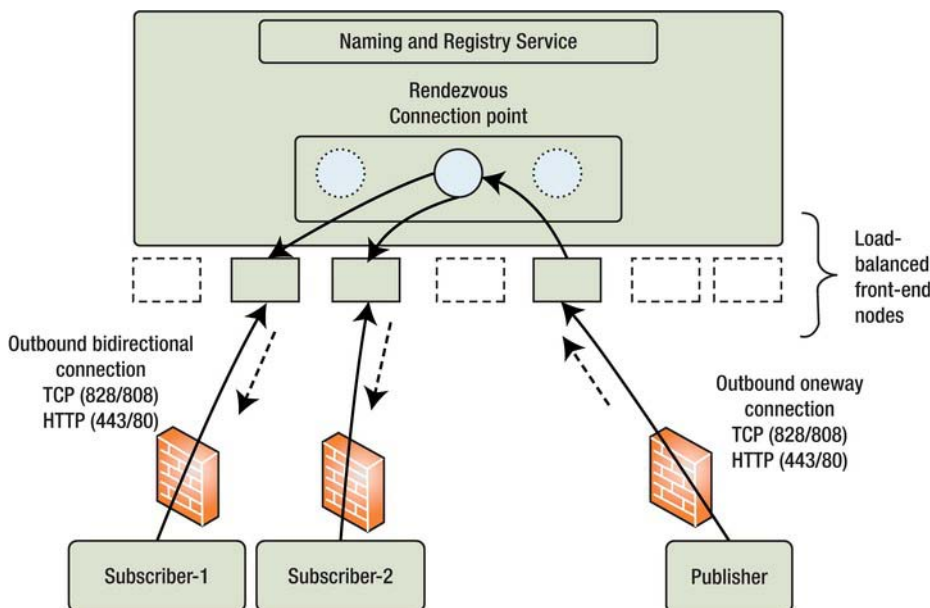


Figure 8-20. *netEventRelayBinding architecture*

In Figure 8-20, one publisher publishes messages on a defined endpoint URI, and two subscribers (Subscriber-1 and Subscriber-2) listen on the same endpoint URI. When the publisher sends a message to the endpoint URI, both receivers receive the message. The AppFabric Service Bus multicasts the message to all the subscribers of the URI. Internally, both the subscribers may be running on different front-end nodes. From the publisher and subscriber perspective, routing of the message to two subscribers is opaque and completely handled by the combination of `netEventRelayBinding` and the AppFabric Service Bus. Because `netEventRelayBinding` inherits from `netOnewayRelayBinding`, it supports the same connectivity modes and security features, as discussed for `netOnewayRelayBinding`.

You should use this binding if you require a publish-subscribe messaging system where a message needs to be sent to multiple receivers at the same time. `netEventRelayBinding` uses a multicast connection mode, whereas `netOnewayRelayBinding` uses a unicast connection mode.

In the ProAzure Energy service example, the control gateway needs to communicate with the head-end server about its availability and when it comes online and goes offline. This offers the head-end server better understanding of a gateway's online/offline pattern and can send scheduled commands to the control gateway only when it's online. The head-end server is a collection of small servers with dedicated specific roles. For example, there is a server instance that only sends scheduled commands to the control gateway when it's online. Another service checks for the required software upgrade on the control gateway and can upgrade the software on the control gateway when it's online. So, this example uses the `netEventRelayBinding` to send ONLINE/OFFLINE messages between the control gateway and the head-end server. When a control gateway is online, it periodically sends an ONLINE message to the head-end server. A control gateway also sends an OFFLINE message if it's shutting down gracefully. The service project for this example is `NetEventRelayServer`, and the client project is `NetEventRelayGateway` in the `Ch8Solution`. The `NetEventRelayGateway` project consists of `netOnewayRelayBinding` as well as `netEventRelayBinding` examples. The same application is used to send one-way as well as publish/subscribe messages.

AppFabric Contract

The AppFabric contract for the `netEventRelayBinding` example consists of two operations: `Online()` and `GoingOffline()`, as shown in Listing 8-14.

Listing 8-14. *netEventRelayBinding Service Contract*

```
[ServiceContract(Name = "IMulticastGatewayOperations.", Namespace =
"http://proazure/ServiceBus/energyservice/gateway")]
public interface IMulticastGatewayOperations
{
    [OperationContract(IsOneWay = true)]
    void Online(string gatewayId, string serviceUri, DateTime utcTime);
    [OperationContract(IsOneWay = true)]
    void GoingOffline(string gatewayId, string serviceUri, DateTime utcTime);
}

public interface IMulticastGatewayChannel : IMulticastGatewayOperations,
IClientChannel
{
}
```

The `IMulticastGatewayOperations` interface has two methods: `Online()` and `GoingOffline()`. Similar to the `netOnewayRelayBinding`, both methods must have the `IsOneWay=true` attribute and must not return any values. The `gatewayId` refers to the unique identifier of a gateway, and the `serviceUri` refers to the URI of the gateway service. I cover the URI of the gateway when I discuss `netTcpRelayBinding`.

Service Implementation

The implementation of the `IMulticastGatewayOperations` interface is shown in Listing 8-15.

Listing 8-15. *Implementation of the IMulticastGatewayOperations Interface*

```
[ServiceBehavior(Name = "MulticastGatewayOperations", Namespace =
"http://proazure/ServiceBus/energyservice/")]
public class MulticastGatewayOperations :
EnergyServiceContract.IMulticastGatewayOperations
{
    public void Online(string gatewayId, string serviceUri, DateTime utcTime)
    {
        Console.WriteLine(String.Format("{0}>ONLINE Uri:{1} @ {2}",
gatewayId, serviceUri, utcTime.ToString("s")));
    }
    public void GoingOffline(string gatewayId, string serviceUri, DateTime utcTime)
    {
        Console.WriteLine(String.Format("{0}>OFFLINE Uri:{1} @ {2}",
gatewayId, serviceUri, utcTime.ToString("s")));
    }
}
```


The implementation prints the name, URI, and the time values to the console.

Service Binding

The service binding for `netEventRelayBinding` is shown in Listing 8-16.

Listing 8-16. *netEventRelayBinding*

```
<netEventRelayBinding>
    <binding name = "OnewayMessageSecurity">

        </binding>
</netEventRelayBinding>
```

Relay Security

In the `netOnewayRelayBinding` example, you saw how to use shared-secret authentication with your ACS solution. This example explores the use of an SWT. Listing 8-17 shows the code segment required to authenticate using an SWT.

Listing 8-17. *SWT Authentication*

```
Uri address = ServiceBusEnvironment.CreateServiceUri("sb", serviceNamespaceDomain,
"Gateway/MulticastService");

TransportClientEndpointBehavior behavior = new TransportClientEndpointBehavior();
behavior.CredentialType = TransportClientCredentialType.SimpleWebToken;
behavior.Credentials.SimpleWebToken.SimpleWebToken =
SharedSecretCredential.ComputeSimpleWebTokenString(issuerName, issuerSecret);

ServiceHost host = new ServiceHost(typeof(MulticastGatewayOperations), address);
host.Description.Endpoints[0].Behaviors.Add(behavior);
```

The code creates an SWT from the issuer name and issuer secret key by calling the method `SharedSecretCredential.ComputeSimpleWebTokenString` (string issuerName, string issuerSecret) method from `Microsoft.ServiceBus.dll`.

Message Security

Similar to the `netOnewayRelayBinding` example, you can create specific configuration files for particular message security scenarios and then switch back and forth between these configuration files depending on the scenario you're executing. When you execute a particular security configuration, make sure you're switching the client security configuration consistently with the service configuration.

Service Endpoints

The service endpoint configuration of `netEventRelayBinding` in this example doesn't define the ACS authentication in the configuration file like `netOnewayRelayBinding`. The ACS authentication is handled in the code. Listing 8-18 shows the service configuration in of the `NetEventRelayServer`.

Listing 8-18. *Service Endpoint Configuration*

```
<services>
<service name="EnergyServiceContract.MulticastGatewayOperations">
  <endpoint address=""
            binding="netEventRelayBinding"
            bindingConfiguration="default"
            name="RelayMulticastEndpoint"
            contract="EnergyServiceContract.IMulticastGatewayOperations"
            />
</service>
</services>
```

The relay authentication is handled in the code and therefore isn't visible in the configuration file.

Service Hosting

The `netEventRelayBinding` example uses SWT tokens for relay authentication instead of issuer name and issuer key as in the `netOnewayRelayBinding` example. So, the service host has to create an SWT from the issuer name and issuer key. The code for the service host is shown in Listing 8-19.

Listing 8-19. *Service Hosting for netEventRelayBinding*

```
string serviceNamespaceDomain = "{your service namespace}"
string issuerName = "{ISSUER NAME}";
string issuerSecret = "{ISSUER KEY}";
ServiceBusEnvironment.SystemConnectivity.Mode = ConnectivityMode.AutoDetect;
TransportClientEndpointBehavior relayCredentials = new TransportClientEndpointBehavior();
relayCredentials.CredentialType = TransportClientCredentialType.SharedSecret;
relayCredentials.Credentials.SharedSecret.IssuerName = issuerName;
relayCredentials.Credentials.SharedSecret.IssuerSecret = issuerSecret;
Uri serviceAddress = ServiceBusEnvironment.CreateServiceUri("sb", serviceNamespaceDomain,
    "Gateway/MulticastService/");
ServiceHost host = new ServiceHost(typeof(MulticastGatewayOperations), serviceAddress);
host.Description.Endpoints[0].Behaviors.Add(relayCredentials);

host.Open();One the service hosts are started, they listen on the endpoint URI
sb://{your service namespace}.servicebus.windows.net/Gateway/MulticastService/
```

Client Design

In this example, the client application performs both the `netOnewayRelayBinding` and the `netEventRelayBinding` operations. When a control gateway comes online, it sends online messages every 10 seconds by calling the `Online()` method on the head-end server's multicast URI:

```
sb://proazure.servicebus.windows.net/Gateway/MulticastService/
```

When you close the client application, it sends an offline message by calling the `GoingOffline()` method on the head-end server's multicast URI:

```
sb://proazure.servicebus.windows.net/Gateway/MulticastService/
```

Figure 8-21 illustrates the design view of the `NetEventRelayGateway` client application.

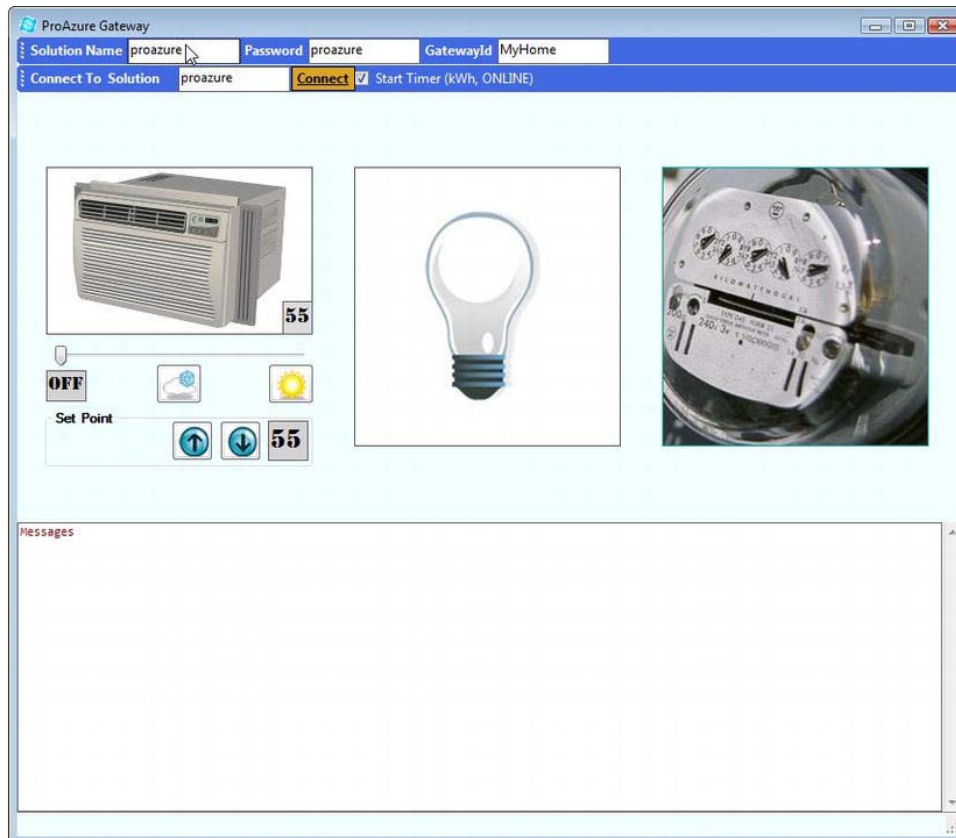


Figure 8-21. *NetEventRelayGateway design view*

The Start Time check box starts the timer to send an online message every 10 seconds.

■ **Note** I've combined the configuration of the `netOnewayRelayBinding` example and the `netEventRelayBinding` example in one project, `NetEventRelayGateway`.

Running the Application

The steps required to run the end-to-end application are as follows:

1. Open `App.config` for the `NetEventRelayGateway` and `NetEventRelayServer` and configure it to represent your service namespace and issuer credentials.
2. Open three command prompts as Administrator, and navigate two prompts to the `bin\Debug` directory of the `NetEventRelayServer` project and the third prompt to the `bin\Debug` directory of the `NetOnewayRelayServer` project. You do this because the client application also supports the `netOnewayRelayBinding` methods from the previous example.
3. Run `NetEventRelayServer.exe` in two prompts and `NetOnewayRelayServer.exe` in the third prompt.
4. Enter the solution name and solution password to start the service when prompted.
5. Open Windows Explorer, and navigate to the `bin\Debug` directory of the `NetEventRelayGateway` project.

■ **Note** Make sure the configuration for the server and the client match in terms of address and security.

6. Double-click `NetEventRelayGateway.exe` to start the client application.
7. Click the Connect button to connect to the relay service. If the connection is successful, the text box displays success messages to connect to two endpoints.
8. Check the Start Time check box if it isn't already checked.
9. If the configurations are correct, then you should see ONLINE messages in the two command windows of `NetEventRelayServer.exe`.

Thus you can build an Internet-scale publish/subscribe messaging service using `netEventRelayBinding`.

Figure 8-22 shows a running instance of the client application, and Figure 8-23 shows the messages received on the server command prompts.

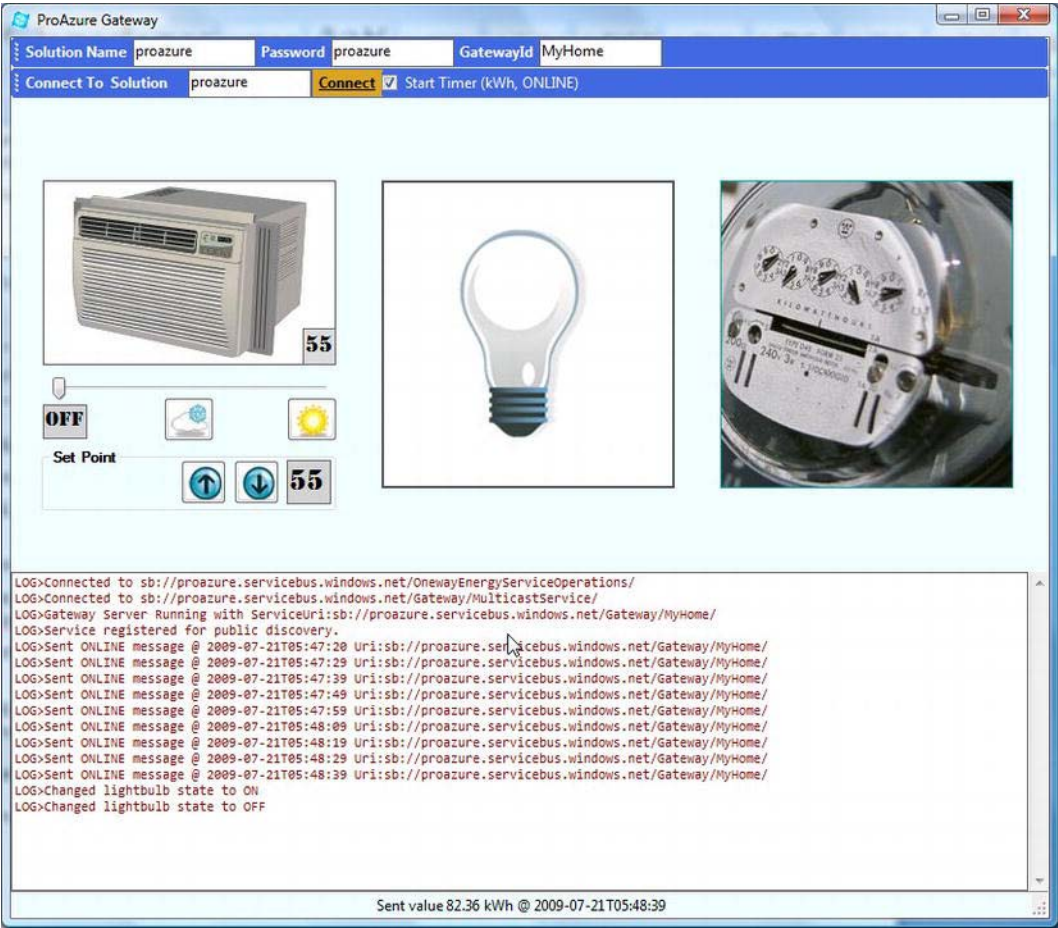
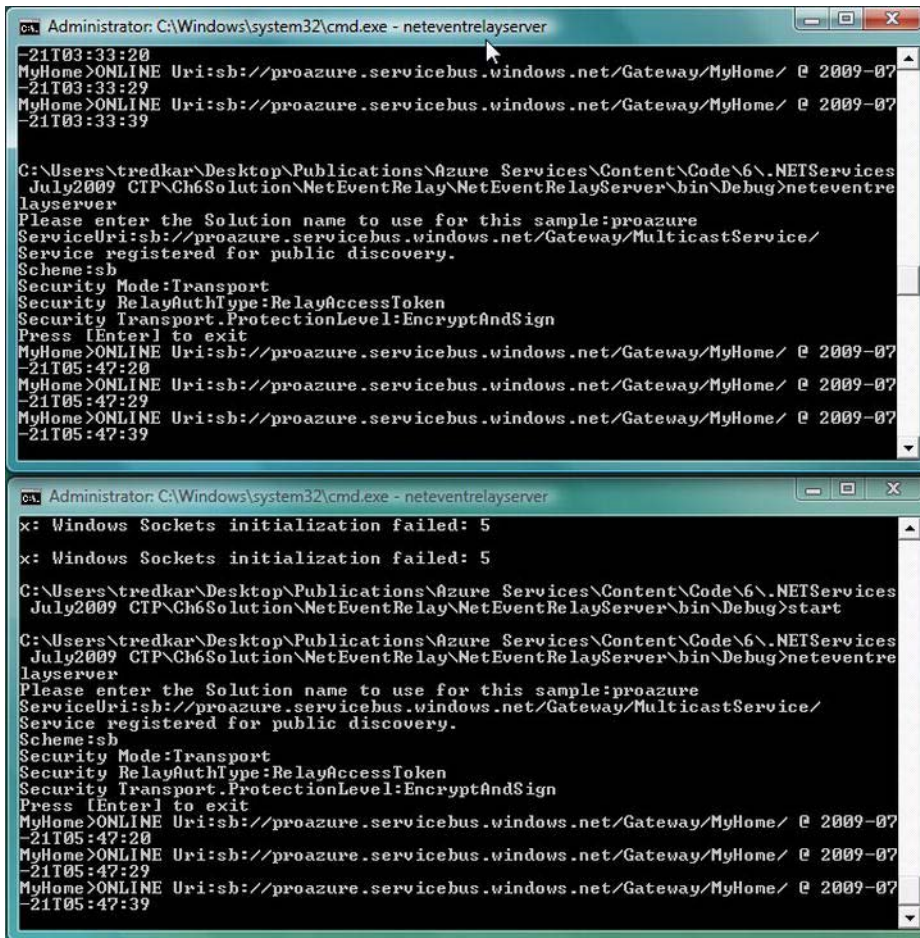


Figure 8-22. NetEventRelayGateway application



```

Administrator: C:\Windows\system32\cmd.exe - neteventrelayserver

-21T03:33:20
MyHome>ONLINE Uri:sh://proazure.servicebus.windows.net/Gateway/MyHome/ @ 2009-07
-21T03:33:29
MyHome>ONLINE Uri:sh://proazure.servicebus.windows.net/Gateway/MyHome/ @ 2009-07
-21T03:33:39

C:\Users\tredkar\Desktop\Publications\Azure Services\Content\Code\6\NETServices
July2009 CTP\Ch6Solution\NetEventRelay\NetEventRelayServer\bin\Debug>neteventre
layserver
Please enter the Solution name to use for this sample:proazure
ServiceUri:sh://proazure.servicebus.windows.net/Gateway/MulticastService/
Service registered for public discovery.
Scheme:sh
Security Mode:Transport
Security RelayAuthType:RelayAccessToken
Security Transport.ProtectionLevel:EncryptAndSign
Press [Enter] to exit
MyHome>ONLINE Uri:sh://proazure.servicebus.windows.net/Gateway/MyHome/ @ 2009-07
-21T05:47:20
MyHome>ONLINE Uri:sh://proazure.servicebus.windows.net/Gateway/MyHome/ @ 2009-07
-21T05:47:29
MyHome>ONLINE Uri:sh://proazure.servicebus.windows.net/Gateway/MyHome/ @ 2009-07
-21T05:47:39

Administrator: C:\Windows\system32\cmd.exe - neteventrelayserver

x: Windows Sockets initialization failed: 5
x: Windows Sockets initialization failed: 5

C:\Users\tredkar\Desktop\Publications\Azure Services\Content\Code\6\NETServices
July2009 CTP\Ch6Solution\NetEventRelay\NetEventRelayServer\bin\Debug>start

C:\Users\tredkar\Desktop\Publications\Azure Services\Content\Code\6\NETServices
July2009 CTP\Ch6Solution\NetEventRelay\NetEventRelayServer\bin\Debug>neteventre
layserver
Please enter the Solution name to use for this sample:proazure
ServiceUri:sh://proazure.servicebus.windows.net/Gateway/MulticastService/
Service registered for public discovery.
Scheme:sh
Security Mode:Transport
Security RelayAuthType:RelayAccessToken
Security Transport.ProtectionLevel:EncryptAndSign
Press [Enter] to exit
MyHome>ONLINE Uri:sh://proazure.servicebus.windows.net/Gateway/MyHome/ @ 2009-07
-21T05:47:20
MyHome>ONLINE Uri:sh://proazure.servicebus.windows.net/Gateway/MyHome/ @ 2009-07
-21T05:47:29
MyHome>ONLINE Uri:sh://proazure.servicebus.windows.net/Gateway/MyHome/ @ 2009-07
-21T05:47:39

```

Figure 8-23. NetEventRelayServer application

NetTcpRelayBinding

netTcpRelayBinding is the recommended and most frequently used AppFabric Service Bus binding. It uses TCP as the relay transport and is based on the WCF netTcpBinding. It performs better than the HTTP bindings, because it uses TCP for message delivery and the messages are encoded in binary format. NetTcpRelayBinding supports WS-ReliableMessaging, which is turned off by default. You can turn it on by setting reliableSessionEnabled to true. In WCF, you typically use netTcpBinding to create service endpoints reachable within the intranet, but with netTcpRelayBinding you can create service endpoints reachable over the Internet. This makes communication over the Internet faster than with HTTP bindings. Similar to netOnewayRelayBinding, netTcpRelayBinding establishes an SSL-protected control channel using outbound TCP port 828 and a non-SSL data channel using outbound TCP port 818.

■ **Note** `netTcpRelayBinding` is the only AppFabric Service Bus binding that supports WCF-style duplex callbacks through the relay service.

`netTcpRelayBinding` supports three different connection modes, as listed in Table 8-6 and defined in the AppFabric Service Bus API as the `Microsoft.ServiceBus.TcpRelayConnectionMode` enumeration.

Table 8-6. TransportClientCredentialType Values

Connection Mode	Description
Relayed (default)	In this mode, all communications between the service and the client are relayed via the AppFabric Service Bus relay service. If the message security (or security mode) is set to either <code>Transport</code> or <code>TransportWithMessageCredential</code> , the channel is SSL protected. The relay service acts a socket-forwarder proxy between the client and the service.
Direct	Direct mode is supported only through Hybrid mode. In Direct Mode, first the service and the client connects to the relay service. The relay service then upgrades the connection to direct communication between the client and the service, enabling direct communication between them. Direct mode is capable of communicating when the client and the service both are behind firewall or NAT routers. In Direct connection mode, the service requires the opening of an additional TCP outbound port 819. Communication is aborted if the client and the service aren't able to establish a direct connection. Direct mode doesn't support <code>Transport</code> security mode; you have to use the <code>Message</code> security mode.
Hybrid	Hybrid is the most commonly used mode. First, the client and the service establish an initial connection to the relay service. The client and the service then negotiate a direct connection to each to each other. The relay service monitors the negotiation and upgrades the communication to Direct mode if possible, or continues with the relayed mode. Hybrid mode doesn't support <code>Transport</code> security mode; you have to use <code>Message</code> security mode.

Figure 8-24 illustrates Relayed mode communications between a client and a service.

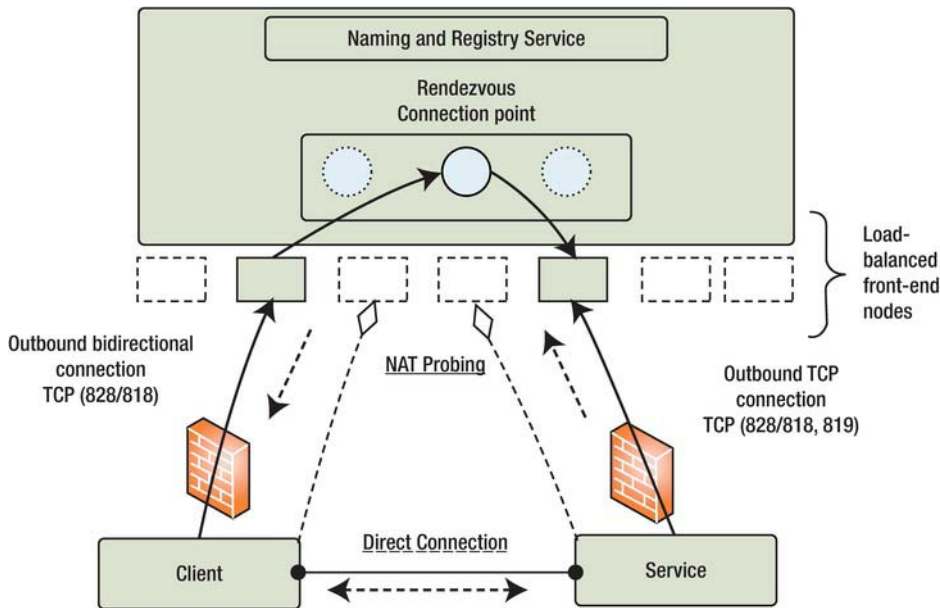


Figure 8-24. Relayed mode

Figure 8-24 shows the following:

- The client and the service first communicate through the relay service.
- Communications begin in Relayed mode.
- Direct connection negotiation between client and service succeeds.
- The relay service keeps on probing for mutual port of communication between the client and the service.
- The probing succeeds, and the relay service provides the communication information to the client and the service to communicate with each other directly.
- The connection is upgraded to a Direct connection without any data loss.
- Future communications continue in Direct mode.
- If the probing of mutual ports fails or times out, the communication continues in Relayed mode.

In the ProAzure Energy Service example, the control gateway itself is a server that accepts commands from the head-end server. An end user can schedule a command to be executed on the gateway at a particular time or execute a real-time command on the control gateway, such as turning off all the lights in the building. The control gateway accepts the command and in turn sends the command to the lighting system on the control network. The control gateway also supports real-time retrieval of device values. For example, an end user can retrieve the current state of the HVAC set point or the lighting system in real time.

AppFabric Contract

The control gateway supports get and set operations on the back-end devices it supports. In the ProAzure Energy service example, it supports get and set operations on the lighting and HVAC systems but only get operation on the energy meter. Listing 8-20 shows the service contract for the control gateway service.

Listing 8-20. Control Gateway Service Contract

```
[ServiceContract(Name = "IEnergyServiceGatewayOperations",
Namespace = "http://proazure/ServiceBus/energyservice/gateway")]
public interface IEnergyServiceGatewayOperations
{
    [OperationContract]
    bool UpdateSoftware(string softwareUrl);

    [OperationContract]
    bool SetLightingValue(string gatewayId, string deviceId,
short switchValue);

    [OperationContract]
    short GetLightingValue(string gatewayId, string deviceId);

    [OperationContract]
    bool SetHVACMode(string gatewayId, string deviceId,
int hvMode);
    [OperationContract]
    int GetHVACMode(string gatewayId, string deviceId);

    [OperationContract]
    bool SetHVACSetpoint(string gatewayId, string deviceId,
int spValue);
    [OperationContract]
    int GetHVACSetpoint(string gatewayId, string deviceId);

    [OperationContract]
    int GetCurrentTemp(string gatewayId, string deviceId);

    [OperationContract]
    double GetKWhValue(string gatewayId, string deviceId);
}

public interface IEnergyServiceGatewayOperationsChannel :
IEnergyServiceGatewayOperations, IClientChannel
```

```
{
}
```

As shown in Listing 8-20, the `IEnergyServiceGatewayOperations` support nine methods that the head-end server can call. Most of the operations are get/set methods, so the method signatures are self explanatory.

Service Implementation

The control gateway itself is the server, so the interface `IEnergyServiceGatewayOperations` is implemented in the control gateway application. The implementation of the `IEnergyServiceGatewayOperations` interface is shown in Listing 8-21.

Listing 8-21. *IEnergyServiceGatewayOperations Implementation*

```
public bool UpdateSoftware(string softwareUrl)
{
    AddLog("UpdateSoftware:" + softwareUrl);
    return true;
}

public bool SetLightingValue(string gatewayId, string deviceId,
short switchValue)
{
    ChangeLightBulbState(false, switchValue);
    AddLog("SetLightingValue:" + switchValue);
    return true;
}

public bool SetHVACMode(string gatewayId, string deviceId, int hvMode)
{
    hvacMode = hvMode;
    trackBar1.Value = hvacMode;
    ChangeHVACMode();
    AddLog("SetHVACMode:" + hvMode);
    return true;
}

public bool SetHVACSetpoint(string gatewayId, string deviceId, int spValue)
{
    ChangeSetPointValue();
    AddLog("SetHVACSetpoint:" + spValue);

    return true;
}

public short GetLightingValue(string gatewayId, string deviceId)
{
    AddLog("GetLightingValue:" + lightBulbState);
}
```

```

        return lightBulbState;
    }

    public int GetHVACMode(string gatewayId, string deviceId)
    {
        AddLog("GetHVACMode:" + hvacMode);

        return hvacMode;
    }

    public int GetHVACSetpoint(string gatewayId, string deviceId)
    {
        AddLog("GetHVACSetpoint:" + txtSetPoint.Text);
        return int.Parse(txtSetPoint.Text);
    }

    public int GetCurrentTemp(string gatewayId, string deviceId)
    {
        AddLog("GetCurrentTemp:" + txtCurrentTemperature.Text);
        return int.Parse(txtCurrentTemperature.Text);
    }

    public double GetKWhValue(string gatewayId, string deviceId)
    {
        AddLog("GetKWhValue:" + kwh);

        return kwh;
    }

```

All the method invocations are logged to the Messages text box on the control gateway application.

Service Binding

The service binding for netTcpRelayBinding is shown in Listing 8-22.

Listing 8-22. netTcpRelayBinding

```

<netTcpRelayBinding>
<binding name="default" connectionMode="Hybrid">
<security mode="None" />
</binding>
</netTcpRelayBinding>

```

Note that the connectionMode specified is Hybrid. You can specify the value as Hybrid or Relayed.

Relay Security

In the previous examples, you saw how to use different types of relay authentication. This example uses the ACS shared secret credentials to authenticate both the client and the service. Listing 8-23 shows the code from the NetEventRelayGateway project for setting the issuer and password for relay authentication.

Listing 8-23. Shared Secret Relay Authentication

```
TransportClientEndpointBehavior behavior = new TransportClientEndpointBehavior();
behavior.CredentialType = TransportClientCredentialType.SharedSecret;
behavior.Credentials.SharedSecret.IssuerName = issuerName;
behavior.Credentials.SharedSecret.IssuerSecret = issuerKey;
ServiceHost Host = new ServiceHost(serviceType);
Host.Description.Endpoints[0].Behaviors.Add(behavior);
```

■ **Note** The NetEventRelayGateway project implements the service contract because the control gateway itself is the server now and the head-end server is the client. Because the server instance implements the interface, you have to set the instance context mode to single, as shown here:

```
[ServiceBehavior(Name = "EnergyServiceGatewayOperations",
    Namespace = "http://proazure/ServiceBus/energyservice/gateway",
    InstanceContextMode=InstanceContextMode.Single)]
public partial class EnergyManagementDevice : Form, IEnergyServiceGatewayOperations
```

Message Security

The netTcpRelayBinding uses Transport as its default message security if you don't explicitly configure it in App.config. This example doesn't use message security, to keep the example simple. Listing 8-24 shows the configuration of netTcpRelayBinding in the App.config file of the server in the NetEventRelayGateway project.

Listing 8-24. Message Security in netTcpRelayBinding

```
<netTcpRelayBinding>
<binding name="default" connectionMode="Hybrid">
<security mode="None" />
</binding>
</netTcpRelayBinding>
```

Service Endpoints

The service endpoint configuration is shown in Listing 8-25.

Listing 8-25. Service Endpoint Configuration

```
<services>
<service name="NetEventRelayGateway.EnergyManagementDevice">
<endpoint name="RelayTcpEndpoint"
contract="EnergyServiceContract.IEnergyServiceGatewayOperations"
binding="netTcpRelayBinding"
bindingConfiguration="default"
address="" />
</service>
```

Note that in the endpoint configuration, the address field is empty: the address is generated at runtime so you can run multiple instances of the same application representing difference control gateways. Each control gateway has its own service endpoint, which the head-end server accesses to call methods on each control device.

Service Hosting

The service is hosted in the control gateway, so NetEventRelayGateway contains the code to host the service. Listing 8-26 shows the code that hosts the service within the NetEventRelayGateway application.

Listing 8-26. Service Hosting

```
Uri address = ServiceBusEnvironment.CreateServiceUri("sb", solutionName, servicePath);
ServiceUri = address.ToString();
TransportClientEndpointBehavior behavior = new TransportClientEndpointBehavior();
behavior.CredentialType = TransportClientCredentialType.SharedSecret;
behavior.Credentials.SharedSecret.IssuerName = issuerName;
behavior.Credentials.SharedSecret.IssuerSecret = issuerKey; Host = new ServiceHost(serviceType,
address);
Host.Description.Endpoints[0].Behaviors.Add(behavior);
Host.Open();
```

In Listing 8-26, the URI for the service is generated dynamically by calling the `ServiceBusEnvironment.CreateServiceUri()` method. The `servicePath` contains the `gatewayID`, which makes the URI unique within the network of all the control gateways. The head-end server uses this URI to call methods on the control gateway.

Client Design

The head-end server acts as a client for all the control gateways. The client in this example is a simple console application that accepts a gateway ID, then creates the endpoint URI programmatically, and finally invokes multiple methods to turn off all the devices attached to the control gateway. The source

code for the client application is in the NetTcpRelayBinding project. Listing 8-27 shows the code for the method (without exception handling) that turns off all the devices attached to the control gateway.

Listing 8-27. *TurnEverythingOff Source Code*

```
static void TurnEverythingOff(string solutionName, string password,
    string gatewayId)
{
    ChannelFactory<IEnergyServiceGatewayOperationsChannel>
netTcpRelayChannelFactory = null;
    IEnergyServiceGatewayOperationsChannel
netTcpRelayChannel = null;

    Uri serviceUri = ServiceBusEnvironment.CreateServiceUri("sb",
solutionName, ServiceBusHelper.GetGatewayServicePath(gatewayId));
    netTcpRelayChannelFactory = new
ChannelFactory<IEnergyServiceGatewayOperationsChannel>
("RelayTcpEndpoint", new EndpointAddress(serviceUri));

    netTcpRelayChannel = netTcpRelayChannelFactory.CreateChannel();
    netTcpRelayChannel.Open();
    netTcpRelayChannel.SetLightingValue(gatewayId, "Lighting-1", 0);

    netTcpRelayChannel.SetHVACMode(gatewayId, "HVAC-1", 0);
    netTcpRelayChannel.SetHVACSetpoint(gatewayId, "HVAC-1", 78);

    netTcpRelayChannel.Close();
    netTcpRelayChannelFactory.Close();
}
```

In Listing 8-27, a channel is created with the endpoint URI based on the gateway identifier. Then, the SetLightingValue(), SetHVACMode(), and SetHVACSetpoint() methods are called on the control gateway to turn off the devices attached to the control gateway. Because the URI is generated dynamically from the gateway identified, you can invoke these methods on any gateway that has an endpoint URI registered with the AppFabric Service Bus. The ACS shared secret is defined in App.config, and therefore you don't need to redefine it in the code. Listing 8-28 shows the definition of the shared secret in App.config of the client project NetTcpRelayBinding.

Listing 8-28. *Shared Secret Definition in the Client*

```
<behaviors>
  <endpointBehaviors>
    <behavior name="sharedSecretClientCredentials">
      <transportClientEndpointBehavior credentialType="SharedSecret">
        <clientCredentials>
          <sharedSecret issuerName="ISSUER_NAME" issuerSecret="ISSUER_KEY" />
        </clientCredentials>
      </transportClientEndpointBehavior>
    </behavior>
  </endpointBehaviors>
</behaviors>
```

Running the Application

The steps required to run the end-to-end application are as follows:

1. Open Windows Explorer, and navigate to the bin\Debug directory of the NetEventRelayGateway project.
2. Double-click NetEventRelayGateway.exe two times to start two instances of the NetEventRelayGateway application.
3. Change the service namespace name, issuer name, and issuer key to your own values.
4. In the GatewayId field of the first application, enter **MyOffice**. Leave the default MyHome in the second application.
5. Click the Connect button on both the instances of NetEventRelayGateway to connect to the relay service. If the connections are successful, the text boxes display success messages with the URIs of the service endpoints. Note how the URIs are created based on the gateway identifier to make them unique.
6. Turn the light switch on, and turn the HVAC mode to HEAT or COOL.
7. Open a command prompt window with Administrator privileges, and navigate to the bin\Debug directory of the NetTcpRelayBinding project.
8. Start the NetTcpRelayBinding.exe console application.
9. Enter the service namespace name when prompted.
10. When prompted, enter the gateway ID MyHome.
11. Observe in the NetEventRelayGateway application that the light switch and the HVAC mode are turned off.
12. Perform the same operation on the gateway ID MyOffice to see similar results

Thus, you can dynamically register thousands of control gateway endpoints with the AppFabric Service Bus and execute methods on these control gateways at Internet scale

Figure 8-25 illustrates the two running instances of NetEventRelayGateway.exe, and Figure 8-26 illustrates the NetTcpRelayBinding.exe command prompt.



Figure 8-25. NetEventRelayGateway application


```

file:///C:/Users/tredkar/Desktop/Publications/Azure Services/Content/Code/6/.NETServices July2009...
Please enter the Solution name to use for this sample: proazure
Your Solution Password: *****
Press enter exit to exit
Press enter a gatewayId to turn everything off
Enter GatewayID>MyHome
Connected to sb://proazure.servicebus.windows.net/Gateway/MyHome/
Light switch is:0
Light switch turned OFF
Current Temperature:55
Current Set Point:55
Current HVAC Mode:0
Set HVAC mode to OFF
Set everything to off on MyHome
Press enter exit to exit
Press enter a gatewayId to turn everything off
Enter GatewayID>MyHome
Connected to sb://proazure.servicebus.windows.net/Gateway/MyHome/
Light switch is:1
Light switch turned OFF
Current Temperature:55
Current Set Point:55
Current HVAC Mode:2
Set HVAC mode to OFF
Set everything to off on MyHome
Press enter exit to exit

```

Figure 8-26. *NetTcpRelayBinding application*

You can catch the connection upgrade event when a Relayed connection is upgraded to a Direct connection by implementing the `ConnectionStateChanged` event on the `IMybridConnectionStatus` interface, as shown in Listing 8-29.

Listing 8-29. *Connection Upgrade Event*

```

IMybridConnectionStatus hybridConnectionStatus =
channel.GetProperty<IMybridConnectionStatus>();
if (hybridConnectionStatus != null)
{
    hybridConnectionStatus.ConnectionStateChanged += (o, e) =>
    {
        //Do work
    };
}

```

HTTP Relay Bindings

As discussed in Table 8-3, the AppFabric Service Bus supports the following HTTP relay bindings:

- `BasicHttpRelayBinding`
- `WebHttpRelayBinding`
- `WSHttpRelayBinding`
- `WS2007HttpRelayBinding`

This section covers only `WS2007HttpRelayBinding` and `WebHttpRelayBinding` because the concepts for using all these bindings are similar. When you use HTTP bindings, the AppFabric Service Bus uses HTTP as the communication protocol instead of TCP as you saw earlier in the `netOnewayRelayBinding`

and `netTcpRelayBinding` sections. HTTP bindings exchange plain XML, SOAP, WS-*, or raw text and binary messages, so they're preferred in non-WCF client environments.

At a higher level, all the HTTP bindings follow the same sequence of steps to communicate via the relay service, as shown in Figure 8-27.

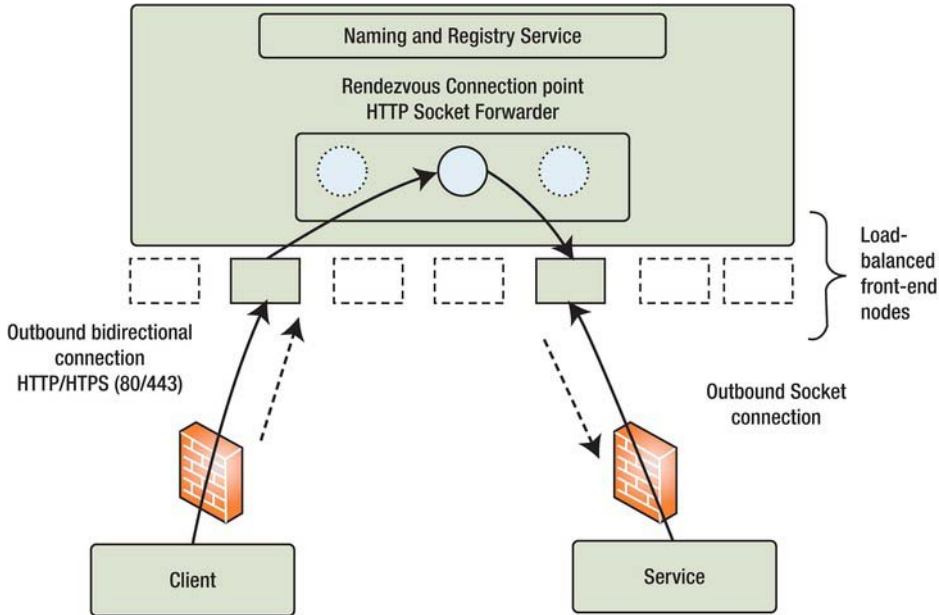


Figure 8-27. HTTP bindings

As shown in Figure 8-27, in an HTTP binding scenario, the service authenticates and registers its endpoint with the relay service. Then, a client authenticates and connects to the relay service to call a method on the service. The relay service routes the HTTP (REST), SOAP 1.1, and SOAP 1.2 calls to the service. Your business logic in the code doesn't change depending on the binding you use. As you saw earlier, you can configure bindings in the configuration file.

WS2007HttpRelayBinding

`WS2007HttpRelayBinding` supports SOAP 1.2 messaging with the latest OASIS standards for reliable message exchange and security. It's used to create SOAP over HTTP interfaces for your service. To demonstrate `WS2007HttpRelayBinding`, you use the same control gateway applications as the service, and the head-end server as the client application as you saw for `netTcpRelayBinding`. By modifying a few lines of code, you can easily convert `netTcpRelayBinding` to `ws2007HttpRelayBinding`.

The binding and service configuration for `WS2007HttpRelayBinding` is shown in Listing 8-30.

Listing 8-30. WS2007HttpRelay Configuration

```

<!--Define the binding -->
<ws2007HttpRelayBinding>
<binding name="default">
<security mode="None" relayClientAuthenticationType="None" />
</binding>
</ws2007HttpRelayBinding>

<!--Define end point -->
<endpoint name="RelayTcpEndpoint"
        contract="EnergyServiceContract.IEnergyServiceGatewayOperations"
        binding="ws2007HttpRelayBinding"
        bindingConfiguration="default"
        address="" />

```

The only difference between `netTcpRelayConfiguration` and `ws2007HttpRelayConfiguration` is the definition of the binding and replacing `netTcpRelayBinding` with `ws2007HttpRelayBinding`. Similarly, in the client application, you can make replacements as shown in Listing 8-31.

Listing 8-31. WS2007HttpRelayBinding Configuration

```

<!--Define the binding -->
<bindings>
<ws2007HttpRelayBinding>
  <binding name="default">
    <security mode="None"/>
  </binding>
</ws2007HttpRelayBinding>
</bindings>
<!--Define end point -->
<client>
<endpoint
name="RelayTcpEndpoint"
contract="EnergyServiceContract.IEnergyServiceGatewayOperations"
binding="ws2007HttpRelayBinding "
bindingConfiguration="default"
behaviorConfiguration="sharedSecretClientCredentials"
address="http://AddressToBeReplacedInCode/" />
</client>

```

The `WS2007HttpRelayBinding` client application authenticates itself with the AppFabric Service Bus using the ACS shared-secret authentication method. In the code, when you generate the URI in both client and the server, you must replace the “sb” protocol from `netTcpRelayBinding` to “http” for `ws2007HttpRelayBinding`:

```
Uri serviceUri = ServiceBusEnvironment.CreateServiceUri("http", serviceNamespace,
ServiceBusHelper.GetGatewayServicePath(gatewayId));
```

The steps required to run the end-to-end application are the same as running the `netTcpRelayBinding` example in the previous section. Figure 8-28 shows the client and service applications using `WS2007HttpRelayBinding`.

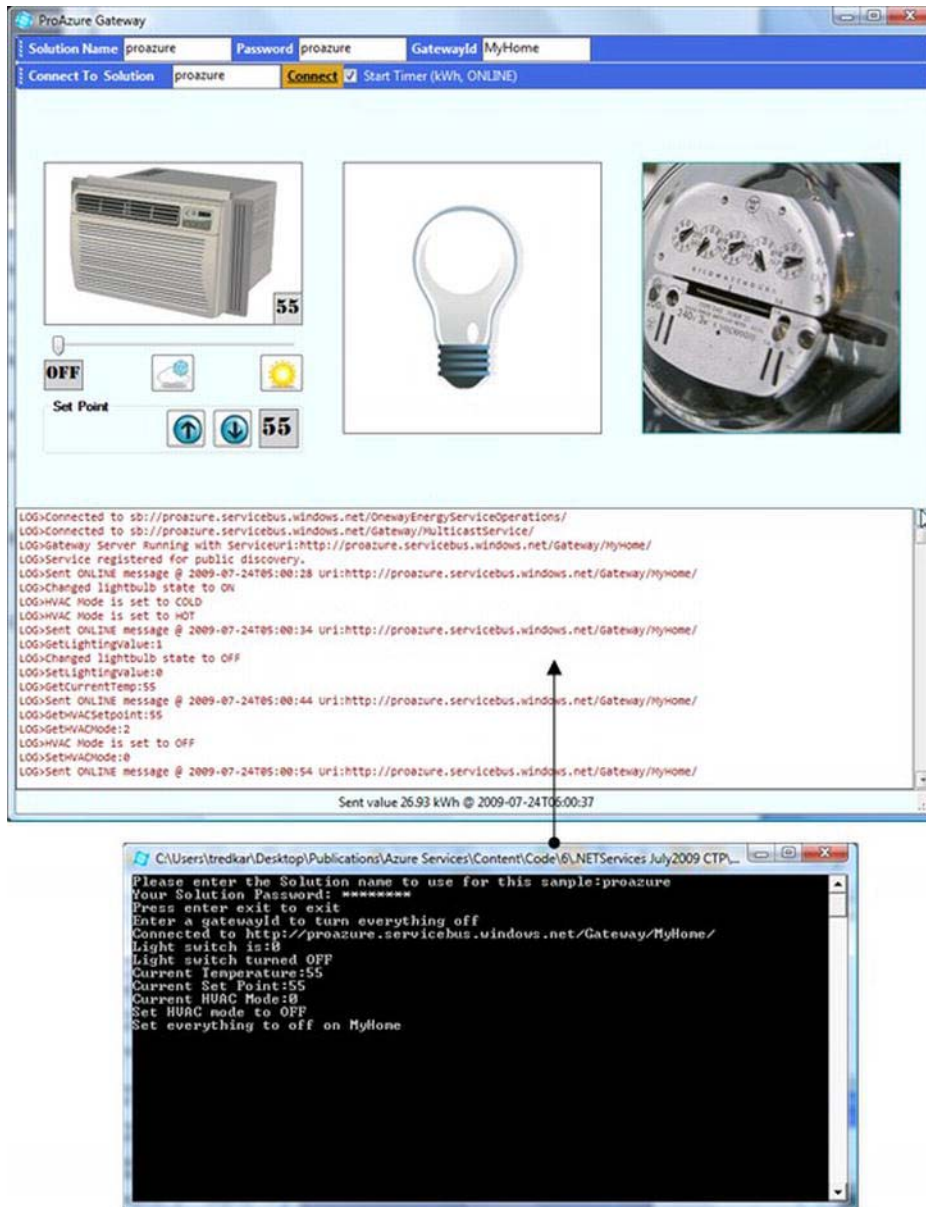


Figure 8-28. WS2007HttpRelayBinding client and service applications

While running the application, note the delay when using the `WS2007HttpRelayBinding` as compared to the `netTcpRelayBinding`. `WS2007HttpRelayBinding` polls the relay service for the message.

WebHttpRelayBinding

In the past few years, REST-style programming has become popular because it uses existing HTTP constructs to communicate messages and remote method invocations. As compared to SOAP, the REST interface is easier to use in manual and scripting interfaces. In the Windows Azure Storage chapters, you learned to use the REST interface exposed by the storage service to interact with storage objects like blobs, queues, and tables. `WebHttpRelayBinding` is used to create HTTP, XML, and REST-style interfaces for your service.

To demonstrate `WebHttpRelayBinding`, you create a simple service contract that represents a REST-style interface over the control gateway service. You can find the example for `WebHttpRelayBinding` in the project `RESTGatewayServer` in `Ch8Solution`.

Listing 8-32 shows the code representing two contracts: one for the lighting service (`IRESTLightswitch`) and the other (`IRESTEnergyMeter`) for the energy meter service.

Listing 8-32. *Lighting Service and Energy Meter Contracts*

```
namespace EnergyServiceContract
{
    [ServiceContract(Name = "IRESTLightswitch.",
        Namespace = "http://proazure/ServiceBus/energyservice/gateway")]
    public interface IRESTLightswitch
    {
        [OperationContract(Action = "GET", ReplyAction = "GETRESPONSE")]
        Message GetLightswitchState();
    }
    public interface IRESTLightswitchChannel : IRESTLightswitch, IClientChannel
    {
    }

    [ServiceContract(Name = "IRESTEnergyMeter.",
        Namespace = "http://proazure/ServiceBus/energyservice/gateway")]
    public interface IRESTEnergyMeter
    {
        [OperationContract(Action = "GET", ReplyAction = "GETRESPONSE")]
        Message GetKWhValue();
    }
}
```

You can combine both interfaces into one, but this example ties the simple HTTP GET operation to each method. The `OperationContract.Action` attribute property represents the HTTP action used to call this operation. This name must be unique within an interface. The `System.ServiceModel.Channels.Message` return type is a generic type of object to communicate information between the client and the service.

Listing 8-33 contains the implementation of both the service contracts.

Listing 8-33. Service Implementation

```

public class GatewayService : IRESTLightswitch, IRESTEnergyMeter
{
    const string ON_FILE = "on.jpg";
    const string OFF_FILE = "off.jpg";
    Image on, off;
    static int LIGHT_BULB_STATE = 0;
    public GatewayService()
    {
        on = Image.FromFile(ON_FILE);
        off = Image.FromFile(OFF_FILE);
    }
    public Message GetLightswitchState()
    {
        Message m = Message.CreateMessage
(OperationContext.Current.IncomingMessageVersion, "GETRESPONSE", "ON");
        return m;
    }
    System.ServiceModel.Channels.Message IRESTLightswitch.GetLightswitchState()
    {
        Message response = StreamMessageHelper.CreateMessage
(OperationContext.Current.IncomingMessageVersion,
"GETRESPONSE", this.WriteImageToStream);
        HttpResponseMessageProperty responseProperty =
new HttpResponseMessageProperty();
        responseProperty.Headers.Add("Content-Type", "image/jpeg");
        response.Properties.Add(HttpResponseMessageProperty.Name,
responseProperty);
        return response;
    }
    public void WriteImageToStream(System.IO.Stream stream)
    {
        Image i = (LIGHT_BULB_STATE == 0) ? off : on;
        i.Save(stream, ImageFormat.Jpeg);
        if (LIGHT_BULB_STATE == 0)
        {
            LIGHT_BULB_STATE = 1;
        }
        else
        {
            LIGHT_BULB_STATE = 0;
        }
    }
    System.ServiceModel.Channels.Message IRESTEnergyMeter.GetKWhValue()
    {
        Random r = new Random();
        double kwhValue = double.Parse
(String.Format("{0:0.00}", (r.NextDouble() * 100)));
        System.ServiceModel.Channels.Message m =Message.CreateMessage
(OperationContext.Current.IncomingMessageVersion, "GETRESPONSE",

```

```
String.Format("{0:00}", kwhValue));
    return m;
}
}
```

In Listing 8-33, the `GatewayService` class implements the `IRESTLightswitch` and `IRESTEnergyMeter` interfaces. The implementation of the methods is very simple because they're only simulating the call and not making any real calls to the devices. The `GetLightswitchState()` method returns an image representing the state of the lighting service. The `GetKWhValue()` method returns a text value representing a randomly generated kWh value. Note the use of the `System.ServiceModel.Channels.Message` object to transfer an image as well as a text value.

Because you can access the REST interface manually from the browser, you don't implement a client for the service. Listing 8-34 shows the configuration for the service.

Listing 8-34. Service Configuration

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <!-- Application Binding -->
      <webHttpRelayBinding>
        <binding name="default" >

          <security
            relayClientAuthenticationType="None" />

        </binding>
      </webHttpRelayBinding>
    </bindings>

    <services>
      <!-- Application Service -->
      <service name="RESTGatewayServer.GatewayService"
        behaviorConfiguration="default">
        <endpoint name="LighswitchEndpoint"
          contract="EnergyServiceContract.IRESTLightswitch"
          binding="webHttpRelayBinding"
          bindingConfiguration="default"
          behaviorConfiguration="cardSpaceClientCredentials"
          address=
            "https://{your service namespace}.servicebus.windows.net/Gateway/MyHome/Lightswitch" />
        <endpoint name="EnergyMeterEndpoint"
          contract="EnergyServiceContract.IRESTEnergyMeter"
          binding="webHttpRelayBinding"
          bindingConfiguration="default"
          behaviorConfiguration="cardSpaceClientCredentials"
          address=
            "https://{your service namespace}.servicebus.windows.net/Gateway/MyHome/Meter" />
        </service>
      </services>
    </behaviors>
```

```

<endpointBehaviors>
  <behavior name="sharedSecretClientCredentials">
    <transportClientEndpointBehavior credentialType="SharedSecret">
      <clientCredentials>
        <sharedSecret issuerName="owner"
issuerSecret="wJBJaobUmarWn6kqv7QpaaRh3ttNVr3w10jiotVEOL4=" />
      </clientCredentials>
    </transportClientEndpointBehavior>
  </behavior>
</endpointBehaviors>
<serviceBehaviors>
  <behavior name="default">
    <serviceDebug httpHelpPageEnabled="false" httpsHelpPageEnabled="false" />
  </behavior>
</serviceBehaviors>
</behaviors> </system.serviceModel>
</configuration>

```

In Listing 8-34, the service is configured to use a shared secret to authenticate with the AppFabric Service Bus. The `relayAuthenticationType=None` value disables the user authentication so that users can access the service without authenticating themselves. You can start the service, and users should be able to access it through the browser.

The steps to run the `RESTGatewayServer` application are as follows:

1. Configure the service with your service namespace and shared secret information.
2. Open a command prompt as Administrator, and navigate to the `bin\Debug` folder of the `RESTGatewayServer` project.
3. Run `RESTGatewayServer.exe`.
4. When the service starts, it displays URIs for the `Lightswitch` and `EnergyMeter` endpoints. Write down the URI access points of `Lightswitch` and `EnergyMeter`, as shown in Figure 8-29.

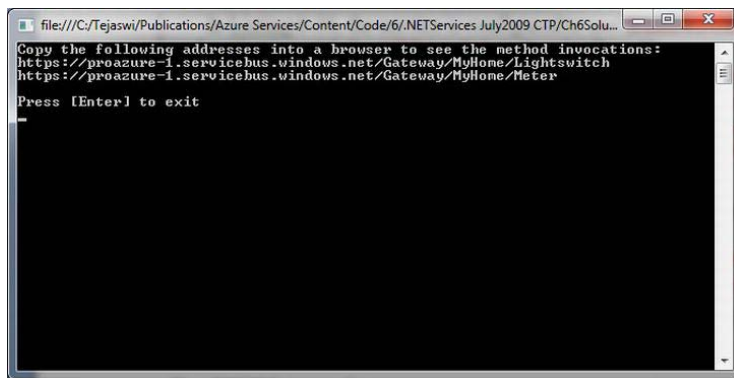


Figure 8-29. Access URLs

- 5. Open a browser, and navigate to each endpoint. The method is automatically invoked, and the result is displayed in the browser as shown in Figures 8-30 and 8-31. Figure 8-30 illustrates the light switch state, and Figure 8-31 illustrates the energy meter value.

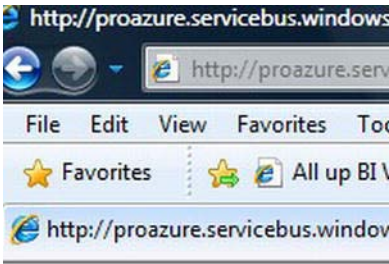
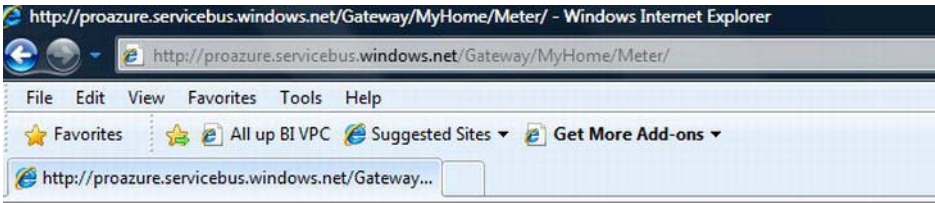


Figure 8-30. Light switch state



<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">26</string>

Figure 8-31. Energy meter value

6. You can also go to the AtomPub feed of the service to invoke methods. Navigate to the solution feed page [http://\[solution name\].servicebus.windows.net/](http://[solution name].servicebus.windows.net/), as shown in Figure 8-32.

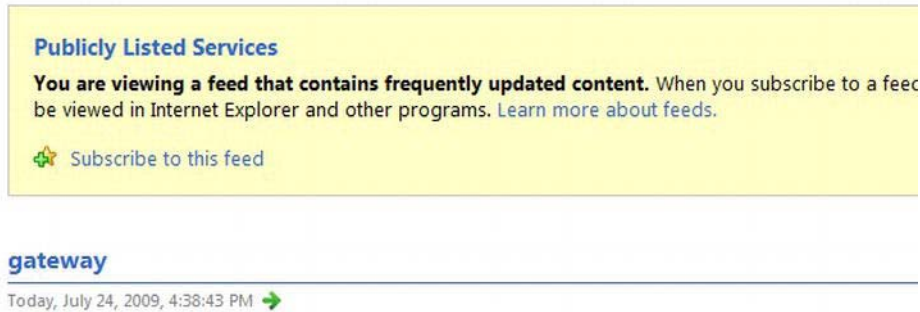


Figure 8-32. Solution feed

7. Click the gateway to go to the list of registered gateways feeds, as shown in Figure 8-33.

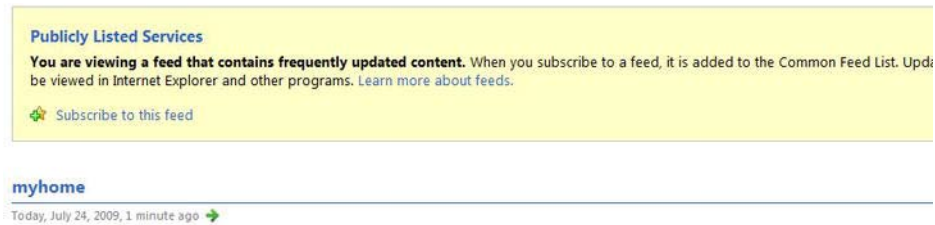


Figure 8-33. Registered gateways

8. Click the gateway (myhome) to go to the gateway operations feed page, as shown in Figure 8-34.



Figure 8-34. Gateway operations

9. Click any of the listed operations to invoke the remote method and see the response in the browser.

Message Buffer

A *message buffer* is a temporary cache you can create in the AppFabric Service Bus. I call it a temporary cache because the data in the cache isn't persistent and can't survive server reboots. Therefore, I recommend that you store only temporary data in message buffers and assume data loss while programming your applications.

■ **Note** At the time of writing, AppFabric Service Bus version 2 had just been released. Therefore, we will cover both Message Buffers and the version 2 replacement, Queues and Topics. Bear in mind that Message Buffers are intended to be deprecated in future releases, and are only supported for backward compatibility going forward.

A message buffer exposes operations through a REST API that you can use to create message buffers and execute CRUD operations on messages. The message buffer REST API integrates with ACS, and therefore you can share the authentication you use for other Service Bus application with the message buffer. Figure 8-35 illustrates the high-level architecture of the message buffer.

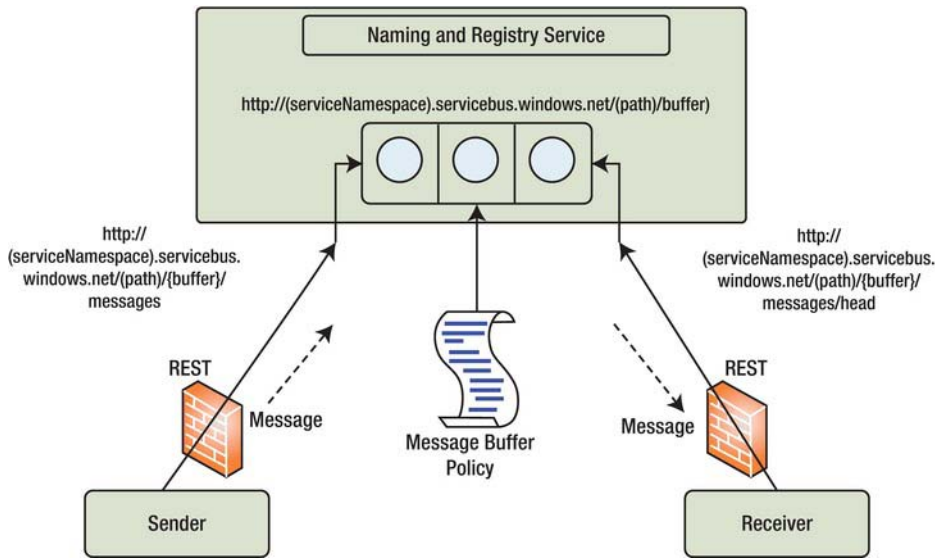


Figure 8-35. Message buffer architecture

As illustrated in Figure 8-35, the message buffer has three main components: a message buffer, a message buffer policy, and the message. The message buffer represents the actual buffer you use to store messages. The message buffer policy represents certain attributes of the message buffer such as the buffer lifetime, maximum message count, and message overflow policy. The message represents the message you send and receive from a message buffer.

The typical developer workflow in a message buffer application is as follows:

1. Create a message buffer policy.
2. Create a message buffer.
3. Send messages to the message buffer.
4. Receive or peek messages from the message buffer.
5. Delete messages.
6. Delete the message buffer.

The Service Bus SDK also provides a `MessageBufferClient` class in the `Microsoft.ServiceBus.dll` assembly for interacting with the message buffer. Figure 8-36 shows the class diagram of the `MessageBufferClient` and `MessageBufferPolicy` classes.

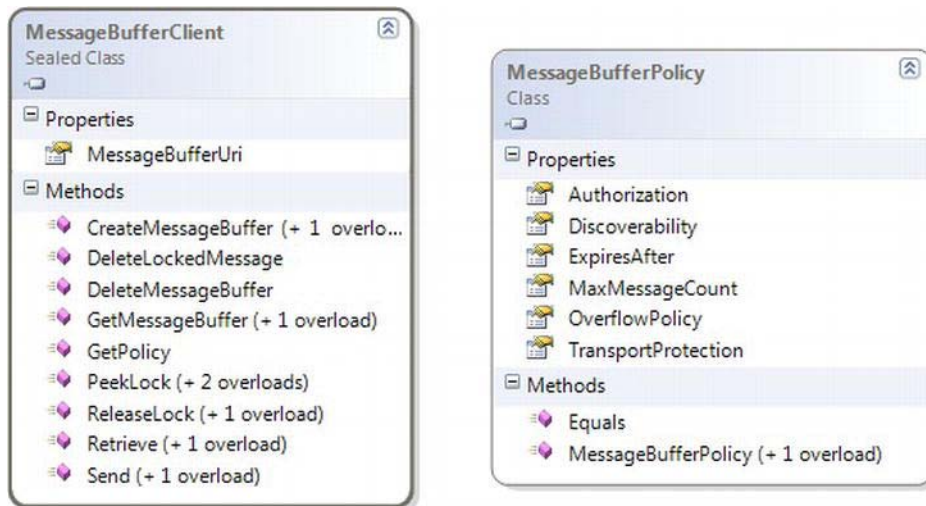


Figure 8-36. *MessageBufferClient and MessageBufferPolicy class diagrams*

As shown in Figure 8-35, the `MessageBufferClient` class includes all the basic operations like `CreateMessageBuffer()`, `DeleteMessageBuffer()`, `Retrieve()`, `Send()`, and `PeekLock()` for interacting with the message buffer. The `MessageBufferClient` class abstracts the REST interface. You can call the `MessageBufferClient` methods from your code directly; the `MessageBufferClient` class translates the method invocations into REST API calls to the message buffer.

■ **Note** To learn more about the message buffer REST API methods, visit the AppFabric SDK at <http://msdn.microsoft.com/en-us/library/ee794877.aspx>.

Programming Message Buffer Applications

Because of the REST API, a message buffer is available to any programming language, cross platform. You can write message buffer applications in any programming language that can make remote HTTP calls. This section goes over the typical developer operations on the message buffer using the `MessageBufferClient` class in the C# language.

Creating a Message Buffer Policy

A message buffer policy represents the runtime attributes of a message buffer. The policy is applied to a message buffer during its creation time. A message buffer policy is represented by the `MessageBufferPolicy` class, which is passed to the `MessageBufferClient.CreateMessageBuffer()` method. Listing 8-35 shows the code to create an instance of the `MessageBufferPolicy` class.

Listing 8-35. Initialize MessageBufferPolicy

```
private static MessageBufferPolicy GetMessageBufferPolicy(double bufferExpirationTime,
int maxMessageCount)
{
    MessageBufferPolicy policy = new MessageBufferPolicy
    {
        ExpiresAfter = TimeSpan.FromMinutes(bufferExpirationTime),
        MaxMessageCount = maxMessageCount,
        OverflowPolicy = OverflowPolicy.RejectIncomingMessage,
        Authorization = AuthorizationPolicy.NotRequired,
        Discoverability = DiscoverabilityPolicy.Public,
        TransportProtection = TransportProtectionPolicy.AllPaths
    };
    return policy;
}
```

In Listing 8-35, `ExpiresAfter` sets the lifetime of the message buffer. The lifetime of the message buffer is automatically renewed when you send a message to the buffer. `MaxMessageCount` represents the message capacity of the message buffer. `OverflowPolicy` represents the policy to be applied if there is a message overflow beyond the capacity of the message buffer. As of the September 2011 release of the Service Bus API, the only overflow policy available was `OverflowPolicy.RejectIncomingMessage`. `AuthorizationPolicy` represents the authorization policy required to access the message buffer. The default policy is `AuthorizationPolicy.Required`, which means that authorization is required to send as well as receiving messages. `Discoverability` determines whether the message buffer is accessible from the AppFabric Atom Feed. If `Discoverability` isn't set to `Public`, then applications must know the explicit URI of the message buffer. The default `Discoverability` value is `Managers`, which means only the application that created the message buffer has access to it. `TransportProtection` represents the end-to-end security of the message that traverses from sender to the receiver.

Creating and Deleting a Message Buffer

When you've created the message buffer policy, you can create the message buffer by calling the `MessageBufferClient.CreateMessageBuffer()` method, as shown in Listing 8-36.

Listing 8-36. Create Message Buffer

```
private MessageBufferClient CreateMessageBuffer(
string serviceNamespace, string messageBufferName, TransportClientEndpointBehavior behavior,
MessageBufferPolicy policy)
{
    MessageVersion messageVersion = MessageVersion.Default;
    Uri messageBufferUri = ServiceBusEnvironment.CreateServiceUri
("https", serviceNamespace, messageBufferName);
    return MessageBufferClient.CreateMessageBuffer(behavior, messageBufferUri, policy,
messageVersion);
}
```

Before you create a message buffer, you have to create an URI for the message buffer endpoint. You can create only one message buffer per endpoint, and when the endpoint is reserved for the message buffer, you can't register any other service on that endpoint. After the message buffer is created, you can get a reference to a message buffer (`MessageBufferClient` object) by calling the method

```
MessageBufferClient client =
MessageBufferClient.GetMessageBuffer(TransportClientEndpointBehavior behavior, Uri
messageBufferUri)
```

You can delete a message buffer by calling the method `MessageBufferClient.DeleteMessageBuffer()`.

Sending Messages to a Message Buffer

To send messages to the message buffer, you can call the `Send()` method on the message buffer `Client` object that was returned either when the message buffer was created or when you called the `GetMessageBuffer()` method. Listing 8-37 shows the method call to send messages to a message buffer.

Listing 8-37. Sending Messages to a Message Buffer

```
private void SendMessage(string message, MessageBufferClient client)
{
    System.ServiceModel.Channels.Message msg =
    System.ServiceModel.Channels.Message.CreateMessage(
        MessageVersion.Default,
        string.Empty,
        message);
    client.Send(msg, TimeSpan.FromSeconds(30));
    msg.Close();
}
```

The `Send()` method accepts a `System.ServiceModel.Channels.Message` object and optionally accepts a method execution timeout value. This is the time the method call should wait before timing out.

Retrieving Message from a Message Buffer

The message buffer client API provides two main methods for retrieving a message from the message buffer: `PeekLock()` and `Retrieve()`. The `PeekLock()` method is used to peek at the first message in the message buffer by locking the message before the buffer is instructed to release or delete the message. The `PeekLock()` method also provides overloads for specifying the method timeout to wait on message and the duration for which the message remains locked. You can lock a message for a duration between 10 seconds and 5 minutes, the default being 2 minutes. You can call the `DeleteLockedMessage()` or `ReleaseLock()` method to release a lock on the message.

The `Retrieve()` method retrieves the message from the message buffer and deletes the message from the message buffer. This kind of read is also called a *destructive read* and is the recommended method for high-performance applications to avoid round trips to the server. Listing 8-38 shows the code for retrieving messages from the message buffer.

Listing 8-38. Retrieving Messages from a Message Buffer

```

private string RetrieveMessage(MessageBufferClient client)
{
    System.ServiceModel.Channels.Message retrievedMessage;

    retrievedMessage = client.Retrieve();
    retrievedMessage.Close();

    return retrievedMessage.GetBody<string>();
}

private string PeekMessage(MessageBufferClient client)
{
    System.ServiceModel.Channels.Message lockedMessage = client.PeekLock();
    client.DeleteLockedMessage(lockedMessage);
    lockedMessage.Close();

    return lockedMessage.GetBody<string>();
}

```

Message Buffer Sample Application

I've created a message buffer sample application in the source code solution for this chapter. The MessageBuffer project in the chapter solution is a Windows application that creates a message buffer, sends messages to the message buffer, retrieves messages from the message buffer, and finally deletes the message buffer. Figure 8-37 shows the application in action.

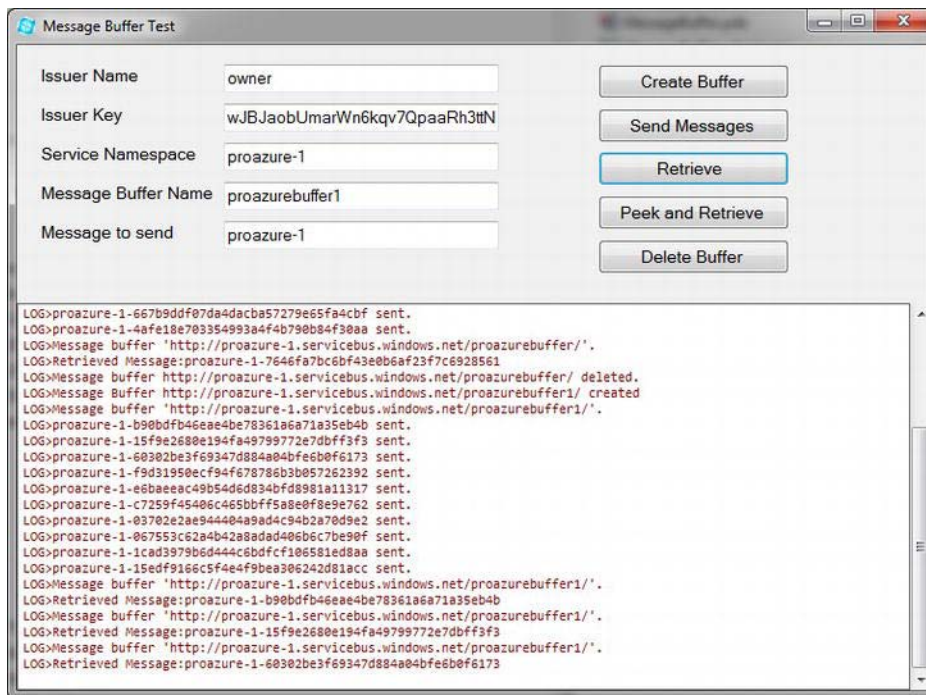


Figure 8-37. Message buffer sample application

In the sample application, you can enter your own issuer credentials and service namespace and start interacting with the message buffer. From the application, you can create a message buffer, send messages, retrieve messages, peek and retrieve messages, and finally delete the message buffer. In this case, the message sender and the message receiver are the same application; but you can separate the message sender and message receiver functionality into different applications because the message buffer API is stateless and so the same instance of the message buffer is accessible to all authenticated applications.

AppFabric Messaging: Queues and Topics

While the AppFabric Service Bus provided a viable ESB solution, there were some limitations. Most notably, the Message Buffer is only a temporary cache, and cannot survive server reboots. Some application architects worked around this by using Windows Azure Storage Queues. However, this did not permit the extra functionality provided by the Service Bus, such as event notification patterns, multicasting, and support for protocols other than HTTP/S.

Version 2 of the Service Bus provides the best of both worlds. The focus of this release is to enable rich messaging scenarios, such as publish/subscribe, temporal decoupling, and load balancing scenarios at Internet scale.⁶ AppFabric Service Bus Queues provides a persistent store for messages, and Topics

⁶ MSDN documentation: <http://msdn.microsoft.com/en-us/library/hh201962.aspx>

enable the ability to distribute messages to multiple consumers using simple rules and a publish/subscribe pattern.

■ **Note** Another great source of information on this topic can be found at <http://blogs.msdn.com/b/windowsazure/archive/2011/11/11/new-article-managing-and-testing-topics-queues-and-relay-services-with-the-service-bus-explorer-tool.aspx>. This blog post covers the Service Bus Explorer Tool, which allows you to administer your messaging entities, but also has links to many other background subjects as well.

AppFabric Service Bus Queues

In building a queuing mechanism, the team set out to incorporate the same features as Microsoft Messaging Queue (MSMQ). To that end, the new features were built by the same team that owns the MSMQ technology. However, in this case they were provided with an Internet-scale technology foundation, as well as the naming and discovery services provided by Service Bus. The result is a cloud cloud-based, message-oriented-middleware technologies to Service Bus that provide reliable message queuing and durable publish/subscribe messaging both over a simple and broadly interoperable REST-style HTTPS protocol with long-polling support and a throughput-optimized, connection-oriented, duplex TCP protocol.⁷ Figure 8-38 shows the AppFabric Service Bus Queues architecture.

Some of the functionality provided by AppFabric Queues includes:

- Peek-Lock delivery pattern for reliable delivery
- NET API and REST API
- Detection of duplicate inbound messages
- Dead-letter queue for messages that expire or fail
- Scheduled delivery of messages

⁷ Clemens Vasters' blog: <http://vasters.com/clemensv/2011/05/16/Introducing+The+Windows+Azure+AppFabric+Service+Bus+May+2011+CTP.aspx>

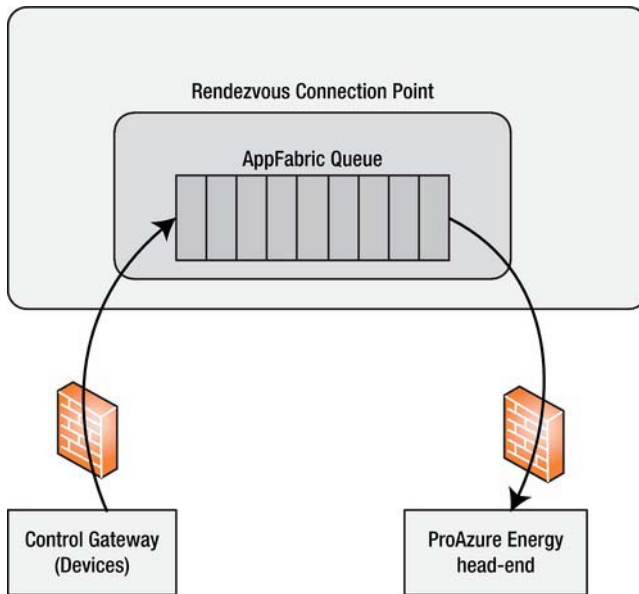


Figure 8-38. AppFabric Service Bus Queues

■ **Note** To view all Service Bus Messaging quotas, go to <http://msdn.microsoft.com/en-us/library/ee732538.aspx>.

AppFabric Service Bus Queues vs. Azure Storage Queues

Both mechanisms provide a queuing mechanism. So, what are the differences, and when should each be used? AppFabric Queues provide a richer messaging environment in that it supports protocols other than HTTP/S, in addition to enabling advanced messaging features:

- WCF binding
- Poison message handling
- Dead-lettering
- Transactions
- Groups
- Sessions
- duplicate detectionMessage Deferral/Scheduled Delivery
- Authentication via ACS

Also, both services support REST over HTTP, but if you require a higher level of performance, you can use bi-directional TCP with the AppFabric Queue.

Another key difference is that AppFabric Queues support the use of sessions. With this, you gain the ability to guarantee First-In First-Out ordering, as well as the ability to support Exactly-Once delivery.

If any of the mentioned capabilities are required, you will need to use AppFabric Queues. If you simply want to use a queue to support cross-service communication, such as inter-role communication at scale, then AppFabric Queues could be overkill. In this case, Azure Storage Queues should be sufficient.

AppFabric Service Bus Topics

Topics build on top of the queue mechanism to provide a way to distribute messages to multiple consumers through the service bus using simple rules via a publish/subscribe pattern. This can enable messaging scenarios where there is one central distribution point for messages, and multiple loosely connected receiver applications that can subscribe to the topic, and add rules to their subscription, so that only certain messages are received from the topic.

A topic allows for concurrent, durable subscriptions. Each subscription contains a set of filtering rules that use expressions to specify which messages should be delivered from the topic when the subscription is accessed.

Continuing with our energy example, let's say that ProAzure Energy decides they need to distribute their workload such that one system specifically handles data or commands specific to heating and cooling (HVAC), and another system handles all commands and data sent from lighting devices. This would normally be very complicated, because it would usually require an update to the devices to enable them to send to different locations. However, in this case Topics save the day. The servers simply create different subscriptions in order to pull different messages. The devices—they still send to the same topic as always, which saves a significant amount of re-work and device updating. See Figure 8-39.

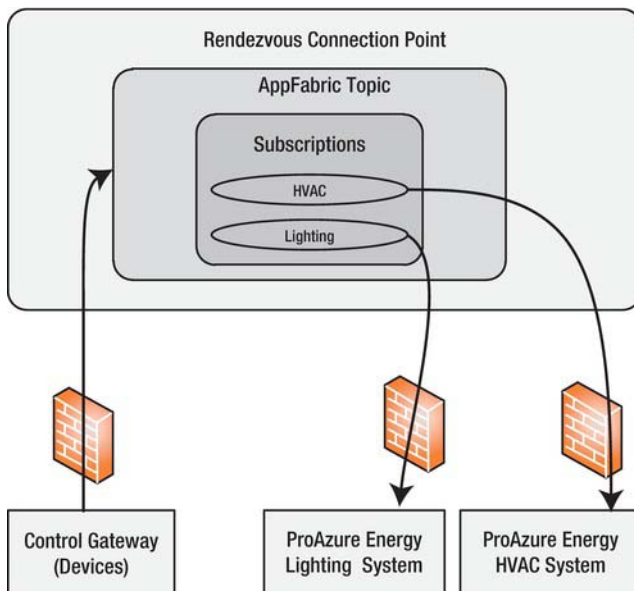


Figure 8-39. AppFabric Service Bus Topics

Subscription Rules

A particular subscription can contain one or more rules that specify what messages the subscription is expecting to find within the topic. When creating rules for a subscription, you have the options discussed in the following sections.

SQLFilterExpression

A `SQLFilterExpression` is an expression created in SQL 92 syntax. If the expression evaluates to true, then the message is a match, and will be delivered to the receiver through the subscription.

CorrelationFilterExpression

When using a `CorrelationFilterExpression`, you provide a GUID-based Correlation Id. This `CorrelationId` represents the header `X_MS_CORRELATION_ID` in the message. All messages with a matching Correlation Id will be delivered to the receiver through the subscription. When attempting to correlate messages, you would set this header when the message is sent to the Topic.

Programming Service Bus Queues and Topics

There are two avenues for programming solutions that use Queues and Topics: a .NET Client API and a REST-based API. We will explore both in the following sections.

.NET Client API

The .NET client API for this new functionality is located in two namespaces: `Microsoft.ServiceBus` and `Microsoft.ServiceBus.Messaging`. In your project, you will need to reference the `Microsoft.ServiceBus.dll` assembly.

Microsoft.ServiceBus Namespace

There are two key classes in this namespace related to Queues and Topics: `NamespaceManager` and `NamespaceManagerSettings`. These classes are used in conjunction with the classes in the `Microsoft.ServiceBus.Messaging` namespace to manage your Queues and Topics.

NamespaceManagerSettings

The `NamespaceManagerSettings` class provides the settings that drive `NamespaceManager` behavior. It contains two properties:

- *Operation Timeout*: Timeout period for all namespace management operations, which will be covered in the `NamespaceManager` section later in this chapter.
- *TokenProvider*: Allows you to define a `TokenProvider` the `NamespaceManager` object will use for authentication purposes.

NamespaceManager

The `NamespaceManager` class provides the ability to manage queues, topics, rules and subscriptions. You can use `NamespaceManager` to create or delete any of these entities, as well as view the metadata properties of each entity. In order to create a `NamespaceManager` object, we will need to provide the constructor with the URI of the namespace being managed, as well as either a `TokenProvider` object or a `NamespaceManagerSettings` object to define the behavior of the `NamespaceManager`.

Creating a Queue/Topic

You can create a `NamespaceManager` object and use it to create the entities you need. An example of creating a queue and a topic is shown in Listing 8-39. The create method for all entities provides overloaded parameters that support either passing in a string representing the path of the entity, or a `Description` object (`QueueDescription`, `TopicDescription`, `RuleDescription`, `SubscriptionDescription`), which contains the metadata needed to define the behavior of the entity. An example of using a `QueueDescription` to enable session state, dead-lettering, and duplicate detection is shown in Listing 8-40.

Listing 8-39. Creating a queue using `NamespaceManager`

```
var baseAddress = RoleEnvironment.GetConfigurationSettingValue("namespaceAddress");
var issuerName = RoleEnvironment.GetConfigurationSettingValue("issuerName");
var issuerKey = RoleEnvironment.GetConfigurationSettingValue("issuerKey");

Uri namespaceAddress = ServiceBusEnvironment.CreateServiceUri("sb", baseAddress,
string.Empty);

NamespaceManager namespaceManager = new NamespaceManager(namespaceAddress,
TokenProvider.CreateSharedSecretTokenProvider(issuerName, issuerKey));
// CreateQueue returns a QueueDescription object, which contains all queue metadata
var queueDescription = namespaceManager.CreateQueue("energyqueue");

// create a topic, returns the TopicDescription
var topicDescription = namespaceManager.CreateTopic("energytopic");

// add subscriptions to topic
var hvacSubscription = this.namespaceManager.CreateSubscription(topicDescription.Path,
"HVACSubscription", new SqlFilter("messageType='hvac'"));

var lightingSubscription = this.namespaceManager.CreateSubscription(topicDescription.Path,
"LightingSubscription", new SqlFilter("messageType='lighting'"));
```

Adding Session State to a Queue

Session state enables many possibilities such as FIFO or Exactly Once delivery. In order to set up a queue that requires session state, you need to pass in a `QueueDescription` object from the `Microsoft.ServiceBus.Messaging` namespace. An example of this is shown in Listing 8-40.

Listing 8-40. Creating a Queue with Session State

```
QueueDescription queueDescription = new QueueDescription {
    RequiresSession = true,
    RequiresDuplicateDetection = true,
    EnableDeadLetteringOnMessageExpiration = true };
this.namespaceManager.CreateQueue(queueDescription);
```

■ **Note** For more information about the all methods and properties available in the NamespaceManager class, go to <http://msdn.microsoft.com/en-us/library/hh293164.aspx>

Microsoft.ServiceBus.Messaging Namespace

There are many new classes in the API that facilitate the new messaging functionality:

Table 8-7. Key classes in Messaging API

Queue/Topic	Class	Description
Both	BrokeredMessage	The unit of communication, this represents the message that is brokered by the Messaging fabric.
Both	MessagingFactory	Factory class used to create messaging clients to send and receive messages.
Both	MessageReceiver	Receives and acknowledges messages from the container (Queue, Subscription).
Both	MessageSender	Send messages to the AppFabric Service Bus.
Queue	QueueClient	Used for run-time operations, such as sending and receiving messages.
Queue	QueueDescription	Used to set or get Queue metadata.
Topic	TopicClient	Used for run-time operations, sending and receiving messages to/from a topic.
Topic	TopicDescription	Used to set or get Topic metadata.
Topic	SubscriptionClient	Used for run-time operations related to subscriptions.
Topic	SubscriptionDescription	Used to set or get Subscription metadata.
Topic	RuleDescription	Used to set or get Rule metadata.

Foundational Message Components

No matter whether you are communicating with a Queue or a Topic, you will be sending or receiving a `BrokeredMessage` object. This object contains the message itself, plus all the properties that define the message metadata, such as `CorrelationId`, time the message expires, send to address, reply to address, and more.

The `MessagingFactory` is an object that represents the Service Bus Messaging namespace itself, and is responsible for creating messaging clients specific to the messaging pattern (Queue, Topic, Subscription). The messaging client object (`QueueClient`, `TopicClient`, `SubscriptionClient`) creates the objects that actually send or receive messages. Listing 8-41 shows how to create the components that will establish the means to implement a messaging infrastructure.

Listing 8-41. Creating Foundational Components Used for Either Sending or Receiving

```
// Create MessagingFactory for this namespace
MessagingFactory factory = MessagingFactory.Create(
    ServiceBusEnvironment.CreateServiceUri("sb", ServiceNamespace, string.Empty),
    credentials);

// Create Queue Client with PeekLock receive mode
QueueClient queueClient = factory.CreateQueueClient("energyqueue", ReceiveMode.PeekLock);

// Create Topic Client
TopicClient topicClient = factory.CreateTopicClient("energytopic");
```

Creating and Sending Messages

Once we have created the base components, we can create a message and send it to the messaging bus. In order to do so, we will first need to create messages to send. Then we will send some to the Queue, and some to the Topic to be picked up by separate subscriptions. The `BrokeredMessage.CreateMessage()` static method is used to create the message. This message serializes your object into the body of the message. You have the option of defining your own `XmlObjectSerializer`, or passing in a Stream object as well. See Table 8-8.

Table 8-8. CreateMessage Overloads

Name	Description
<code>CreateMessage()</code>	Creates a brokered message.
<code>CreateMessage(Object)</code>	Creates a brokered message from a given object by using <code>DataContractSerializer</code> with a binary <code>XmlDictionaryWriter</code> .
<code>CreateMessage(Stream, Boolean)</code>	Creates a brokered message using the supplied stream as its body.
<code>CreateMessage(Object, XmlObjectSerializer)</code>	Creates a brokered message from a given object using the provided <code>XmlObjectSerializer</code> .

Create and Send to Queue

Because Queues are less complicated than Topics and don't have any subscriptions or filtering rules, sending messages is simply a matter of creating the message and sending to the Queue.

Listing 8-42. Creating a Message and Sending to AppFabric Queue

```
// Create message
BrokeredMessage message = new BrokeredMessage("Test message");

// send to queue
queueClient.Send(message);
```

Retrieve from Queue

To retrieve from a queue, we create a `MessageReceiver` object, set the `ReceiveMode` (Peek-Lock in this case), and check the queue. We set the `waitTime` to 5 seconds, meaning that if the queue is empty, the receiver will wait five seconds in case any messages come in. If there are messages, it will return immediately.

Listing 8-43. Retrieve from Queue Using Peek-Lock

```
QueueClient queueClient = this.messagingFactory.CreateQueueClient(queueName,
    ReceiveMode.PeekLock);
// check for a message, wait 5 seconds if queue is empty
BrokeredMessage receivedMessage = queueClient.Receive(new TimeSpan(0, 0, 5));
```

Create and Send to Topic

When we send to a topic, we expect subscriptions to apply filtering rules. Hence, we need knowledge of the type of message being sent, so we can apply `FilterExpressions`. Typically, we would be able to refer to the properties of the serialized object. In this case, though, we are going to explicitly set the properties of the message. The `Properties Dictionary` object allows us to set application-specific properties, which is perfect for our purposes, because our message is a simple string object. We will set a property called `messageType`, which will contain either `hvac` or `lighting` as its value. In Listing 8-44, two messages are for HVAC, one is for lighting.

Listing 8-44. Creating and Sending Topic Messages

```
private static void CreateTopicMessage(string messageContents, string messageType)
{
    BrokeredMessage message = new BrokeredMessage(messageContents);
    message.Properties["messageType"] = messageType;
    topicClient.Send(message);
}
```

Retrieve Messages Using Subscriptions

Now that the messages are waiting for us in the topic, we can retrieve them using our subscription. For illustrative purposes, we are using the Peek-Lock mode to receive messages for HVAC messages, and

using `Receive` and `Delete` for the lighting messages. Note that in the Peek-Lock pattern, we have to use `message.Complete()` to remove the message from the queue once we are done processing. Running the code in Listing 8-45 should result in the HVAC subscription receiving two messages, and the lighting subscription receiving one.

Listing 8-45. Retrieving Messages Through Subscriptions

```
// HVAC subscription – PeekLock mode

SubscriptionClient hvacSubscriptionClient = factory.CreateSubscriptionClient("EnergyTopic",
"HVACSubscription", ReceiveMode.PeekLock);

// Lighting subscription – receive and delete
SubscriptionClient lightingSubscriptionClient =
factory.CreateSubscriptionClient("EnergyTopic", "LightingSubscription",
ReceiveMode.ReceiveAndDelete);

// get HVAC messages from topic
BrokeredMessage receivedHvacMessage = hvacSubscriptionClient.Receive(new TimeSpan(0, 0, 5));
string messageBody = message.GetBody<string>();
// Process the message here
message.Complete(); // remove the PeekLock, can ONLY be called when using PeekLock

// get lighting messages using receive and delete
BrokeredMessage receivedLightingMessage = lightingSubscriptionClient.Receive(new TimeSpan(0,
0, 5));
string messageBody = message.GetBody<string>();
// no need to complete, it was removed from queue
```

Handling Problem Messages and Abandonment

It's inevitable that errors will occur in message processing. Common scenarios include invalid or poison messages, or an issue that occurred with the server processing the message. Let's look at both scenarios.

Invalid/Poison Messages

In this case, the main concern is that we don't want to return these messages to the processing queue, as it will just cause issues for the next server that processes the messages. In addition, the poison messages will accumulate, and exponentially degrade performance. So it's not a good idea to abandon this message, nor to let the lock expire, as either will return the message to the queue. The best practice is to transfer it somewhere else where it can be handled as an exception. AppFabric Service bus provides the dead letter queue for exactly this purpose.

This is accomplished using the `DeadLetter()` method of the `BrokeredMessage` object. This moves it to a queue named `$DeadLetterQueue`. Once it arrives in this queue, you can decide how you want to handle the message. One approach would be to retrieve the messages using the `ReceiveAndDelete` pattern, and log them for later analysis. See Listing 8-46.

Listing 8-46. *Retrieving Messages from \$DeadLetterQueue and Log Information*

```
// Log the dead-lettered messages that could not be processed:
using (MessageReceiver deadLetterReceiver = queueClient.CreateReceiver("$DeadLetterQueue",
ReceiveMode.ReceiveAndDelete))
{
    BrokeredMessage receivedDeadLetterMessage;
    while (deadLetterReceiver.TryReceive(TimeSpan.FromSeconds(10), out
receivedDeadLetterMessage))
    {
        LogOrder(receivedDeadLetterMessage);
    }
}
QueueClient deadLetterClient = factory.CreateQueueClient("$DeadLetterQueue ",
ReceiveMode.ReceiveAndDelete);
BrokeredMessage deadLetterMessage = deadLetterClient.Receive(new TimeSpan(0, 0, 5));
//Process dead lettered message
```

Server Error During Message Processing

Another common scenario involves errors that occur on the server, while there is nothing wrong with the message, and it will need to be re-processed. In this scenario, we want to return the message to the queue so that another server can process it. The mechanisms for this depend on the ReceiveMode.

For Peek-Lock, we could simply let the TimeToLive expire, in which case the AppFabric Service Bus would return the message to the queue. However, if we want to be more proactive, we should use the Abandon() method of the BrokeredMessage object.

If you're using ReceiveAndDelete, then the message was deleted from the queue at the time it was received. Neither Abandon() or relying on TimeToLive will work. You'll have to explicitly send the message back into the queue.

REST API

If you don't want to use the .NET client, then the REST API exposes functionality that can be used to interact with queues. I've broken these out into several categories: Message commands and Management commands. Message commands simply send/receive/delete messages to and from existing queues and topics, while the management commands provide the ability to manage the queues or topics themselves.

■ **Note** When using the REST API, the sb:// is replaced by https://. Also, unless otherwise noted, all requests require HTTP/1.1 version. Additionally, set request Header Content-type to application/atom+xml;type=entry;charset=utf-8.

Securing REST API Requests with Access Control Service

You can secure your REST API requests using WRAPv0.9.7.2 SWT tokens obtained from the Access Control Service. See Chapter 6 for more information about obtaining tokens from ACS. A string such as this will be returned from ACS:

```
wrap_access_token=net.windows.servicebus.action%3dListen%252cManage%252cSend%26http%253a%252f%252fschemas.microsoft.com%252faccesscontrolservice%252f2010%252f07%252fclaims%252fidentityprovider%3dhttps%253a%252f%252fbvt.s1002-sbususer-0-9-sb.accesscontrol.aadint.windows-int.net%252f%26Audience%3dhttp%253a%252f%252fbvt.s1002-sbususer-0-9.Windows-bvt.net%26ExpiresOn%3d1304710330%26Issuer%3dhttps%253a%252f%252fbvt.s1002-sbususer-0-9-sb.accesscontrol.aadint.windows-int.net%252f%26HMACSHA256%3d3mytM7yEZ4ZDHy05rDBeReJien%252f%252bIrmJJVeZsUPqbU%253d&wrap_access_token_expires_in=1199
```

You'll need to extract the token and URL-decode. Also, MSDN documentation notes the following important points⁸:

- The received string is URI-decoded (%26 => &) and is in double quotes. Put this into the `HttpAuthorizationHeader`.
- The `ExpiresOn` time in the middle of the string is specified as a Unix File Time (that is, the number of seconds since 01/01/1970 at 12:00am). You should scrub the identity provider, audience, issuer and `hmacsha` fields.
- The domain used when requesting a token uses the HTTP scheme, even though calls to the service are always issued over HTTPS.
- Make sure that the content type in the HTTP header is of type: `application/x-www-form-urlencoded`.

Once completed, it should look something like

```
WRAP_access_token="net.windows.servicebus.action=Listen%2cManage%2cSend&http%3a%2f%2fschemas.m
icrosoft.com%2faccesscontrolservice%2f2010%2f07%2fclaims%2fidentityprovider=https%3a%2f%2fbVtS
n1002-sbususer-0-9-sb.accesscontrol.aadint.windows-int.net%2f&Audience=http%3a%2f%2fbVtSn1002-
sbuser-0-9-Windows-bvt.net&ExpiresOn=1304710330&Issuer=https%3a%2f%2fbvtSn1002-sbususer-0-9-
sb.accesscontrol.aadint.windows-
int.net%2f&HMACSHA256=3mytM7yEZ4ZDHy05rDBeReJien%2f%2bIrmJJVezsUpqbu%3d"
```

Once this token is fully extracted, you can add the Authorization Request Header, and set to WRAP access_token="{swt}", where {swt} is the token you obtained.

Queues: Message Commands

Message commands are commands that facilitate the sending, receiving, and deleting of messages from a queue. Next we will cover how to perform each of these tasks.

⁸ Appfabric Service Bus REST API Reference: http://msdn.microsoft.com/en-us/library/gg278338.aspx#RESTAPI_1

■ **Note** The REST API contains only a subset of functionality provided by the .NET Client API. Missing are features to group receivers, enrich messages, set custom filter destinations, and perform batching.

Send to Queue

In order to send a request to a Queue via the REST API, create a PUT request:

PUT https://{servicenamespace.Windows.net[:{port}]/{path} HTTP/1.1

{path} can be any depth you wish, but has a maximum length of 290 characters. For example, you could specify a path of /US, /US/CA, or /US/CA/SanRamon. It is just the path to your queue, you can name it logically.

In the body of the request, pass your message in an Atom entry:

```
<entry xmlns='http://www.w3.org/2005/Atom'>
  <content type='application/xml'>
    {description}
  </content>
</entry>
```

If the operation fails, you'll receive a response code indicating the reason for failure. Otherwise, you'll receive a 200 code with the following response:

```
<?xml version="1.0" encoding="utf-8" ?>
<entry xmlns='http://www.w3.org/2005/Atom'>
  <id>https://{serviceNameSpace}.servicebus.windows.net/{path}</id>
  <published>{createdTime}</published>
  <updated>{lastUpdatedTime}</updated>
  <link rel='self'>https://{serviceNameSpace}.servicebus.windows.net/{path} </link>
  <content type='application/xml'>
    {description}
  </content>
</entry>
```

Receive from Queue

The receive operation is represented by a GET operation, following the same pattern as the PUT used to send a message to the Queue:

https://{servicenamespace.Windows.net[:{port}]/{path}

If the operation fails, you'll receive a response code indicating the reason for failure. Otherwise, you'll receive a 200 code with the following response:

```
<?xml version="1.0" encoding="utf-8" ?>
<entry xmlns='http://www.w3.org/2005/Atom'>
  <id>https://{serviceNameSpace}.Windows.net/{path}</id>
  <published>{createdTime}</published>
  <updated>{lastUpdatedTime}</updated>
  <link rel='self'>https://{serviceNameSpace}.Windows.net/{path} </link>
```

```
<content type='application/xml'>
  {description}
</content>
</entry>
```

Delete from Queue

The receive operation is represented by a GET operation, following the same pattern as the PUT used to send a message to the Queue:

`https://{servicenamespace.Windows.net[:{port}]}/{path}`

If the operation fails, you'll receive a response code indicating the reason for failure. Otherwise, you'll receive a 200 code with nothing in the response body.

Queue: Management Commands

Management commands are commands that involve creating or deleting a queue, or getting information about the queue itself. In the following we cover the commands that are available in the REST interface.

QueueDescription

It's important to cover the queue description first. This is an AtomPub document that defines the properties for a queue. It is used in REST API request and responses, sent when creating a queue, or received when requesting the properties of a queue. The QueueDescription properties are shown in Table 8-9.

Table 8-9. Properties of QueueDescription

Property	Description	Range	Default
MaxQueueSizeInBytes	Maximum queue size (in bytes)	1-100*1024*1024	100*1024*1024
DefaultMessageTimeToLive	Default time to live for a message before it is either deleted or moved to the DeadLetterQueue	1 second – TimeSpan.MaxValue	TimeSpan.MaxValue
LockDuration	Amount of time a message is locked while being processed. Once reached, the message is unlocked for the next receiver to	0-300 seconds	30 seconds

	consume		
RequiresSession	Sets queue session awareness. Not supported through REST interface, intended for .NET client API.	true, false	False
RequiresDuplicateDetection	Indicates whether service bus should check for duplicate messages.	true, false	False
EnableDeadLetteringOnMessageExpiration	Defines whether to delete a message or move to DeadLetterQueue once message TTL is reached.	true, false	False
DuplicateDetectionHistoryTimeWindow	Defines the time span for which Service bus will check for duplicate messages	1 second – 7 days	10 minutes

Create Queue

To create a new queue, execute the following REST command:

PUT https://{serviceName}.windows.net/{Queue Path} HTTP/1.1

The payload for this command is a queue description as defined, which sets the properties and behaviors for the queue. Once you have created a queue, you cannot change its properties; you will have to delete and re-create with a new queue description. See Listing 8-47.

Listing 8-47. Creating an AppFabric Service Bus Queue

```
PUT /MyQueues/Queue1 HTTP/1.1
Host: proazure-1.servicebus.windows.net
Content-Type: application/atom+xml
Accept: application/atom+xml
Authorization: ...
Content-Length: nnn

<entry xmlns='http://www.w3.org/2005/Atom'>
<content type='application/xml'>
```

```
<QueueDescription xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.microsoft.com/net/services/2010/10/servicebus/connect">
  <LockDuration>PT30S</LockDuration>
  <MaxQueueSizeInBytes>104857600</MaxQueueSizeInBytes>
  <RequiresDuplicateDetection>>false</RequiresDuplicateDetection>
  <RequiresSession>>false</RequiresSession>
  <DefaultMessageTimeToLive>P10675199DT2H48M5.4775807S</DefaultMessageTimeToLive>
  <DeadLetteringOnMessageExpiration>>false</DeadLetteringOnMessageExpiration>
  <DuplicateDetectionHistoryTimeWindow>PT10M</DuplicateDetectionHistoryTimeWindow>
</QueueDescription>
</content>
</entry>
```

Delete Queue

To delete a queue, execute the following REST command:

```
DELETE https://{serviceNamespace}.windows.net/{Queue Path} HTTP/1.1
```

Keep in mind that this deletes all the messages in the queue as well. If you are trying to re-create a queue with new properties, or migrating to a new queue, you will want to make sure you get all messages off the queue before you delete it. See Listing 8-48.

Listing 8-48. Deleting an AppFabric Service Bus Queue

```
DELETE /MyQueues/Queue1 HTTP/1.1
Host: proazure-1.servicebus.windows.net
Content-Type: application/atom+xml
Accept: application/atom+xml
Authorization: ...
Content-Length: nnn
```

Get Queue

This command gets the queue and all its associated state. Of course, this means it must remove all messages from the queue to return the state information. So, you might use this command in the migration of messages from one queue to another, or in the deleting and re-creating a queue with new properties (see Listing 8-49). Execute the following REST Command:

```
GET https://{serviceNamespace}.windows.net/{Queue Path} HTTP/1.1
```

Listing 8-49. Deleting an AppFabric Service Bus Queue

```
DELETE /MyQueues/Queue1 HTTP/1.1
Host: proazure-1.servicebus.windows.net
Content-Type: application/atom+xml
Accept: application/atom+xml
Authorization: ...
Content-Length: nnn
```


List Queues

Lists all queues that exist in the service namespace.

```
GET https://{serviceNamespace}.windows.net/$Resources/Queues HTTP/1.1
```

Topics and Subscriptions: Message Commands

Message commands are commands that facilitate the sending, receiving, and deleting of messages from a topic. In the following sections we will cover how to perform each of these tasks.

Send to Topic

To enqueue a message into a topic, execute the following REST command:

```
POST http{s}://{serviceNamespace}.Windows.net/{topic path}/messages HTTP/1.1
```

The request body contains the message payload. When the topic is created, the maximum number of messages may be set in the topic description. If the topic is already at its maximum number of messages allowed, a quota exceeded error will be returned.

Read Message from Subscription with Non-Destructive Peek-Lock

Use this technique when At-Least-Once delivery is required. When you use this command, the message is read from the queue but is locked, thus preventing other receivers from being able to process the message. If the lock expires, then the message will be available for other receivers to process.

It's important to note that when using this pattern, the receiver is responsible for deleting the message with the lock ID received from this operation once processing is complete. If the receiver does not delete the message, then the lock will eventually expire, and it will be processed by another receiver, resulting in duplicate processing. If processing must be abandoned, the receiver should issue an unlock command so that other receivers are free to process the message.

To use Peek-Lock reading of messages:

```
POST https://{serviceNamespace}.Windows.net/{topic path}/subscriptions/{subscription Name}/messages/head?timeout={timeout} HTTP/1.1
```

Note the URI parameter timeout. This represents the amount of time the server will wait for messages if there are no existing messages. Acceptable values are 0-120 seconds, and 0 is the default.

There are several important headers returned in the response. All of the information returned is required to delete or unlock the message once processing is complete or abandoned. See Table 8-10.

Table 8-10. Peek-Lock Response Headers

Response Header	Description
X-MS-MESSAGE-LOCATION	URI of the message for unlocking purposes.
X-MS-LOCK-ID	Lock Id that needs to be passed when deleting a locked message.
X-MS-LOCK-LOCATION	The lock URI for the locked message for unlocking when abandoning

message processing.

Read and Delete Message from Subscription (Destructive read)

This command should be used in scenarios where At-Least-Once delivery is not required, and some loss of messages is acceptable. The reason you would want to use this instead of the Peek-Lock is that the read and delete executes as an atomic operation. No locks are held, and the receiver is not required to return to the topic to delete the message. Reducing that extra processing required to support Peek-Lock will increase performance.

To execute this command:

```
DELETE http{s}://{serviceNameSpace}.windows.net/{topic path}/subscriptions/{subscription
Name}/messages/head?timeout={timeout} HTTP/1.1
```

There is only one URI Parameter for this command: timeout. This parameter represents the amount of time the server will wait for messages if there are no existing messages. Acceptable values are 0-120 seconds, and 0 is the default.

Unlock Message from Subscription

If you have abandoned processing and want to make the message available for other receivers to process, you will need to remove the lock object. Execute this command:

```
DELETE http{s}://{serviceNameSpace}.servicebus.windows.net/{buffer}/messages/{message-
id}/{lock-id} HTTP/1.1
```

Note the URI parameters in this request. Message-id is the Id of the message to be unlocked. You got this in the X-MS-MESSAGE-LOCATION response header when you retrieved the message using the PeekLock command. Lock-id is the X-MS-LOCK-ID response header that was also returned in the same response.

Delete Message from Subscription

Once your receiver has completed processing in a Peek-Lock scenario, it will need to delete the message from the subscription to prevent duplicate processing. To execute this command:

```
DELETE http{s}://{serviceNameSpace}.Windows.net/{topic path}/subscriptions/{subscription
Name}/messages/{message-id}?lockid={lock-id} HTTP/1.1
```

Once again, the same URI parameters are required as when unlocking a message. Message-id is the ID of the message to be unlocked. You got this in the X-MS-MESSAGE-LOCATION response header when you retrieved the message using the PeekLock command. Lock-id is the X-MS-LOCK-ID response header that was also returned in the same response.

Topics: Management Commands

Management commands are commands that involve creating or deleting a topic, or getting information about the topic itself. Later we cover the commands that are available in the REST interface.

TopicDescription

As with queues, topic descriptions are used to define the properties for a topic. This is an AtomPub document that defines the properties for a queue. It is used in REST API request and responses, sent when creating a topic, or received when requesting the properties of a topic. The properties of the TopicDescription object are shown in Table 8-11.

Table 8-11. Topic Description Properties

Property	Description	Range	Default
MaxTopicSizeInBytes	Maximum topic size (in bytes). Once reached, all attempts to enqueue messages will result in an error being returned.	1-100 MB	100*1024*1024
DefaultMessageTimeToLive	Default time to live for a message before it is either deleted or moved to the DeadLetterQueue	1 second – TimeSpan.MaxValue	TimeSpan.MaxValue
MaximumNumberOfSubscriptions	Maximum number of subscriptions that can be associated with a topic	1-2000	2000
MaximumNumberOfSqlFilters	Maximum number of SQL filter expressions	1-2000	2000
MaximumNumberOfCorrelationFilters	Max number of correlation filter expressions	1-2000	2000
RequiresDuplicateDetection	Indicates whether service bus should check for duplicate messages	true, false	false
EnableDeadLetteringOnMessageExpiration	Defines whether to delete a message or move to DeadLetterQueue once message TTL is reached	True, false	false
DuplicateDetectionHistoryTimeWindow	Defines the time span for which Service bus will check for duplicate	1 second – 7 days	10 minutes

messages

■ **Note** Multiple copies of a message that exist in multiple subscriptions are counted as single message, and thus having a message on multiple subscriptions does not count against the size quota.

Create Topic

To create a new topic, execute the following REST command:

```
PUT https://{serviceNamespace}.windows.net/{topic Path} HTTP/1.1
```

The payload for this command is a topic description as defined earlier, which sets the properties and behaviors for the topic. Once you have created a topic, you cannot change its properties, you will have to delete and re-create with a new topic description. See Listing 8-50.

Listing 8-50. Creating an AppFabric Service Bus Topic

```
PUT /MyTopics/Topic1 HTTP/1.1
Host: proazure-1.servicebus.windows.net
Content-Type: application/atom+xml
Accept: application/atom+xml
Authorization: ...
Content-Length: nnn
```

```
<entry xmlns='http://www.w3.org/2005/Atom'>
<content type='application/xml'>
<TopicDescription xmlns:i='http://www.w3.org/2001/XMLSchema-instance'
xmlns='http://schemas.microsoft.com/net/services/2010/10/servicebus/connect'>
  <DefaultMessageTimeToLive>P10675199DT2H48M5.4775807S</DefaultMessageTimeToLive>
  <MaxTopicSizeInBytes>104857600</MaxTopicSizeInBytes>
  <RequiresDuplicateDetection>false</RequiresDuplicateDetection>
  <DuplicateDetectionHistoryTimeWindow>P7D</DuplicateDetectionHistoryTimeWindow>
  <MaxSubscriptionsPerTopic>2000</MaxSubscriptionsPerTopic>
  <MaxSqlFiltersPerTopic>1000</MaxSqlFiltersPerTopic>
  <MaxCorrelationFiltersPerTopic>2000</MaxCorrelationFiltersPerTopic>
  <DeadLetteringOnMessageExpiration>false</DeadLetteringOnMessageExpiration>
  <DeadLetteringOnFilterEvaluationExceptions>true</DeadLetteringOnFilterEvaluationExceptions>
</TopicDescription>
</content>
</entry>
```

Delete Topic

To delete a topic, execute the following REST command:

```
DELETE https://{serviceNamespace}.windows.net/{Topic Path} HTTP/1.1
```

Keep in mind that this deletes all the subscriptions and messages in the topic as well. If you are trying to re-create a topic with new properties, or migrating to a new topic, you will want to make sure you get all subscriptions and messages off the topic before you delete it.

Get Topic

This command simply retrieves the topic description for the topic.

GET https://{serviceName}.windows.net/{Topic Path} HTTP/1.1.

List Topics

Lists all topics that exist in the service namespace.

GET https://{serviceName}.windows.net/\$Resources/Topics HTTP/1.1

Subscriptions: Management Commands

Management commands are commands that involve creating or deleting a subscription, or getting information about the subscription itself. Later we cover the commands that are available in the REST interface.

SubscriptionDescription

Continuing the theme of setting and retrieving properties via description objects, the Subscription description is an AtomPub document that defines the properties for a subscription. It is used in REST API request and responses, sent when creating a subscription, or received when requesting the properties of a subscription. The properties of the SubscriptionDescription object are shown in Table 8-12.

Table 8-12. SubscriptionDescription Properties

Property	Description	Range	Default
DefaultMessageTimeToLive	Default time to live for a message before it is either deleted or moved to the DeadLetterQueue	1 second – TimeSpan.MaxValue	TimeSpan.MaxValue
LockDuration	Amount of time a message is locked while being processed. Once reached, the message is unlocked for the next receiver to consume	0-300 seconds	30 seconds

RequiresSession	Sets subscription session awareness. Not supported through REST interface, intended for .NET client API.	true, false	false
EnableDeadLetteringOnMessageExpiration	Defines whether to delete a message or move to DeadLetterQueue once message TTL is reached.	true, false	False
DuplicateDetectionHistoryTimeWindow	Defines the time span for which Service bus will check for duplicate messages	1 second – 7 days	10 minutes

Create Subscription

To create a new subscription, execute the following REST command:

```
PUT https://{serviceNamespace}.servicebus.windows.net/{topic path}/subscriptions/{subscription name} HTTP/1.1
```

The payload for this command is a subscription description as defined previously, which sets the properties and behaviors for the subscription. Once you have created a subscription, you cannot change its properties, you will have to delete and re-create with a new subscription description. See Listing 8-51.

Listing 8-51. Creating a Subscription

```
PUT /MyTopics/Topic1/Subscriptions/FirstSubscription HTTP/1.1
Host: proazure-1.Windows.net
Content-Type: application/atom+xml
Accept: application/atom+xml
Authorization: ...
Content-Length: nnn

<entry xmlns="http://www.w3.org/2005/Atom">
  <title type="text">MySubscription</title>
  <link rel="alternate"
href="https://contoso.Windows.net/MyTopic/subscriptions/MySubscription"/>
  <link rel="self" href="https://Contoso.Windows.net/Resources/Topics('MyTopic')/
Subscriptions('MySubscription')"/>

  <content type="application/xml"
xmlns="http://schemas.microsoft.com/net/services/201?/???/servicebus/connect">
```

```

    <SubscriptionDescription>
      <MaxSubscriptionSizeInBytes>100000000</MaxSubscriptionSizeInBytes>
      <LockDuration>P30S</LockDuration>
      <RequiresMessageGrouping>False</RequiresMessageGrouping>
      <DefaultRule>True</DefaultRule>
    </SubscriptionDescription>
  </content>
</entry>
<entry xmlns='http://www.w3.org/2005/Atom'>
<content type='application/xml'>
<SubscriptionDescription xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.microsoft.com/net/services/2010/10/servicebus/connect">
  <LockDuration>PT5M</LockDuration>
  <RequiresSession>>false</RequiresSession>
  <DefaultMessageTimeToLive>P10675199DT2H48M5.4775807S</DefaultMessageTimeToLive>
  <DeadLetteringOnMessageExpiration>>false</DeadLetteringOnMessageExpiration>
<DeadLetteringOnFilterEvaluationExceptions>true</DeadLetteringOnFilterEvaluationExceptions>
</SubscriptionDescription>
</content>
</entry>

```

Delete Subscription

To delete a subscription, execute the following REST command:

```

DELETE https://{serviceName}.servicebus.windows.net/{topic
path}/subscriptions/{subscription name}
HTTP/1.1

```

Keep in mind that this deletes all the messages in the subscription as well. If you are trying to re-create a subscription with new properties, or migrating to a new subscription, you will want to make sure you get all messages off the subscription before you delete it.

Get Subscription

This command simply retrieves the topic description for the topic.

```

GET https://{serviceName}.windows.net/{topic path}/subscriptions/{Subscription Name}
HTTP/1.1.

```

List Subscriptions

Lists all subscriptions that exist in the specified topic.

```

GET https://{serviceName}.windows.net/{topic path}/subscriptions/
HTTP/1.1

```

Rules: Mangement Commands

Management commands are commands that involve creating or deleting a rule, or getting information about the rule itself. In the following we cover the commands that are available in the REST interface.

Rule Description

Rule Description is an AtomPub document that defines the properties for a Rule. It is used in REST API request and responses, sent when creating a rule, or received when requesting the properties of a rule. The properties of the RuleDDescription object are shown in Table 8-13.

Table 8-13: Rule Description Properties

Property	Description
Filter: SqlFilterExpression	A SQL 92 syntax expression encoded as a string. Expression must evaluate to true or false.
Filter: CorrelationFilterExpression	Match based on GUID-based correlationId. Messages whose CorrelationId (X-MS-CORRELATION-ID) match will be returned.
FilterAction	The action to be taken if the rule expression evaluates to true. A string interpreted as a SQL 92 operation.

Create Rule

To create a new Rule, execute the following REST command:

```
PUT https://{serviceNamespace}.windows.net/{topic path}/subscriptions/{subscription
name}/rules/{rule name} HTTP/1.1
```

The payload for this command is a rule description as defined earlier, which sets the properties and behaviors for the rule. Once you have created a rule, you cannot change its properties, you will have to delete and re-create with a new rule description. See Listing 8-52.

Listing 8-52. Creating a Rule

```
PUT /MyTopics/Topic1/Subscriptions/FirstSubscription/Rules/FirstRule HTTP/1.1
Host: proazure-1.servicebus.windows.net
Content-Type: application/atom+xml
Accept: application/atom+xml
Authorization: ...
Content-Length: nnn
```

```
<entry xmlns='http://www.w3.org/2005/Atom'>
<content type='application/xml'>
<RuleDescription xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.microsoft.com/net services/2010/10/servicebus/connect">
```



```

<Filter i:type="SqlFilterExpression">
  <SqlExpression>MyProperty='XYZ'</SqlExpression>
</Filter>
<Action i:type="SqlFilterAction">
  <SqlExpression>set MyProperty2 = 'ABC'</SqlExpression>
</Action>
</RuleDescription>
</content>
</entry>

```

Delete Rule

To delete a rule, execute the following REST command:

```
DELETE https://{serviceNamespace}.windows.net/{topic path}/subscriptions/{subscription
name}/rules/{rule name} HTTP/1.1
```

Get Rule

This command simply retrieves the rule description for the rule.

```
GET https://{serviceNamespace}.windows.net/{topic path}/subscriptions/{subscription
name}/rules/{rule name} HTTP/1.1.
```

List Rules

Lists all rules that exist in the specified topic.

```
GET https://{serviceNamespace}.windows.net/{topic path}/rules/
HTTP/1.1
```

Summary

Microsoft has built the AppFabric Service Bus as a foundation for cross-platform and cross-enterprise application integration. Services across the same or different enterprises can communicate with each other, even if they're behind firewalls. Its integration with ACS and the security at the transport-level makes it secure to send encrypted messages over the Internet. The programming model is very similar to WCF, and you can utilize your existing WCF skills to build AppFabric Service Bus applications.

Message buffers are a different concept than WCF programming, but they're similar to the Windows Azure queues that you read about earlier in the book. You can use message buffers in nonreliable asynchronous store-and-forward scenarios.

In this chapter, you learned the concepts behind the AppFabric Service Bus that can help you build integration applications at Internet-scale. Early releases of the AppFabric Service Bus included another component called Workflow Services that is planned for a future release.

The next chapter covers Microsoft's database for the cloud: SQL Azure.

Bibliography

Lowy, J. (n.d.). Securing The .NET Service Bus. Retrieved from MSDN: <http://msdn.microsoft.com/en-us/magazine/dd942847.aspx>.

Microsoft Corporation. (2009). Windows Azure platform AppFabric November 2009 CTP. Retrieved from MSDN: <http://msdn.microsoft.com/en-us/library/ee173584.aspx>.

Microsoft Corporation. (n.d.). Windows Azure SDK. Retrieved from MSDN: <http://msdn.microsoft.com/en-us/library/dd179367.aspx>.

OASIS Standards. (n.d.). OASIS Standards. Retrieved from OASIS Standards: www.oasis-open.org/home/index.php.

Vasters, C. (n.d.). Azure: Microsoft .NET Service Bus. Retrieved from Clemens Vasters, Bldg 42: <http://blogs.msdn.com/clemensv/archive/2008/10/27/azure-microsoft-net-service-bus.aspx>.

Microsoft Corporation (2011) Service Bus API REST Interface. Retrieved from MSDN: <http://msdn.microsoft.com/en-us/library/hh367521.aspx>

Microsoft Corporation (2011) Windows Azure AppFabric Class Library. Retrieved from MSDN: <http://msdn.microsoft.com/en-us/library/hh394905.aspx>

AppFabric: Caching

In my conversations with customers, I make a point to stress the importance of understanding the implications of a PaaS architecture. With Windows Azure, the fact that load balancing can only be done in a round-robin fashion certainly has architectural impact, especially when pitted against the fact that two instances must be running to remain within the terms of the Azure Service Level Agreement. This is where customers typically realize what I really mean when I say that an app must be stateless.

Sometimes, this is followed by a confession that their “stateless” web farm-based application uses in-memory session state, and that they employ a sticky-IP type of load balancing to ensure that requests go to the same server. And what follows that is a long discussion about the merits of rewriting the app versus the performance implications of switching to SQL Server for session state.

One way around this is to store session state in SQL Azure. However, this creates additional processing load and extra connections on SQL Azure. Under a heavy system load, this could be the difference between getting throttled or not. In addition, the overhead of opening and closing connections can have a performance impact.

In June 2011, Windows Azure AppFabric Caching was released. With it came the solution to both of these problems, among others. It is a major piece of the puzzle in making Windows Azure a complete platform. AppFabric Caching is a distributed in-memory cache provided as a cloud service. It provides the low latency of an in-memory cache, while providing high scalability by having the cache distributed across the memory of multiple nodes. In addition, it is provided as a high throughput cloud service, meaning there’s no need to manage instances to scale up or down.

The architecture/typical usage of Azure AppFabric Caching is shown in Figure 9-1.

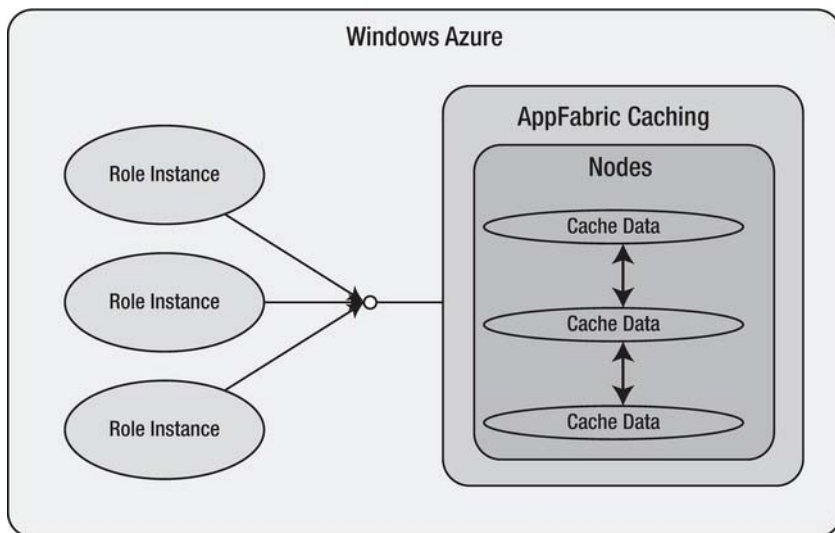


Figure 9-1. Windows Azure AppFabric caching

■ **Note** People are often confused by Microsoft's naming of Windows Server AppFabric and Windows Azure AppFabric. They are not the same set of services. However, in this case, they really are built on the same underlying technology, except that the Azure AppFabric Caching Service provides a subset of features from Server AppFabric Caching. However, Azure Caching doesn't require any installation or setup. It is provisioned and maintained via the portal. For more information about the differences between Azure AppFabric and Server AppFabric Caching, go to <http://msdn.microsoft.com/en-us/library/gg185678.aspx>.

In this chapter, we will compare AppFabric Caching to its open-source counterpart, memcached. We will then delve into provisioning a cache and creating clients that can interact with the cache. Finally, we will take a look at specific ASP.NET scenarios that are enabled by AppFabric Cache: Session State and Page Output Caching.

AppFabric Caching vs. Other Cache Providers

There are other providers of caching services that can exist in Windows Azure, but none is as well integrated and scalable. For example, memcached is a utility that will run in Windows Azure, but AppFabric Caching offers several advantages. Most notably, there is little setup required to provision or scale AppFabric Caching. Also, scalability is dynamic and automatic in AppFabric Caching, whereas if using memcached, you would need to scale role instances up and down as necessary, requiring you to implement an auto-scaling solution.

The biggest disadvantage of AppFabric Caching is that the initial release only included support for the .NET client SDK. Java and REST APIs were not released, but will likely be included in future releases.

Once we have decided that using AppFabric Cache is the appropriate solution, we need to provision the cache, which is described in the following section.

Provisioning an AppFabric Cache

Provisioning a cache is quite simple. In fact, you may have already done it in the process of creating an appfabric namespace for either Service Bus or Access Control Service. In the AppFabric section of the Azure Management Portal, click New Namespace and provide a name for your namespace. To add a cache to an existing namespace, click Modify Namespace. Note that, in Figure 9-2, you can choose a cache size from 128 MB to 4GB. Once provisioned, you can change the cache size by clicking the Change Cache Size button.

Figure 9-2. Provisioning a Windows Azure AppFabric cache

Once you have provisioned the cache, you need to create clients who can interact with the server cache. This is described in the following sections.

AppFabric Cache Clients

The application that accesses the cache must be configured to use the cache. The application will need references to the Caching DLLs, as well as configuration to access the cache.

Assembly References

To enable an application to use the AppFabric Cache, you will need to add references to `Microsoft.ApplicationServer.Caching.Client.dll` and

Microsoft.ApplicationServer.Caching.Core.dll. In addition, for ASP.NET applications, you will need a reference to Microsoft.Web.DistributedCached.dll.

These DLLs are part of the Windows Azure AppFabric SDK. To add the reference, you will need to click the Browse tab of the Add Reference dialog in Visual Studio. The default location for these assemblies is .\Program Files\Windows Azure AppFabric SDK\V1.0\Assemblies\NET4.0\Cache.

Configuring the Cache Client

The cache client can be configured either programmatically or via configuration. The cache can be configured to use SSL or non-SSL endpoints.

Configuring Cache Client Using Application Configuration File

Fortunately, the Management Portal provides most of the information you will need to configure the cache client via the config file. In the Management Portal, click View Client Configuration. A dialog box will display the configuration code you need, including your authorization token and namespaces, as shown in Figure 9-3. Copy this code and paste it into your app.config or web.config file. In Listing 9-1, I have also provided sample configuration code you can paste into the application, and then look up the necessary information in the properties for the cache on the right-hand side of the portal.

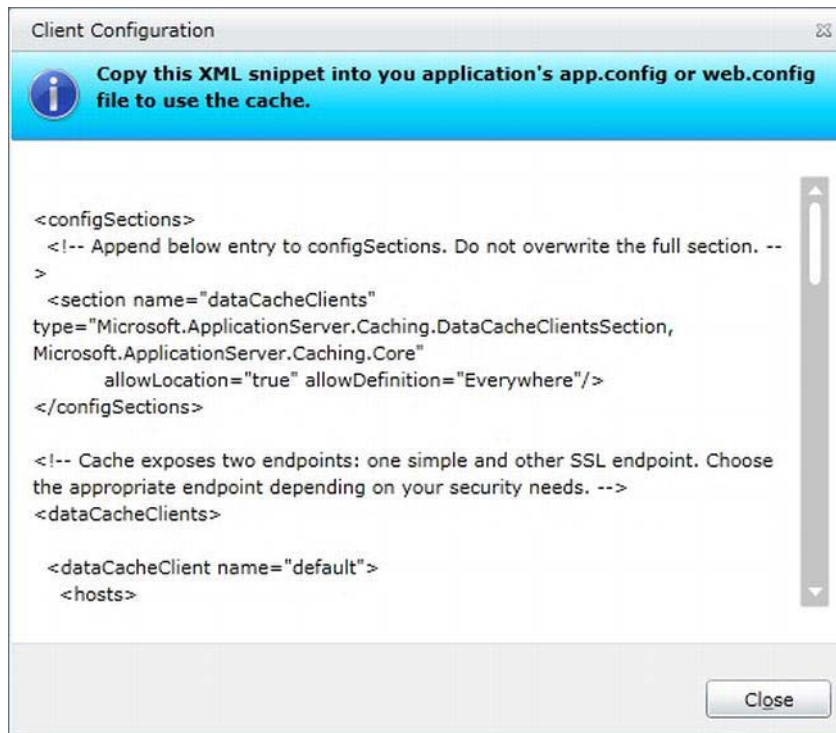


Figure 9-3. Client configuration XML snippet from Azure Management portal

■ **Note** The `sessionState` and `outputCache` sections are only relevant for ASP.NET applications. You should remove these sections if your application is not ASP.NET, or you don't want to use these features. I will cover usage of these features in ASP.NET later in this chapter, in the sections "ASP.NET Session State Provider" and "Enabling ASP.NET Output Cache in AppFabric Cache."

Listing 9-1. Configuration Sections for Default and SSL Endpoints

```
<configSections>
  <!-- Append below entry to configSections. Do not overwrite the full section. -->
  <section name="dataCacheClients"
type="Microsoft.ApplicationServer.Caching.DataCacheClientsSection,
Microsoft.ApplicationServer.Caching.Core"
    allowLocation="true" allowDefinition="Everywhere"/>
</configSections>

<dataCacheClients>
  <dataCacheClient name="default">
    <hosts>
      <host name="[Insert Cache EndPoint]" cachePort="22233" />
    </hosts>
    <securityProperties mode="Message">
      <messageSecurity
        authorizationInfo="[Encrypted ACS token goes here]">
      </messageSecurity>
    </securityProperties>
  </dataCacheClient>

  <dataCacheClient name="SslEndpoint">
    <hosts>
      <host name="[Insert Cache EndPoint]" cachePort="22243" />
    </hosts>
    <securityProperties mode="Message" sslEnabled="true">
      <messageSecurity
        authorizationInfo="[Encrypted ACS token goes here]">
      </messageSecurity>
    </securityProperties>
  </dataCacheClient>

</dataCacheClients>
```

Once the work is complete in the config file, the next step is to create the cache in your application code. This is relatively straightforward. The settings in the application configuration file are the default settings for the `DataCacheFactory`, and so we can initialize the cache using default settings.

Listing 9-2. Initializing the Cache in the Application

```
using Microsoft.ApplicationServer.Caching;

// skipping through code to relevant part

// Create Cache client with default settings from app.config or web.config.
DataCacheFactory cacheFactory = new DataCacheFactory();
DataCache cache = cacheFactory.GetDefaultCache();

// To get SSL cache instead...
DataCache sslCache = cacheFactory.GetCache("SslEndpoint");
```

Configuring Cache Client Programmatically

While using the configuration file seems pretty easy, perhaps your application requires the cache to be configured at run time. For this and other scenarios, you can configure the cache client programmatically.

You will need several pieces of information from the Management Portal, as shown in Figure 9-4.

The screenshot shows a web form titled "Cache Properties" with four sections:

- Service URL:** A text box containing "proazure-1.cache.windows.net".
- Service Port:** A text box containing "22233".
- Secure Service Port:** A text box containing "22243".
- Authentication Token:** A text box containing "<Hidden>" and a "View" button.

Figure 9-4. Cache Properties in Management Portal

The first step is to create the cache endpoint(s), by following these steps:

1. Create an array of type `DataCacheServerEndpoint`. The number of elements depends on the number of endpoints. If you want both SSL and non-SSL endpoints, then initialize the array with a single element.
2. Initialize this element of the array with a new `DataCacheServerEndpoint`, passing in the Service URL and Port (from the management portal) as parameters.

■ **Note** The ability to specify multiple named `dataCacheClients` is only available when configuring via application configuration file¹. When configuring programmatically, only configure a single endpoint, and use the default constructor with no parameters when initializing the cache. Sample code for this that includes configuring for SSL or Non-SSL is included in Listing 9-3.

Next, we will need to set up the security properties of the data cache configuration factory in order to enable signing and/or encryption between the cache client and server. The `DataCacheSecurity` object is responsible for this. However, the `AuthenticationToken` must be passed into this object as a `SecureString` object, so convert the string and initialize the object as shown in lines 10-18 in Listing 9-3.

Then, create our `DataCacheFactoryConfiguration` class. Set the `Servers` property to the collection of `DataCacheServerEndpoint` objects that were created earlier. Also, set the `SecurityProperties` property to the `DataCacheSecurity` object that was created earlier as well.

Now, we can create a `DataCacheFactory` object, passing in the `DataCacheFactoryConfiguration` object we just created. Then finally, we create the cache client by calling the `GetDefaultCache()` method of the `DataCacheFactory` object. The `DataCache` object that is returned can be used to access the cache programmatically.

Listing 9-3. Programmatic Configuration of Cache Client

```
private DataCache InitializeCache(bool sslEnabled)
{
    string hostName = "[Service URL]";
    int cachePort;

    cachePort = sslEnabled ? 22243 : 22233; // Default port
    List<DataCacheServerEndpoint> servers = new List<DataCacheServerEndpoint>();
    servers.Add(new DataCacheServerEndpoint(hostName, cachePort));

    // Setup DataCacheSecurity configuration.
    string strAuthToken = "[Authentication Token from Portal]";
    var secureAuthToken = new SecureString();
    foreach (char a in strAuthToken)
    {
        secureAuthToken.AppendChar(a);
    }
    secureAuthToken.MakeReadOnly();
    DataCacheSecurity factorySecurity = new DataCacheSecurity(secureAuthToken);

    // Setup the DataCacheFactory configuration.
    DataCacheFactoryConfiguration factoryConfig = new DataCacheFactoryConfiguration();
    factoryConfig.Servers = servers;
    factoryConfig.SecurityProperties = factorySecurity;
}
```

¹ MSDN: `DataCacheFactoryConfiguration` Constructor (String): <http://msdn.microsoft.com/en-us/library/hh371022.aspx>

```
// Create a configured DataCacheFactory object.
DataCacheFactory cacheFactory = new DataCacheFactory(factoryConfig);

// Get a cache client for the default cache.
DataCache defaultCache = cacheFactory.GetDefaultCache();

return defaultCache;
}
```

Programming AppFabric Cache

Because the Azure AppFabric Cache is a subset of functionality from the Windows Server AppFabric Cache, there are only a few methods of importance, which are listed in Table 9-1. You will also find a code sample demonstrating usage of these methods in Listing 9-4.

Table 9-1. Key AppFabric Cache Methods

Method	Description
Add(string key, object value)	This adds a new object to the cache. It will throw an exception if the item already exists in the cache.
Put(string key, object value)	Replaces an object if it is already in the cache, or adds a new object if it doesn't already exist.
Get(string Key)	Returns an object from the cache.
Remove(string key)	Removes an object from the cache.

Listing 9-4. Programmatically Accessing the Cache

```
// Assume a dataCache object was already created using one of the techniques shown above.
// add a new object to the cache
dataCache.Add("cacheItem", "TestValue");
// Get this value
string initialValue = dataCache.Get("cacheItem");
// Update value in cache
dataCache.Put("cacheItem", "NewTestValue");
// Get the new value
string newValue = dataCache.Get("cacheItem");

// Compare the strings to ensure value was updated in the cache
int result = string.Compare(initialValue, newValue);

// remove the cache item
dataCache.Remove("cacheItem");
```

■ **Note** For the full list of configuration settings for ASP.NET 4 and what is supported, go to <http://msdn.microsoft.com/en-us/library/gg185682.aspx>.

Another great resource of information for AppFabric Cache (as well as many other Azure topics) is With Windows Azure Customer Advisory Team. You can find posts on caching techniques at <http://windowsazurecat.com/tag/caching/>.

Earlier in this chapter, I mentioned support for ASP.NET. The following sections provide guidance on implementing ASP.Net functionality using AppFabric caching: Session State and Page Output Caching.

ASP.NET Session State Provider

The Azure AppFabric Cache provides support for ASP.NET Session State via the AppFabric session state provider. As mentioned earlier, this has many advantages. However, the provider created for the Azure AppFabric also has several improvements over other ASP.NET session state providers.

For starters, it uses the `NetDataContractSerializer` class internally. This provides a wider range of types that can be serialized in and out of the cache, including binary serializable types. Also, it allows you to share session state among multiple ASP.NET applications. This could be extremely useful in many scenarios. It also avoids server-side request queuing by allowing multiple readers to concurrently access session state using read-only access, queuing only write requests.

And finally, and possibly most importantly, in a cloud environment where consumers are charged for the resources consumed, compression is supported.

Enabling Session State in AppFabric Cache

In addition to the work performed earlier to create the cache client, additional steps need to be taken to enable support for storing ASP.NET Session State in the AppFabric Cache. Specifically, the `<sessionState>` element needs to be added to the `web.config` file. If you are already using an ASP.NET session provider, replace with the code in Listing 9-5 to migrate.

Listing 9-5. *Web.config Code for Enabling ASP.NET Session State (from Management Portal)*

```
<!-- If session state needs to be saved in AppFabric Caching service, add the following to
web.config inside system.web. If SSL is required, then change dataCacheClientName to
"SslEndpoint". -->
<sessionState mode="Custom" customProvider="AppFabricCacheSessionStoreProvider">
  <providers>
    <add name="AppFabricCacheSessionStoreProvider"
      type="Microsoft.Web.DistributedCache.DistributedCacheSessionStateStoreProvider,
Microsoft.Web.DistributedCache"
      cacheName="default"
      useBlobMode="true"
      dataCacheClientName="default" />
  </providers>
```

```
</sessionState>
```

If you want compression enabled, you may need to modify the `dataCacheClient` element in your `web.config`:

```
<dataCacheClient name="default" isCompressionEnabled="true">
```

As you can see, there really isn't much work required to enable ASP.NET Session State caching in Azure AppFabric Cache. It provides a lot of value without much work to get it enabled.

Enabling ASP.NET Output Cache in AppFabric Cache

Output caching provides the ability for the server to cache http responses, and present those when requested instead of going through the full rendering life cycle. This can significantly increase performance of the web application when it has many typically static pages.

Using AppFabric Cache to store the output cache has some important benefits. Most important, the output cache data is not lost when the web application is recycled. This is particularly useful in the Azure environment, because the Fabric Controller can recycle a role instance at any time. Also, using the AppFabric cache takes load off of the web server. It frees memory that would have been used to hold the output cache data, which potentially frees memory for processing. In addition, it increases the effective limit of the output cache, because the AppFabric cache can support up to a 4 GB cache, which many web servers would not support. Finally, compression is supported here, as it is with session state. See the previous section for instructions on enabling compression on the `dataCacheClient`.

■ **Note** For the current release, only page output caching is supported. Web forms control output caching is not yet supported. The specifics of using output caching in ASP.NET are beyond the scope of this book. I will focus on enabling the cache here.

Enabling support for output caching through AppFabric cache is straightforward. Add the code from Listing 9-6 to your `web.config` file.

Listing 9-6. *Web.config Code for Enabling ASP.NET Output Caching (from Management Portal)*

```
<!-- If output cache content needs to be saved in AppFabric Caching service, add the following
to web.config inside system.web. -->
<caching>
  <outputCache defaultProvider="DistributedCache">
    <providers>
      <add name="DistributedCache"
        type="Microsoft.Web.DistributedCache.DistributedCacheOutputCacheProvider,
Microsoft.Web.DistributedCache"
        cacheName="default"
        dataCacheClientName="default" />
    </providers>
  </outputCache>
</caching>
```

Summary

In this chapter, you learned why the AppFabric Cache is an important part of the Windows Azure platform. You also learned how to create an AppFabric Cache and access it programmatically. And finally, you learned about enabling support for ASP.NET Session State and Output Caching.

Bibliography

MSDN: Caching Service (Windows Azure AppFabric): <http://msdn.microsoft.com/en-us/library/gg278356.aspx>

MSDN: using the ASP.NET 4 Caching Providers for AppFabric: <http://msdn.microsoft.com/en-us/library/gg185665.aspx>

SQL Azure

SQL Azure is Microsoft's relational database service in the cloud. In this chapter, we will cover SQL Azure in depth, starting with a look at how SQL Azure was architected, then leading into a discussion of what features are supported and what the limitations are. We'll discuss the different means of connecting to SQL Azure, and the impact of the distance between your application and SQL Azure. Then, we'll cover how to develop Windows Azure applications for SQL Azure, as well as some techniques for achieving efficient use of SQL Azure, and standard practices for database migration. Finally, we will briefly touch on two new technologies: SQL Azure Reporting and SQL Azure Data Sync. Throughout the chapter, we will be building an application that uses SQL Azure as well as other Windows Azure technologies such as Service Bus. It's a lot of ground to cover, so let's get started.

SQL Azure Overview

Any enterprise application, be it cloud or on-premise, is incomplete without the support of a back-end database. The database can be used to store business data, consumer data, or system data.

Applications are volatile, whereas databases are persistent. Front-end web applications usually depend on databases to persist business and system data. Therefore, databases can become the bottleneck in a system and need careful attention when you're architecting scalability, high availability, and performance for a system. You can scale-out front-end web applications by adding more load-balanced nodes, but to scale-out database servers, you need to not only scale out the database servers but also the storage on which these databases depend. On top of that, you have to make sure you aren't jeopardizing the high availability of the database server and its storage. Typically, on-premise databases use clustering techniques to provide high availability. Thus, scaling-out databases (typically via sharding) is an expensive effort in terms of the costs involved in scaling-out storage and database servers.

SQL Azure provides high availability to your databases out of the box. At any point in time, SQL Azure maintains three replicas of your databases in the cloud. If one replica fails, SQL Azure automatically creates a new one to maintain three replicas available at any point in time.

SQL Azure is based on the Microsoft SQL Server relational database engine. SQL Server is Microsoft's relational database, which is used by enterprises in their on-premise systems and is also offered as a hosted service by database hosting providers. With the launch of SQL Azure, Microsoft aims to offer a cloud relational database as a service for on-premise and cloud applications. When SQL Data Services (SDS) was launched at the Professional Developers Conference 2008, the service offering was an Entity-Attribute-Value (EAV) architecture with full scalability, fault tolerance, and high-availability features. Microsoft's vast partner and user community expressed the need for a relational database instead of a completely new EAV architecture because of the existing skill sets and applications that can be readily migrated to the cloud. Microsoft considered the feedback seriously and began the necessary work to replace the EAV architecture with traditional relational database features. In August 2009,

Microsoft announced the availability of the Community Technology Preview 1 (CTP 1) version of the SQL Azure relational database. The EAV capabilities were removed from the product, and only the relational database was made available. SQL Azure doesn't provide all the features available in SQL Server, but it does provide the bare minimum features required to deploy and maintain a database. For example, features like the Service Broker, Common Language Runtime (CLR) stored procedures, and HTTP endpoints aren't available in SQL Azure. This may change in the future, depending on customer demand.

As of the time of writing, Microsoft has released Community Technology Preview (CTP) version of SQL Data Sync, and Reporting Services. In future versions of SQL Azure, Microsoft plans to add other features that are missing from the SQL Azure platform.

■ **Note** This chapter assumes that you're familiar with SQL Server database concepts and that you can comfortably program TSQL SQL queries and data access using the ADO.NET API.

SQL Azure Architecture

SQL Azure is a scalable and highly available database utility service in the cloud. Like all other Windows Azure services, it runs in Microsoft data centers around the world. The data center infrastructure provides the SQL Azure service with load balancing, failover and replication capabilities. Figure 10-1 illustrates the high-level SQL Azure architecture.

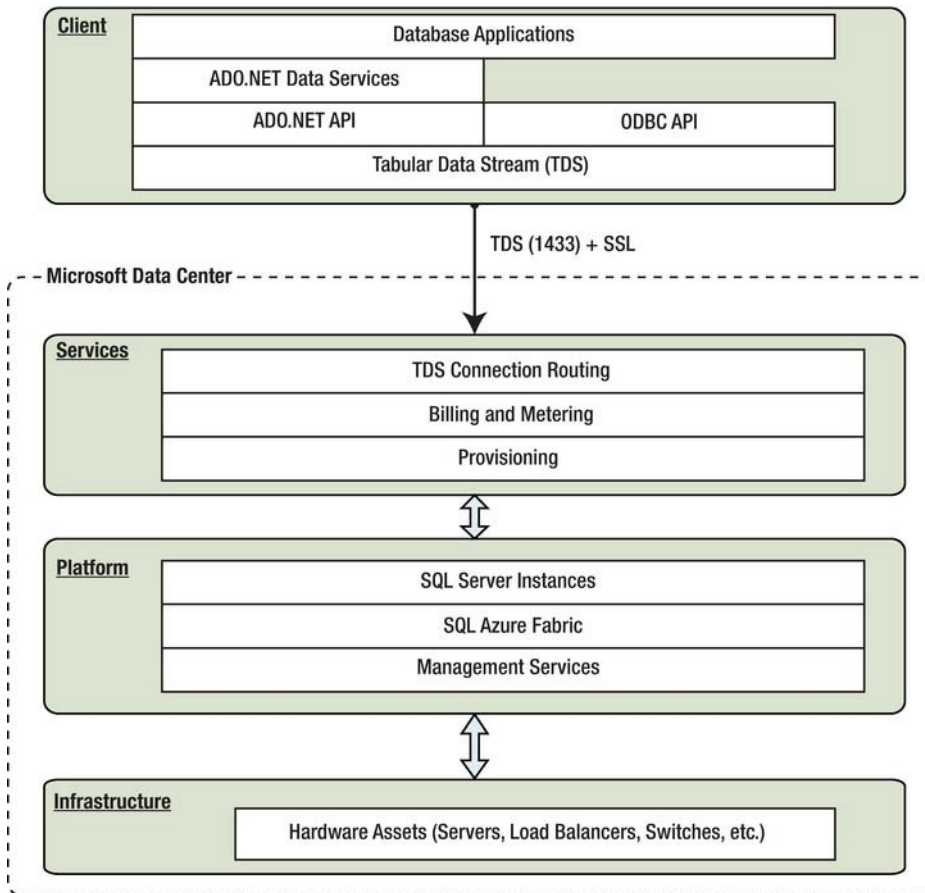


Figure 10-1. SQL Azure architecture

As shown in Figure 10-1, the SQL Azure service is composed of four layers: infrastructure, platform, services, and client. All the layers, except the client layer, run inside a Microsoft data center.

Infrastructure Layer

The infrastructure layer is the supporting layer providing administration of hardware and operating systems required by the services layer. This is the core data center layer that is shared across multiple services in a data center.

Platform Layer

The platform layer consists of the SQL Server instances and the SQL Azure fabric, and Management services. The SQL Server instances represent the deployed databases, their replicas, and the operating

system instances that host the SQL Server instances. The SQL Azure fabric is the underlying framework that automates the deployment, replication, failover, and load balancing of the database servers.

The SQL Azure fabric is responsible for creating three replicas of your database instance and provides automatic failover capabilities to these instances. As shown in Figure 10-2, if the primary instance of your database experiences a failure, the SQL Azure fabric designates one of the replicas as the primary instance and automatically routes all the communications to the new primary instance. In an effort to maintain three replicas at all times, SQL Azure also creates a new replica of the database.

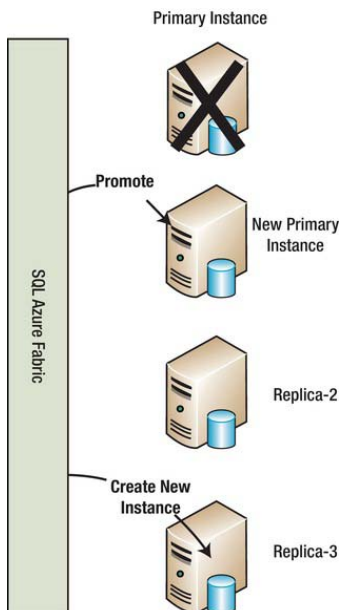


Figure 10-2. SQL Azure database replicas

The Management services are responsible for maintaining the health, upgrades, consistency, and provisioning of the hardware and software to support the SQL Azure fabric.

Services Layer

The services layer comprises external (customer) facing machines and performs as a gateway to the platform layer. It exposes the tabular data stream (TDS), billing, metering, and account provisioning services to customers.

■ **Note** TDS is the native Microsoft SQL Server protocol that database clients can use to interact with a SQL Server database. You can find the TDS protocol specification at [http://msdn.microsoft.com/en-us/library/dd304523\(prot.13\).aspx](http://msdn.microsoft.com/en-us/library/dd304523(prot.13).aspx).

The services layer exposes the TDS protocol on port 1433 over Secure Sockets Layer (SSL). The services layer is also responsible for routing connections to the primary database instance in the platform layer. This layer maintains runtime information about your database replicas and routes the TDS coming from client applications to the appropriate primary instance. The services layer is also responsible for provisioning your database when you create a database in SQL Azure. The provisioning of databases involves communicating with the SQL Azure fabric in the platform layer to provision appropriate replicas of the database.

The billing and metering service is responsible for monitoring the runtime usage of your database for billing purposes. The billing and metering service tracks the usage of databases at the account level.

Client Layer

The client layer is the only layer that runs outside of the Microsoft data center. The client layer doesn't include any SQL Azure-specific components; instead, it uses all the existing features of SQL Server client components like ADO.NET, ODBC, Visual Studio SQL Server Management Studio, ADO.NET Data Services, and so on. The client API initiates a TDS connection to SQL Azure on port 1433, which is routed by the services layer to the platform layer to the appropriate database instance.

SQL Azure Limitations and Supported Features

Even though SQL Azure is based on SQL Server, it includes some limitations because of its Internet availability and cloud deployment. When you use SQL Server on-premise, the tools and client APIs have full access to the SQL Server instance, and communications between the client and the database are in a homogeneous and controlled environment.

The first release of SQL Azure has only limited functionality of the SQL Server database. One of the most important limitations in SQL Azure is that fact that the size of the database can't exceed 50GB. So, as a database administrator or an architect, you must plan the growth and availability of data accordingly. The supported and unsupported features of SQL Azure in version 1.0 are described in the following sections.

■ **Note** Several important features were announced at the SQL PASS Conference in October 2011. The key announcements related to size limitations and increasing the maximum database size to 150GB by the end of 2011, and SQL Azure Federation, which allows you to elastically scale out your database beyond 150GB using the sharding database pattern. Cihan Biyikoglu's blog is very useful for providing more information about Federation and SQL Azure in general: <http://blogs.msdn.com/b/cbiyikoglu/>.

Avkash Chauhan's blog is also a great source of information on SQL Azure. This post in particular is located at: <http://blogs.msdn.com/b/avkashchauhan/archive/2011/10/13/sql-azure-databases-will-be-expanded-3x-from-50-gb-to-150-gb-and-sql-azure-reporting-amp-sql-azure-data-sync-ctp.aspx>.

Database Features

SQL Azure supports the following database features:

- CRUD operations on tables, views, and indexes
- TSQL query JOIN statements
- Triggers
- TSQL functions
- Application stored procedures (only TSQL)
- Table constraints
- Session-based temp tables
- Table variables
- Local transactions
- Security roles
- Spatial data types: geography and geometry

SQL Azure does *not* support the following database features:

- Distributes query
- Distributed transactions
- Any TSQL query and views that change or retrieve physical resource information, like physical server DDL statements,¹ Resource Governor, and file group references

Application Features

SQL Azure does *not* support the following application-level features:

- Service Broker
-

¹ SQL Azure Team Blog: <http://blogs.msdn.com/ssds/default.aspx>

- HTTP access
- CLR stored procedures

Administration Features

SQL Azure supports the following administration features:

- Plan and statistics
- Index tuning
- Query tuning

SQL Azure does *not* support the following administration features:

- Replication
- SQL profiler
- SQL trace flag
- Backup command
- Configuration using the `sp_configure` stored procedure

■ **Note** The SQL Azure SDK documentation lists all the other limitations that aren't covered in this section. See <http://msdn.microsoft.com/en-us/library/ee336245.aspx>.

SQL Azure Data Access

SQL Azure allows you to connect to the cloud database only using the TDS protocol with limited support, as described in the previous section. But because the TDS protocol is supported by most of the SQL Server client APIs, all the features supported by SQL Azure work with existing client APIs. You can use two common patterns to connect to SQL Azure databases: code near and code far.

Code-Near Connectivity

In *code-near* connectivity, your application is deployed in Windows Azure, which uses SQL Azure. You geo-locate both of them in the same data center by configuring the geo-location features of Windows Azure and SQL Azure. Figure 10-3 illustrates applications with code-near connectivity to a SQL Azure database.

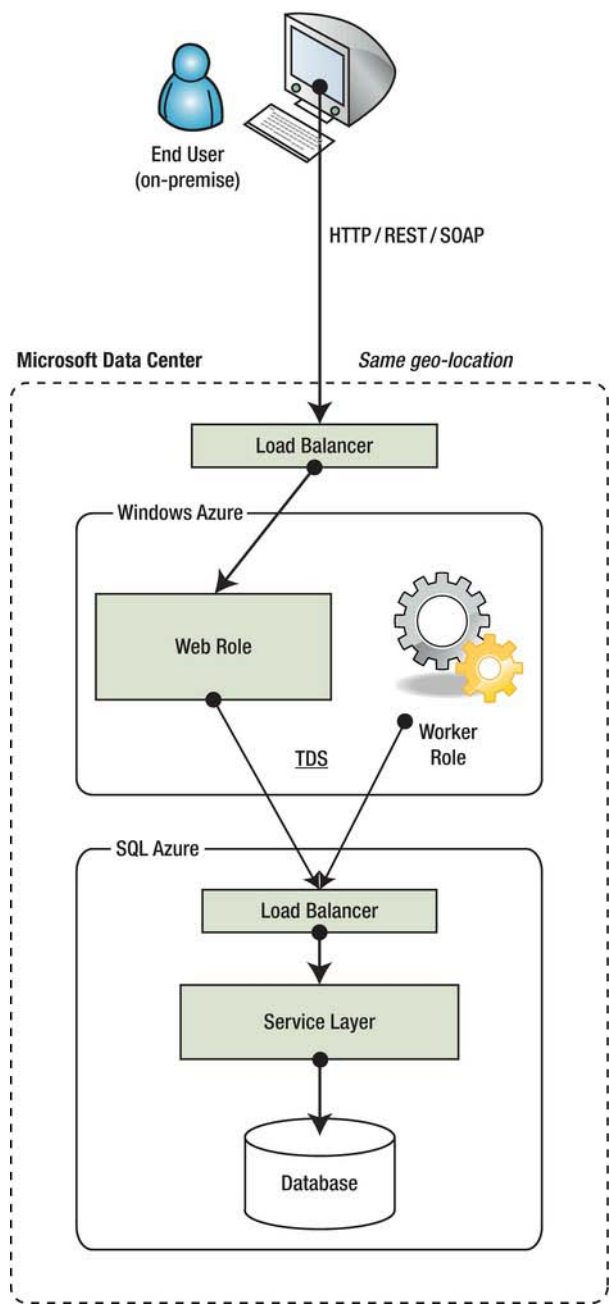


Figure 10-3. Code-near connectivity to SQL Azure

In a typical code-near architecture, the data access application is located in the same data center as the SQL Azure database. The end users or on-premise applications access the web interface are exposed via a Windows Azure web role. This web role may be hosting an ASP.NET application for end users or a web service for on-premise applications.

The advantages of the code-near approach are as follows:

- Business logic is located closer to the database.
- You can expose open standards-based interfaces like HTTP, REST, SOAP, and so on to your application data.
- Client applications don't have to depend on the SQL Server client API.

The disadvantage of this approach is the performance impact your application experiences if you're using Windows Azure as a middle tier to access the database.

Code-Far Connectivity

In *code-far* connectivity, your application is typically deployed on-premise or in a different data center than SQL Azure. In this pattern, the client application makes a SQL query using the TDS protocol over the Internet to the SQL Azure database. Figure 10-4 illustrates applications with code-far connectivity to a SQL Azure database.

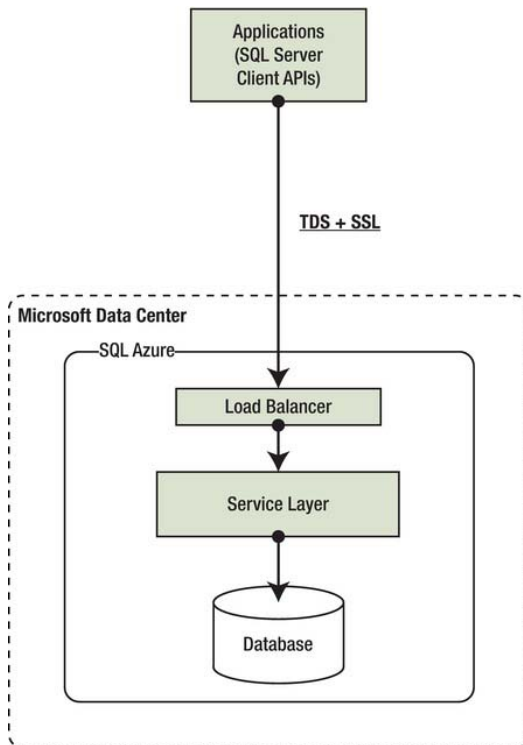


Figure 10-4. Code-far connectivity to SQL Azure

The biggest advantage of the code-far approach is the performance benefit your application can experience because of direct connectivity to the database in the cloud. The biggest disadvantage is that all the client applications must use the TDS protocol to access the database. Therefore, the data access clients must use SQL Server-supported client APIs like ADO.NET, ODBC, and so on, reducing data-access possibilities from APIs or platforms that don't support the TDS protocol.

Getting Started with SQL Azure

SQL Azure is a core component of the Windows Azure platform. Like all other Windows Azure components, administration is originally performed through the Azure Management Portal (shown in Figure 10-5). Go to <https://windows.azure.com>, and you will see a Database navigation item in the lower part of the left-hand navigation frame. Clicking this link will take you to your SQL Azure Management Portal. This is where you will create servers and manage administrators and firewalls.

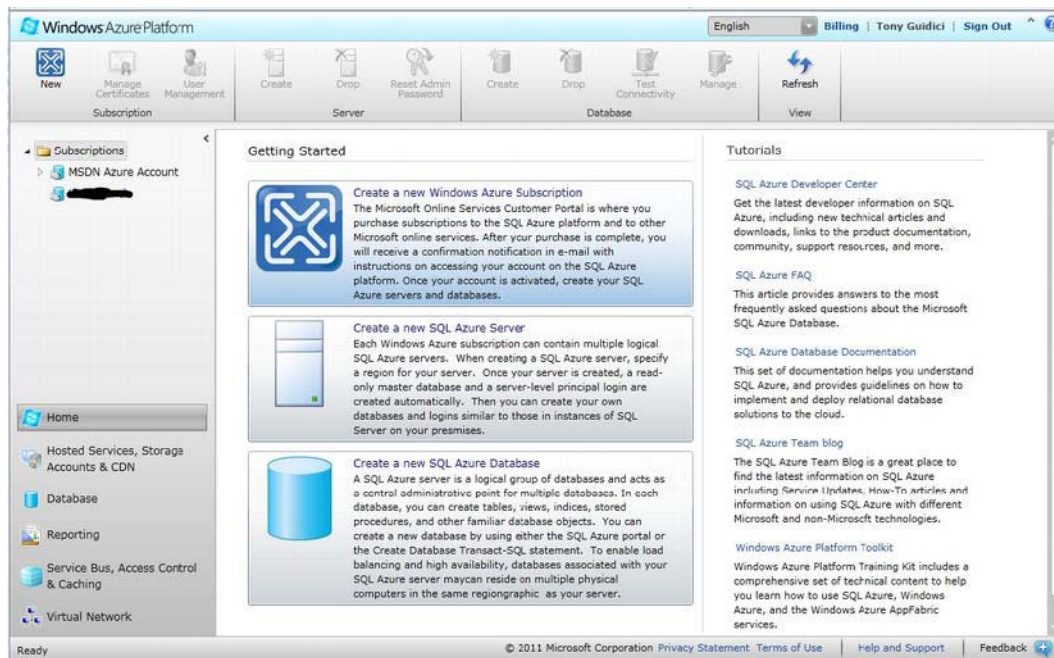


Figure 10-5. SQL Azure Management Portal

■ **Note** The links on the right side of the portal contain some very useful resources.

Creating a SQL Azure Server

The first thing to do is create a server to host your databases. Click “Create a new SQL Azure Server” and you will be taken through a wizard. Fill in the following information:

1. *Select a subscription:* If you happen to have more than one Azure subscription, choose the one for which you want to be billed.
2. *Region:* Select the geographic region in which the server will be provisioned.
3. *System Administrator login:* choose a username and password for your system administrator account. (Note that logins such as ‘sa’ and other typically insecure logins will not be allowed.)
4. *Firewall rules:* Add IP addresses for which you want to provide access to the server. This will be required for any on-premise IP addresses or ranges that you want to have access to the server directly. If all of your access will be through the portal or Database Manager, then you don’t need to add anything. Also of importance is the check box “Allow other Windows Azure services to access this portal.” This is required if you plan to access the database from an application deployed in Azure. Since the IP addresses of the role instances are dynamic, the platform manages this access for you. You also have the opportunity to modify these settings later in the portal.

■ **Tip** Rather than creating the entire database in the cloud, it may be simpler to use SQL Express to create the database schema, and then use a tool like SQL Azure Migration Wizard from codeplex to migrate the structure to SQL Azure. If you are migrating an existing database, it can even move the data for you. Check out <http://sqlazuremw.codeplex.com>

Once created, your portal should show the screen in Figure 10-6.

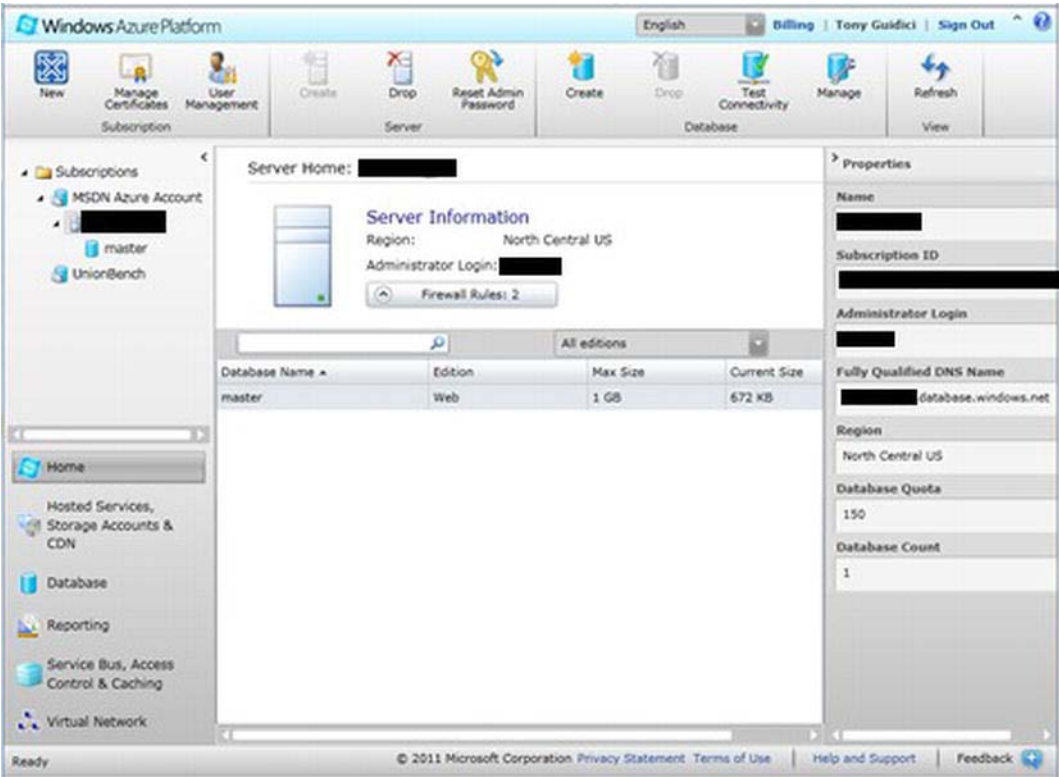


Figure 10-6. SQL Azure portal, Server view

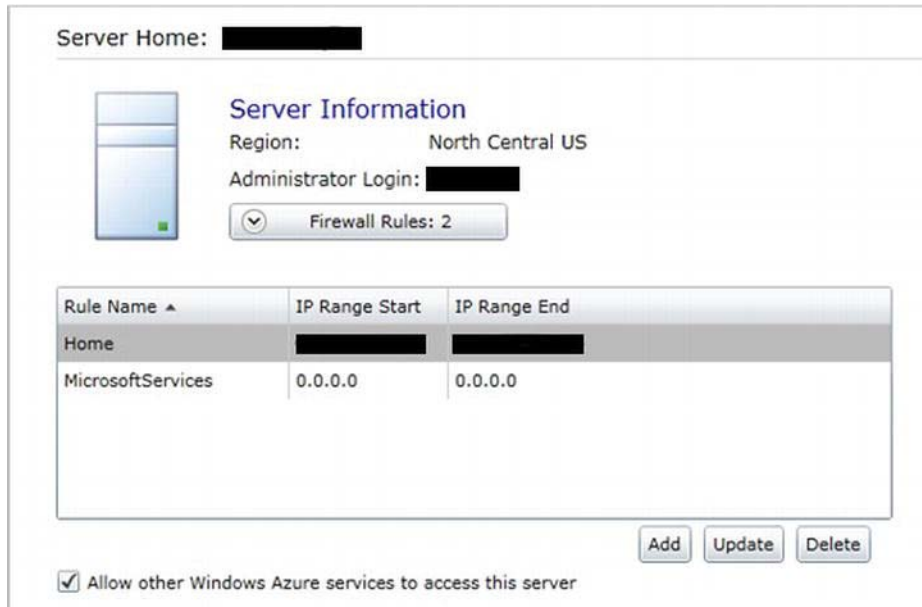
On the left will be a tree view of your subscriptions, servers and databases. On top is a set of buttons—grouped by Server and Database—that allow you to manage your server and databases. On the right, the properties for the server are displayed. This provides you with some key information about your server, such as the fully qualified DNS name, which is essential when building connection strings. It also tells you the number of databases permitted for the server, as well as how many you have created.

The key functionality is in the middle pane. By clicking on a database row in the lower pane, you can select a database to manage, which will be covered later. On the top half of the middle pane is where you manage the firewall rules for the server (see Figure 10-7).

By default, SQL Azure blocks all IP traffic to the server except that which comes through the portal itself. If you intend to use tools or applications that will be accessing this SQL Azure server, you will need to add a firewall rule. Otherwise, access to the server will be denied.

Click the Firewall Rules button in the middle, and the panel will appear showing the current set of firewall rules (see Figure 10-8). Note the option to allow other Windows Azure services access to this server. Again, this provides Azure role instance access to the SQL Server through the firewall.

■ **Note** In my work with customers and partners at Microsoft, forgetting to configure the firewall rules is one of the most common issues. If you are having problems connecting to SQL Azure, check the firewall rules first. If you can connect on-premise or in the Compute Emulator, but get connection errors when deployed to the Azure environment, make sure the 'Allow Windows Azure services' checkbox is selected.



The screenshot shows the 'Server Home' page for a SQL Azure instance. At the top, it says 'Server Home: [redacted]'. Below this is a 'Server Information' section with a server icon, the region 'North Central US', and the administrator login '[redacted]'. A button labeled 'Firewall Rules: 2' is also present. The main part of the page is a table of firewall rules:

Rule Name ▲	IP Range Start	IP Range End
Home	[redacted]	[redacted]
MicrosoftServices	0.0.0.0	0.0.0.0

At the bottom right of the table are buttons for 'Add', 'Update', and 'Delete'. Below the table is a checkbox labeled 'Allow other Windows Azure services to access this server' which is checked.

Figure 10-7. Firewall rules

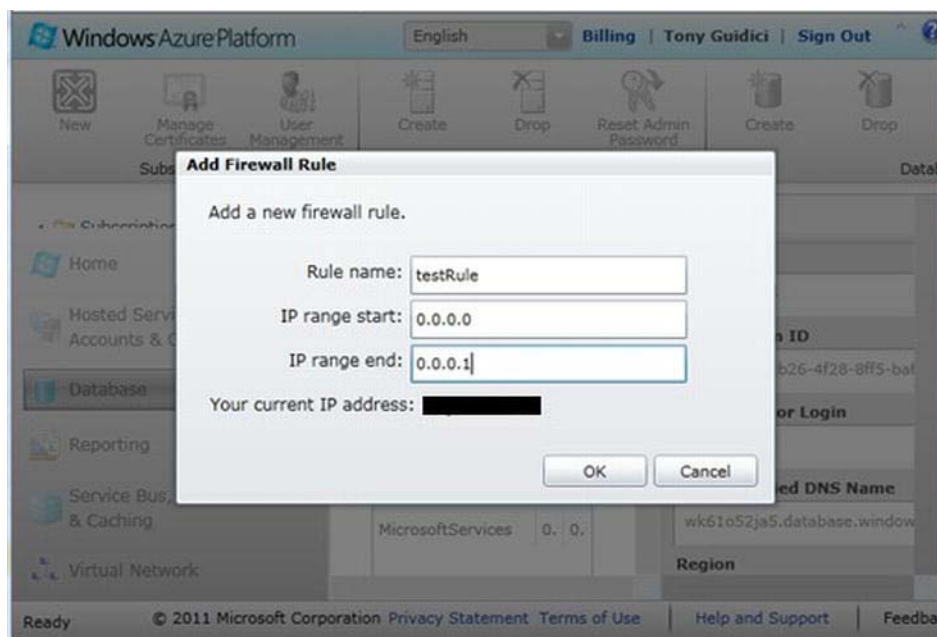


Figure 10-8. Adding a firewall rule

Creating a SQL Azure Database

To create a new database, follow these steps:

1. Click the Create Database button at the top of the portal (see Figure 10-9).

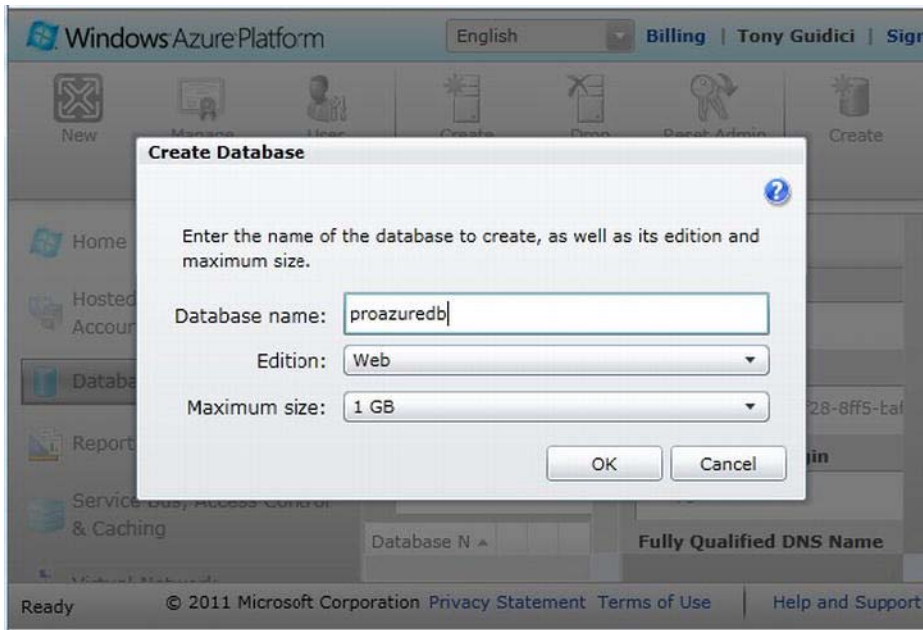


Figure 10-9. Creating a database: initial settings

2. Create DatabaseName the database “proazuredb” and click Create. The proazuredb database shows up in the database list on the portal page and in the tree view on the left. Clicking the database name in the left navigation will bring up the database man page. This is where you can view database properties such as connection strings. Clicking the ellipse button on the right will provide you with connections strings for your database, as shown in Figure 10-10.

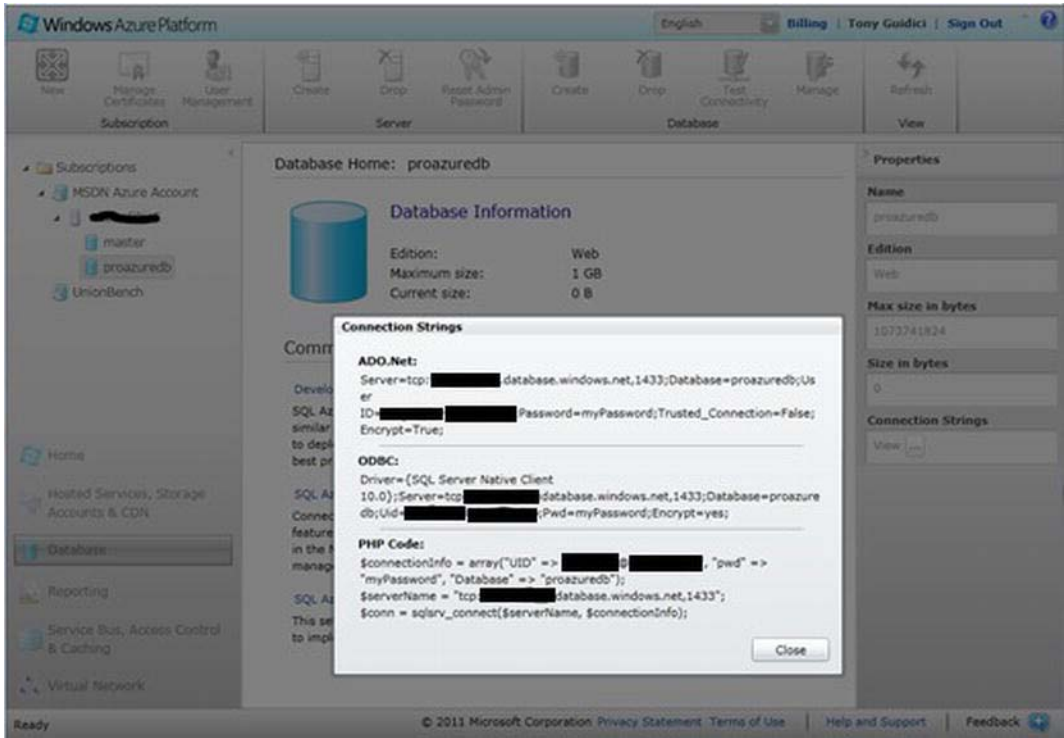


Figure 10-10. Proazuredb connection strings

Note that all the databases in the same project have the same master database and administrator. If you want to drop the database, you can click the Drop button.

Connecting to a SQL Azure Database

After the database is created, you can connect to it from anywhere with SQL Server client capabilities. You can connect to the SQL Azure database in the following four ways:

- SQL Server Management Studio
- Database Manager portal application
- SQLCMD
- ADO.NET

Connecting Using SQL Server Management Studio

The steps to connect to the SQL Azure database are as follows:

1. Open SQL Server Management Studio.

2. Choose Start ► All Programs ► SQL Server 2008 ► SQL Server Management Studio.

■ **Note** In the current CTP, click the Cancel button on the Login dialog box.

3. Click the New Query button in SQL Server Management Studio (see Figure 10-11).

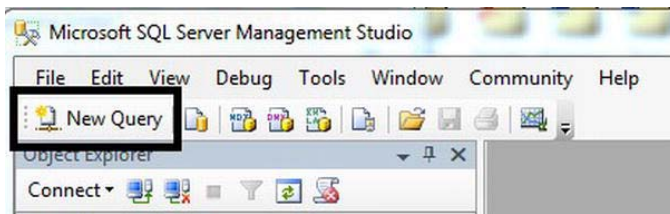


Figure 10-11. New Query window

4. A new login dialog appears (see Figure 10-12). Enter the SQL Azure server name and administrator username/password that you created while provisioning the database. The format for the server name is {your server name}. database.windows.net, where {your server name} is the name of the server assigned to your database during provisioning. You can get it from the Server Administration page on the SQL Azure portal.



Figure 10-12. Database login

5. If you want to connect to a specific database, click the Options button and enter the database name (such as **proazuredb**) that you want to connect to (see Figure 10-13). If you don't choose a database name, you're connected to the master database by default.



Figure 10-13. Enter a database name

6. Keep the database name set to default, and click Connect to connect to the master database. If the connection is successful, a new query window opens, as shown in Figure 10-14.

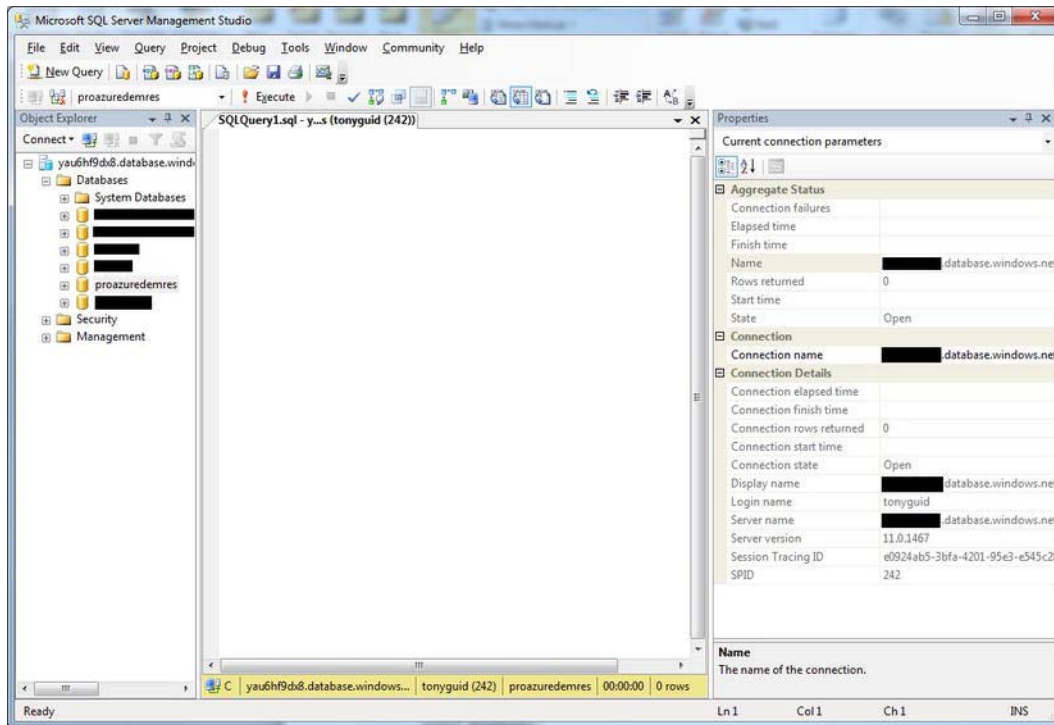


Figure 10-14. New Query window

- Now you're connected to the SQL Azure cloud master database. Type the following queries in the query window, and click Execute:

```
select * from sys.databases;
select @@version;
select @@language;
```

- The first query returns the list of databases, @@version, returns the version of the database server, and @@language returns the database language currently in use. As shown in Figure 10-15, the query returns two databases: master and proazuredb. The master database is created by the system during provisioning, but the proazuredb is the user created database.

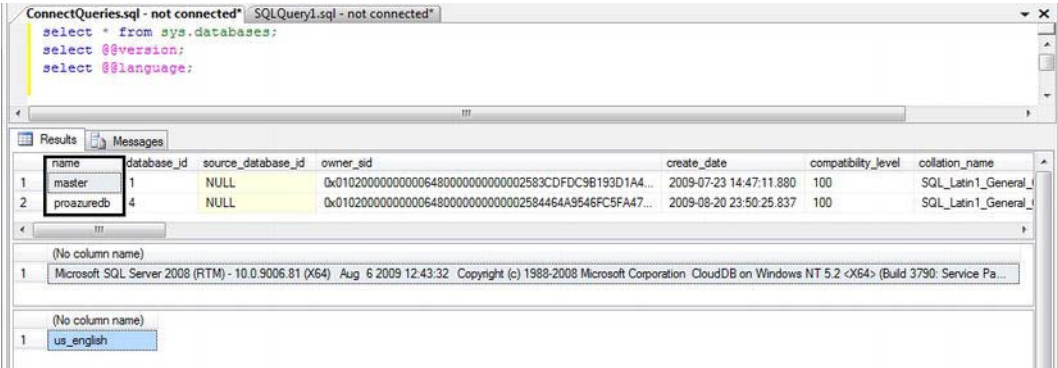


Figure 10-15. Execute queries

9. Create a new database named MyCloudDb. Execute the following query in the query window to create the MyCloudDb database:

```
CREATE DATABASE MyCloudDb;
```
10. When you execute the CREATE DATABASE statement, SQL Azure creates a new database in the cloud. Note that you don't have to worry about the location of the data files because SQL Azure abstracts the location of the data files from you.
11. Execute the "select * from sys.databases" query again. You see the new database in the list of databases returned by the query (see Figure 10-16).

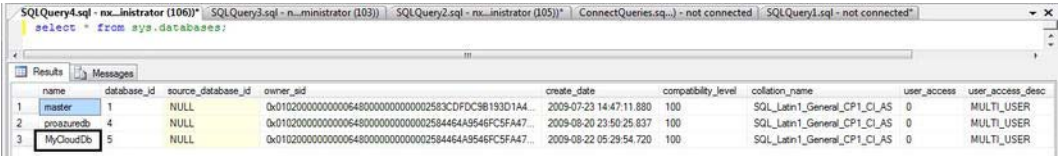


Figure 10-16. List the new database

Creating Logins

From SQL Server Management Studio, you can execute common administration SQL statements like creating logins and users, and assigning users to database roles. To create a new login, you have to first create a new login in the master database, and then create a new user for the login in the MyCloudDb database, and finally add the new user to one of the database roles using the system stored procedure sp_addrolemember:

1. Connect to the master database using SQL Server Management Studio. Make sure you set the database name to master in the Connection Properties tab, as shown in Figure 10-17.



Figure 10-17. Set the master database in Connection Properties

2. Create a new login named test user by executing the following query in the query window (see Figure 10-18).

■ **Tip** Use your own password in the query.

```
CREATE LOGIN testuser WITH PASSWORD = 'pas@word1'
```

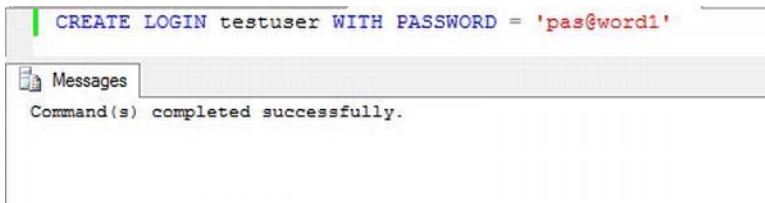


Figure 10-18. Create a new login

3. Connect to the MyCloudDb using your administrator account and by typing MyCloudDb in the Connection Properties tab, as shown in Figure 10-19.



Figure 10-19. Set the database to MyCloudDb in Connection Properties

4. After the connection is successful, type the following query to create a new user test user for the new login in MyCloudDb (see Figure 10-20):

```
CREATE USER testuser FOR LOGIN testuser;
```

■ **Note** You cannot simply use the ‘use {database}’ statement to switch between databases. It is not supported in SQL Azure. You must create a connection to each database independently.



Figure 10-20. Create a new user for the login

5. After you add the user to the database, you have to add the user to a particular database role. Figure 10-21 illustrates the default roles in SQL Server.

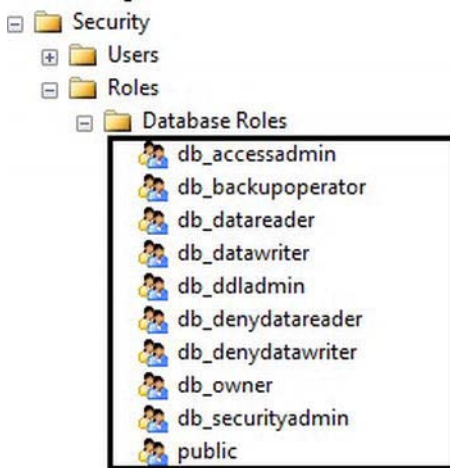


Figure 10-21. Default SQL Server roles

6. Add the test user to the db_owner group by executing the following query:
EXEC sp_addrolemember 'db_owner', 'testuser'

■ **Tip** In real-world applications, don't add users to the db_owner role because the db_owner role has extensive privileges to the database.

7. Connect to MyCloudDb using the newly created test user, as shown in Figure 10-22.



Figure 10-22. Connecting to MyCloudDb as testuser

Now you don't need to log in to the database with the administrator user; instead you can log in as test user. From SQL Server Management Studio, you can also execute data definition (DDL) and data manipulation (DML) commands like CREATE TABLE, UPDATE TABLE, INSERT, DELETE, and so on. I cover DDL and DML in a later section with an example.

Connecting Using Database Manager

In this section I will show you how to connect to and manage your SQL Azure database from the Database Manager online tool. From the main SQL Azure portal, select a database, and click the 'Manage' button as shown in Figure 10-23. This will bring you to a Silverlight application you can use to manage your database.

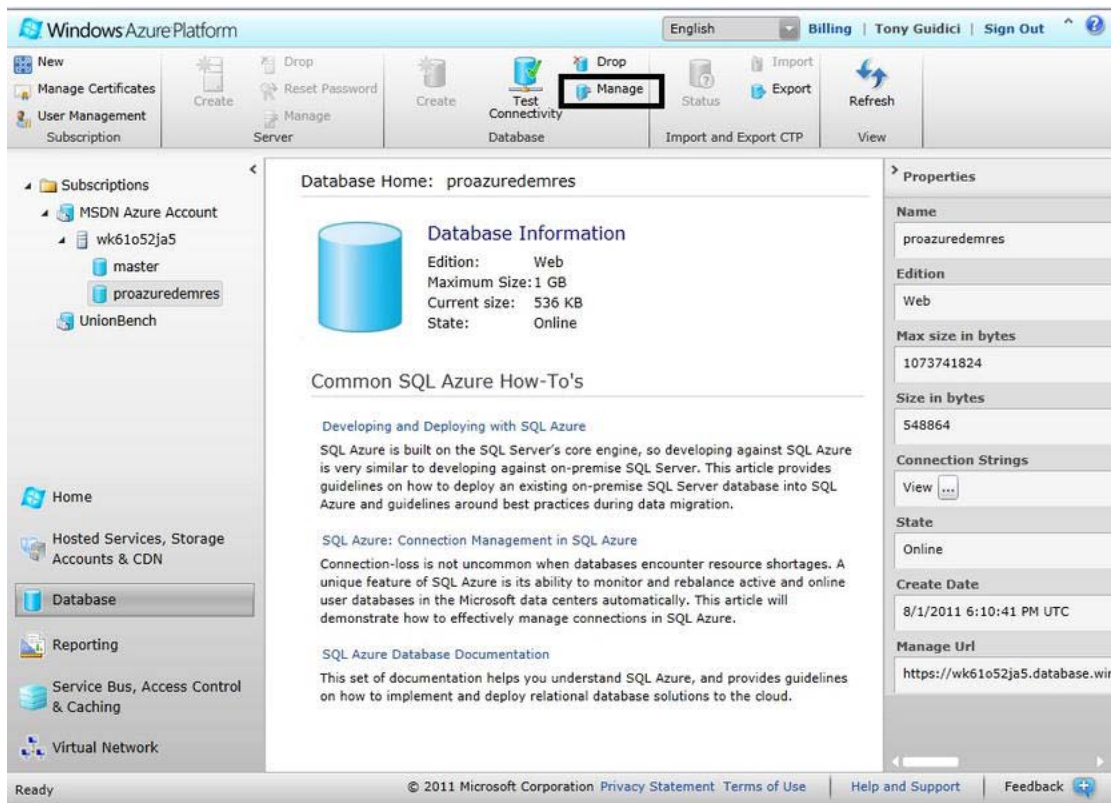


Figure 10-23. Accessing Database Manager from the Azure portal

From there, you will be taken to a login screen where you must enter your SQL credentials. Once entered, you will be taken to the Database Manager application, as shown in Figure 10-24.

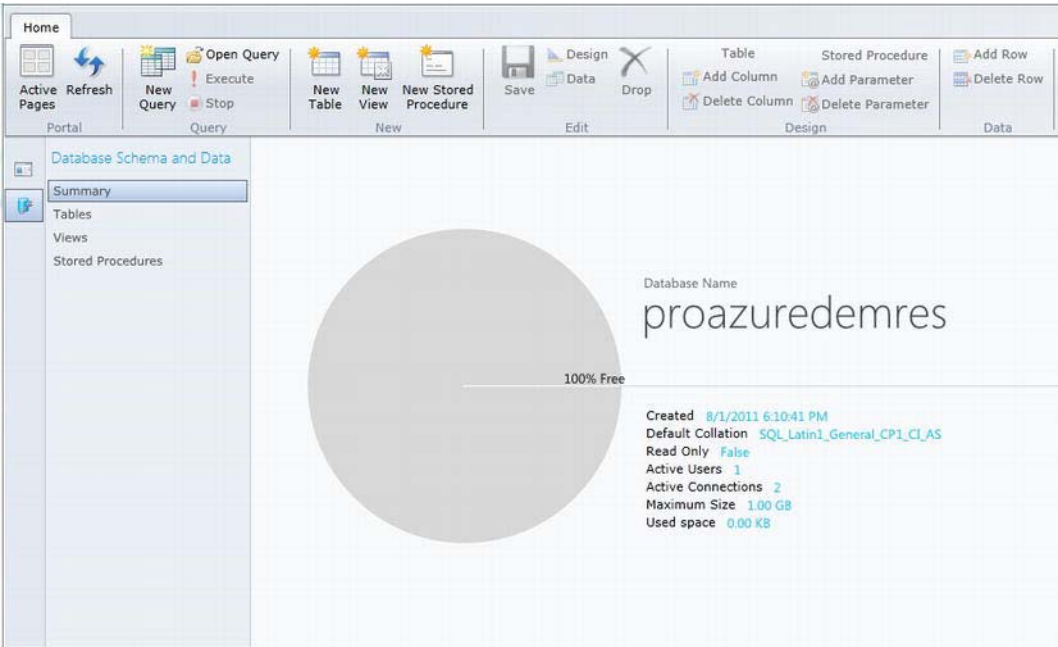


Figure 10-24. Database Manager application

From here, you have the ability to perform many common database tasks. Clicking New Query gives you the ability to create and execute ad-hoc queries, as shown in Figure 10-25. In addition to ad-hoc queries, this is where you would execute SQL commands for tasks such as creating logins and users.

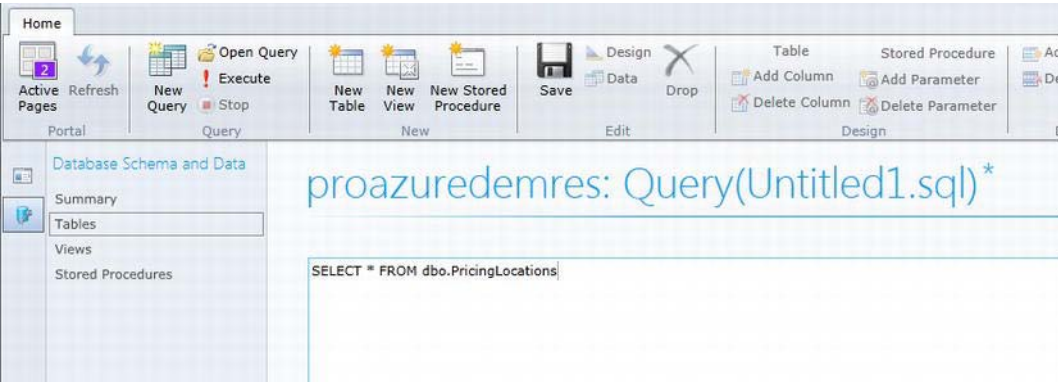


Figure 10-25. Creating an ad-hoc query

Clicking ‘Tables’ in the left-hand navigation brings up a list of tables in your database. Clicking each row in the list provides you with the option of viewing the data in the table, or modifying the design, as

shown in Figure 10-26. In addition you can create new tables by clicking the New Table button. The UI for creating/modifying tables is shown in Figure 10-27.

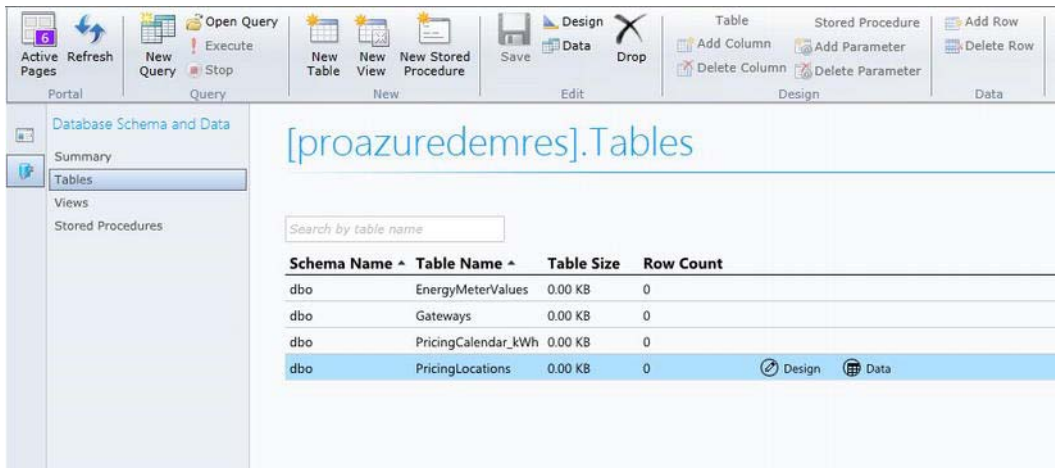


Figure 10-26. Database Manager table listing

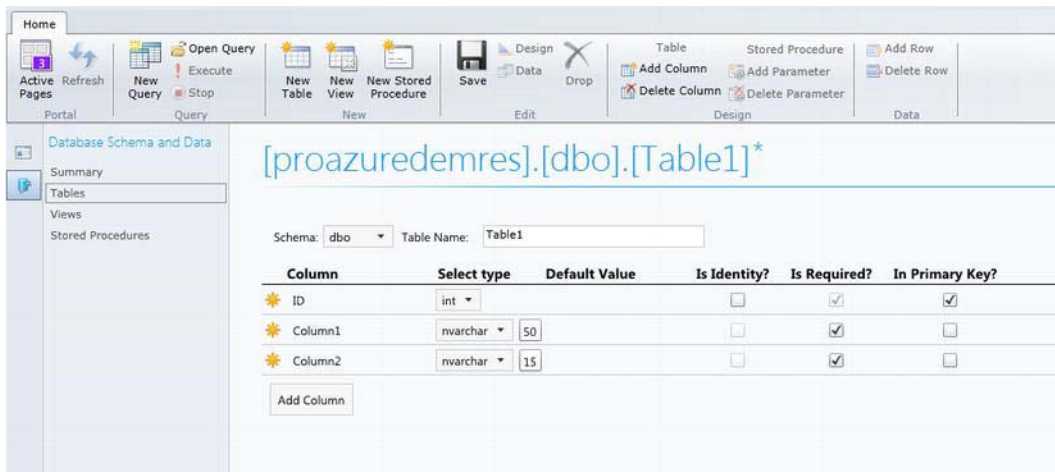


Figure 10-27. Creating a table

Creating a stored procedure is also possible. Simply click New Stored Procedure and you will be presented with a user interface that allows you to write a stored procedure and add parameters, as shown in Figure 10-28.

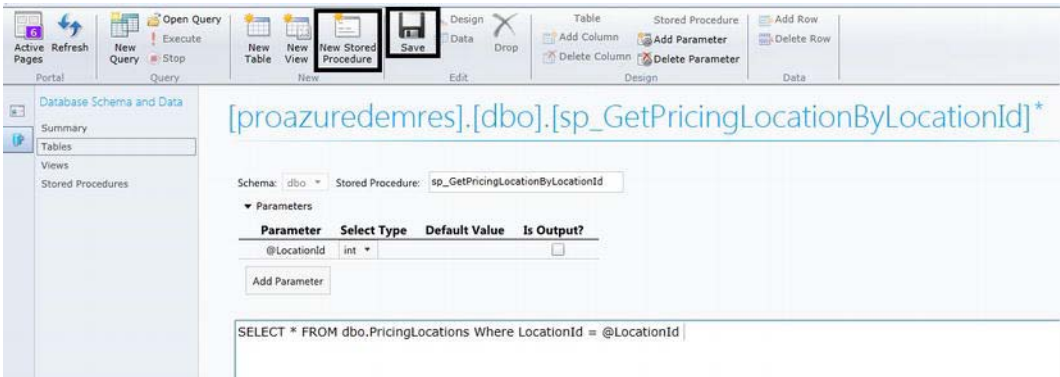


Figure 10-28. Creating a stored procedure

One last thing to note is that Database Manager tries to be a Windows-type application, in that each of the user interface pages you open remains open until you explicitly close it. If you want to see which pages are open, or navigate to one of these pages while you are working on another, click the Active Pages button, and you will see all the pages you have open, as shown in Figure 10-29.

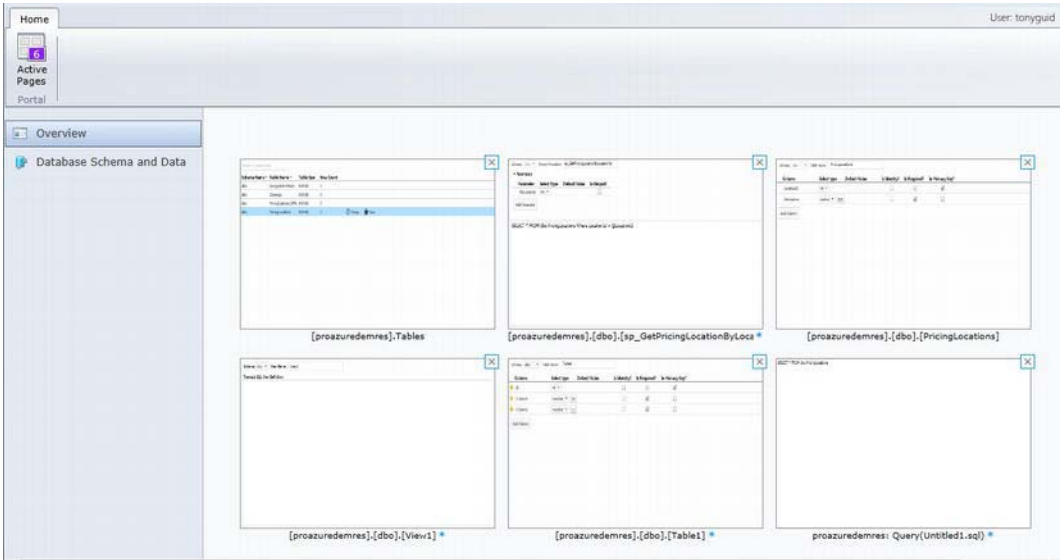


Figure 10-29. Active Pages

Connecting Using SQLCMD

SQLCMD.exe is a command-line utility used to execute commands on SQL Server. It comes with the SQL Server installation. SQL Server Management Studio is a good user interface tool to connect to SQL Azure; but in real-world production environments where automation is heavily used in administering SQL

Servers, SQLCMD is the preferred tool. You can automate the execution of SQL command by scheduling SQLCMD. It accepts inline SQL as well as scripts files as input.

The steps to connect to SQL Azure using SQLCMD are as follows:

1. Open a command prompt as administrator, as shown in Figure 10-30.

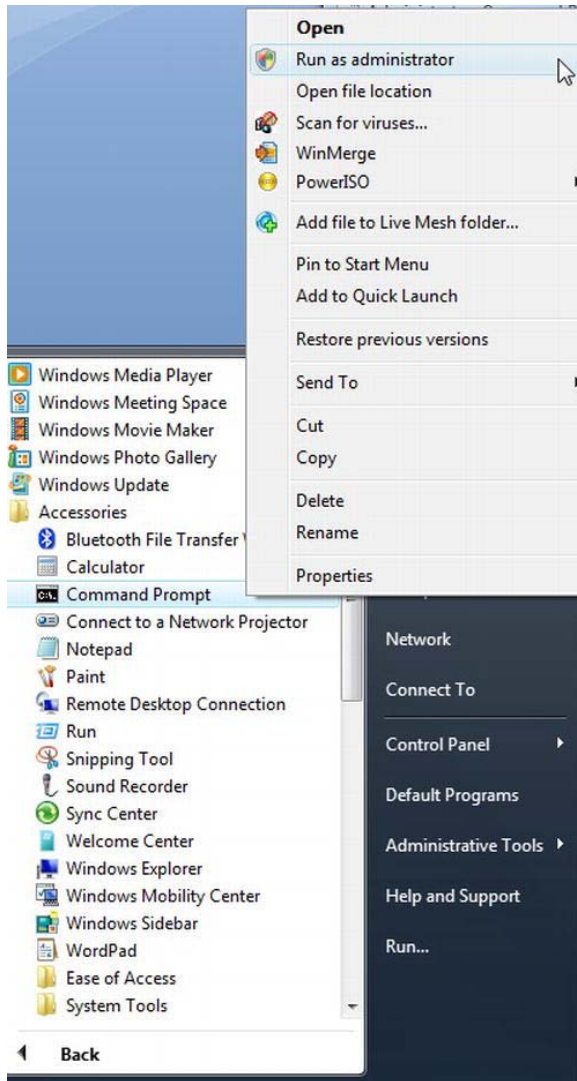


Figure 10-30. Open a command prompt

2. The syntax to connect to a database using sqlcmd.exe is as follows (it should be entered as a single line):

```
sqlcmd -U <userlogin@servername>
-P <password> -S <Fully Qualified ServerName> -d <database name>
```

3. *userlogin* is the user name you created for the database. In the previous example, it's either administrator or test user, if you're connecting to the MyCloudDb database. *servername* is the server name from your Server Administration page in the developer portal. Don't provide the fully qualified server name. *password* is the password for the login. *Fully Qualified ServerName* is the server name appended by the fully qualified name of the SQL Azure server (servername.ctp.database.windows.net).
4. Execute the following command on the command line to connect to MyCloudDb (see Figure 10-31). Replace the server name with your own server name:

```
sqlcmd -U testuser@nx8qpedcoo -P pas@word1
-S nx8qpedcoo.ctp.database.windows.net -d MyCloudDb
```

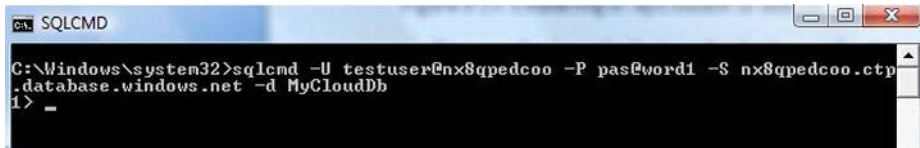


Figure 10-31. Connect to MyCloudDb

5. If the connection is successful, you see “1>” displayed on the command prompt.
6. Execute the following command to create a table in MyCloudDb (see Figure 10-32):

```
CREATE TABLE CloudTable
(ColNumber1 int primary key clustered, ColNumber2 varchar(50), ColNumber3 float);
GO
```

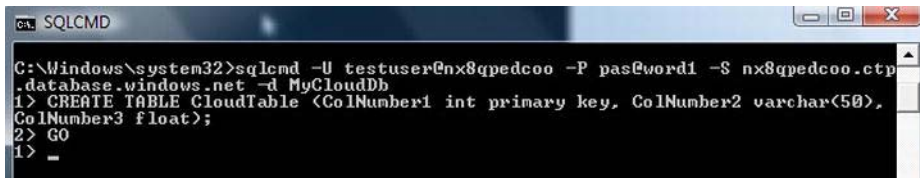
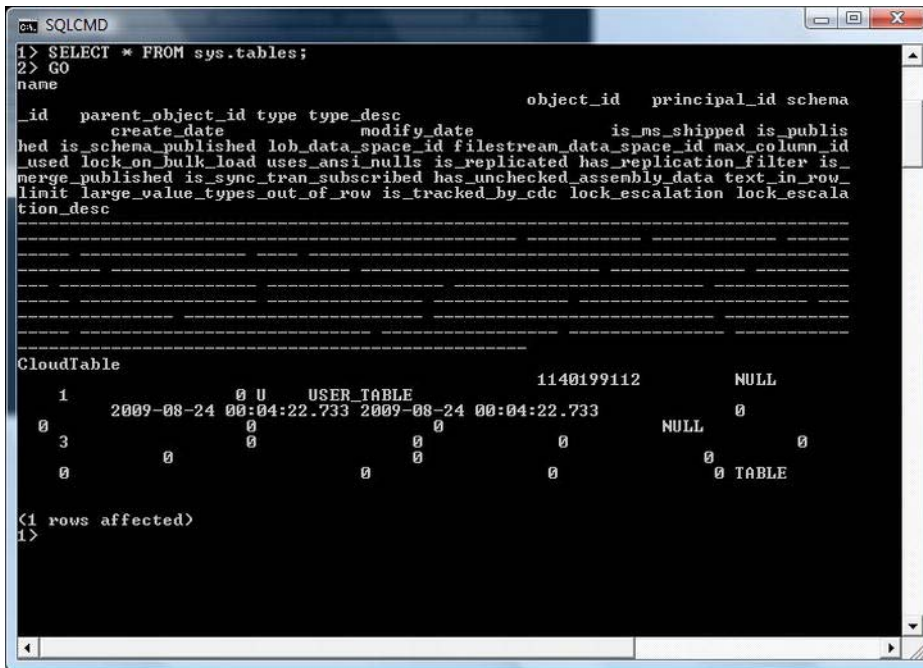


Figure 10-32. Create Table

7. Execute the following command to get the information about the tables in the MyCloudDb database (see Figure 10-33).

```
SELECT * FROM sys.tables
```



```

C:\> SQLCMD
1> SELECT * FROM sys.tables;
2> GO
name
-----
id      parent_object_id type type_desc      object_id  principal_id schema
create_date      modify_date      is_ms_shipped is_public
hed is_schema_published lob_data_space_id filestream_data_space_id max_column_id
used lock_on_bulk_load uses_ansi_nulls is_replicated has_replication_filter is_
merge_published is_sync_tran_subscribed has_unchecked_assembly_data text_in_row
limit large_value_types_out_of_row is_tracked_by_cdc lock_escalation lock_escal
tion_desc

-----
CloudTable
1          0 0  USER_TABLE          1140199112      NULL
2009-08-24 00:04:22.733 2009-08-24 00:04:22.733      0
0 3          0          0          0      NULL      0
0          0          0          0          0      0 TABLE
<1 rows affected>
1>

```

Figure 10-33. Select all tables

■ **Note** You can find more information about the SQLCMD command-line utility in SQL Server Books online at <http://msdn.microsoft.com/en-us/library/ms162773.aspx>. In this chapter, I use SQL Server Management Studio to connect to the SQL Azure database manually; but in an environment where you need scripting to automate SQL Server tasks, you should use SQLCMD.

Connecting Using ADO.NET

ADO.NET is the most popular method to connect to an on-premise SQL Server programmatically. SQL Azure supports ADO.NET connectivity similar to an on-premise SQL Server. To open a connection to SQL Azure database using ADO.NET, you have to pass the connection string of the database acquired from the Server Administration page of the SQL Azure developer portal; or, you can programmatically build a connection string using the `System.Data.SqlClient.SqlConnectionStringBuilder` class, as shown in Listing 10-1.

Listing 10-1. *Build a Connection String Using SqlConnectionStringBuilder*

```
private static string GetUserDbString()
{
    // Create a connection string for the sample database
    SqlConnectionStringBuilder connString2Builder =
        new SqlConnectionStringBuilder();
    string server = "yourservername.ctp.database.windows.net";
    connString2Builder.DataSource = server;
    connString2Builder.InitialCatalog = "user database";
    connString2Builder.Encrypt = true;
    connString2Builder.TrustServerCertificate = true;
    connString2Builder.UserID = "userName";
    connString2Builder.Password = "pass@word1";
    return connString2Builder.ToString();
}
```

The `SqlConnectionStringBuilder` class is used to build the string value of the SQL Azure database connection string. Note that the `Encrypt` and `TrustServerCertificate` properties are set to true, which is a best practice in general for connecting to databases in the cloud or in another domain.

After the connection string is constructed, you can connect to the SQL Azure database by opening a `SqlConnection` to the database. Listing 10-2 shows a series of database queries executed on a SQL Azure database after the connection is successfully established.

Listing 10-2. *SqlConnection and Query Execution*

```
using (SqlConnection conn = new SqlConnection(GetUserDbString()))
{
    using (SqlCommand command = conn.CreateCommand())
    {
        conn.Open();
        // Create table
        command.CommandText =
            "CREATE TABLE MyTable1(Column1 int primary key clustered, " +
            "Column2 varchar(50), Column3 datetime)";
        command.ExecuteNonQuery();
        // Insert records
        command.CommandText = String.Format
            ("INSERT INTO MyTable1 (Column1, Column2, Column3) " +
            "values ({0}, '{1}', '{2}')" , 1, "TestData", DateTime.Now.ToString("s"));
        int rowsAdded = command.ExecuteNonQuery();
        DisplayResults(command);
        // Update a record
        command.CommandText =
            "UPDATE MyTable1 SET Column2='Updated String' WHERE Column1=1";
        command.ExecuteNonQuery();
        AddText("UPDATED RECORD");
        DisplayResults(command);
        // Delete a record
        command.CommandText = "DELETE FROM MyTable1 WHERE Column1=1";
        command.ExecuteNonQuery();
    }
}
```

```

        DisplayResults(command);
    } //using
}

```

Listing 10-2 shows the execution of CREATE, INSERT, UPDATE, and DELETE commands on a SQL Azure database in a sequential operation.

■ **Note** SQL Azure requires you to have a clustered index on the table to insert entries into the table.

Similarly, Listing 10-3 shows the code for the DisplayResults function, demonstrating SELECT command execution on the database table.

Listing 10-3. *SELECT Command*

```

private static void DisplayResults(SqlCommand command)
{
    command.CommandText = "SELECT Column1, Column2, Column3 FROM MyTable1";
    using (SqlDataReader reader = command.ExecuteReader())
    {
        // Loop over the results
        while (reader.Read())
        {
            AddText(command.CommandText);
            AddText(String.Format("Column1: {0}, Column2: {1}, Column3: {2}",
                                   reader["Column1"].ToString().Trim(),
                                   reader["Column2"].ToString().Trim(),
                                   reader["Column3"].ToString().Trim()));
            AddText("\n");
        }
    }
}

```

You can find the code for Listings 10-1 through 10-3 in the ADONETConnection project in Ch10Solution located in the source code directory of this chapter. To run the ADONETConnection application, go to the bin\Debug directory of the ADONETConnection project and double-click ADONETConnection.exe. Figure 10-34 illustrates the user interface for the ADONETConnection.exe Windows application.

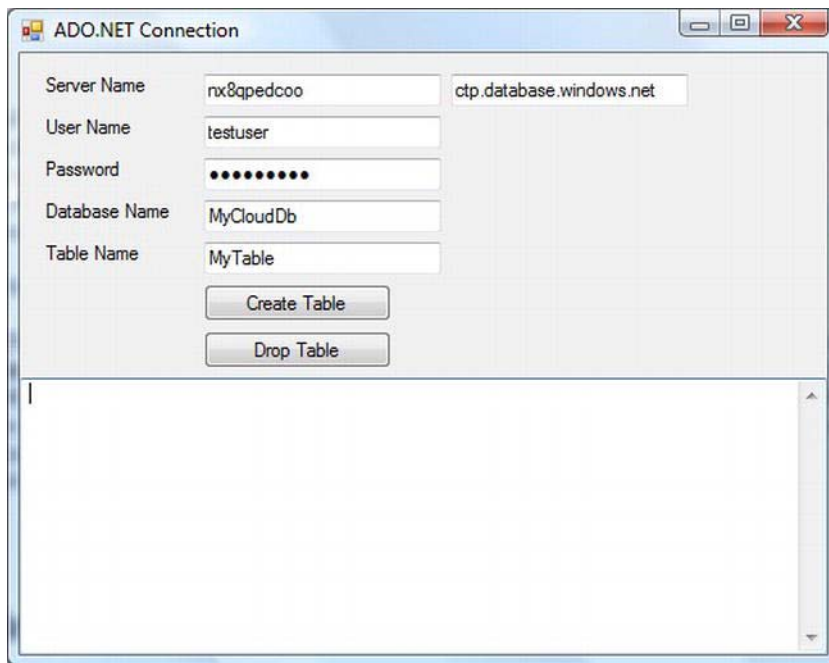


Figure 10-34. ADONETConnection Windows application

In the ADONETConnection application, you have to enter the server name, username, password, and database name of your database. Then, click the Create Table button to create a new table called MyTable1. Insert a record into that table, update the record, and delete the record. The results of the operation are displayed in the text box. To drop the table, click the Drop Table button. If the table already exists, the Create Table operation throws an exception. So, you may have to drop the table if you receive a “table already exists” exception.

■ **Caution** Here are a few important points to consider while designing SQL data access: test your queries for SQL Injection; use parameterized queries wherever possible; encrypt your database connection string; and encrypt the username, password, and database information if you’re constructing the connection string using the SqlConnectionBuilder.

Developing Windows Azure Services That Use SQL Azure

As an exercise to learn SQL Azure, in this section you develop an end-to-end application involving SQL Azure, Windows Azure, and AppFabric Service Bus. This example also helps you learn to integrate these three technologies seamlessly when building cloud applications.

Service Description

Consider a hypothetical company called SixFrogs Incorporated that offers a cloud-based demand-response service (Dem-Res) directly to utility companies and indirectly to consumers through utility companies.

A *demand-response (Dem-Res) system* is a popular pattern used by utility companies to curtail the electricity load during peak usage when the pricing for usage is very high. The cost savings are then passed on to consumers. The curtailment is determined in real time based on peak usage, pricing, and several other factors. In the interest of keeping the example conceptual, assume that the load reduction depends on peak usage and pricing.

Processes for Curtailment

The process flow for the Dem-Res system between SixFrogs, utility companies, and consumers is as follows:

1. Multiple utility companies subscribe to SixFrog's Dem-Res cloud system.
2. Consumers subscribe to the Dem-Res system through their utility company in return for a discount on their monthly electric bill.
3. Utility companies install their hardware (gateway) in consumers' houses and/or buildings and point those gateways to the Dem-Res system in the cloud.
4. Utility companies configure load curtailment for consumers for specific load devices (this example considers only HVAC).
5. Utility companies receive electric load-pricing information for a particular period of time. This pricing information is in dollars per kWh for a particular period of time. For example, a particular pricing entry may be represented as *\$16/kWh between 1:00pm and 3:00pm on Monday August 24th 2009*.
6. Utility companies pass this pricing information to the Dem-Res system.
7. The Dem-Res system reads the pricing information and sends commands to the applicable gateways in buildings and houses to automatically curtail the load.
8. Periodically, the gateways communicate the energy usage to the Dem-Res system. The Dem-Res system checks the database for peak load and pricing information and sends curtailment commands to the gateways if required.

Figure 10-35 illustrates the high-level process for the Dem-Res system.

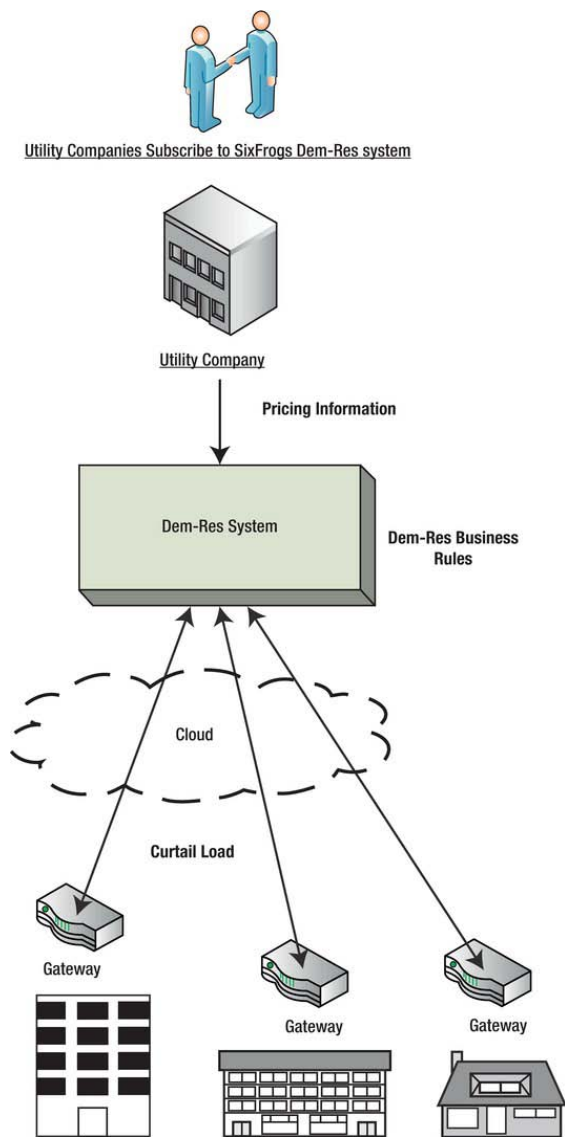


Figure 10-35. Dem-Res process

Technical Architecture

This section discusses the design of the Dem-Res system and its communication endpoints. The goal is to map the process architecture to the system architecture in the cloud representing Windows Azure components. From the earlier section and your knowledge of Windows Azure so far, it should be clear

that you can build a complete Dem-Res system in Windows Azure. Figure 10-36 illustrates the technical architecture of the Dem-Res system.

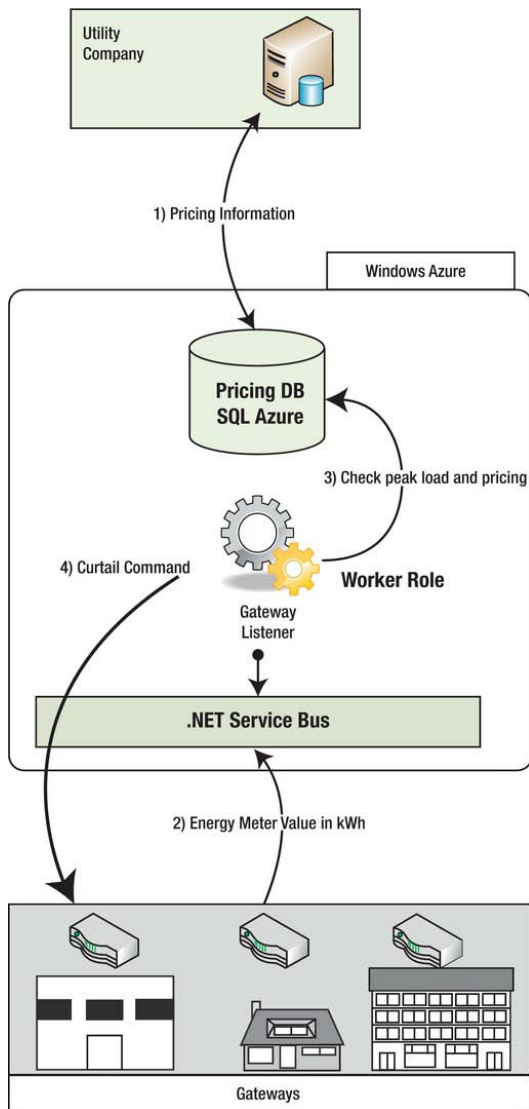


Figure 10-36. Dem-Res system architecture

As shown in Figure 10-36, the Dem-Res system consists of three core components:

- Pricing and Gateway database
- Gateway listener

- Gateway application

The flow of information within the components is as follows:

1. The utility company periodically sends gateway and pricing information to the pricing database in the Dem-Res system.
2. Periodically, the gateway sends an energy usage value to the gateway listener via the AppFabric Service Bus.
3. The gateway listener worker role queries the pricing database to check if the energy value is more than the peak load value.
4. If the gateway energy value is more than the peak load value, the gateway listener sends a curtail command to the gateway application. The gateway application in turn sends the control command to the appropriate device (such as HVAC, in this example).

Pricing and Gateway Database Design

The Pricing and Gateway database is hosted in SQL Azure and is geo-located in the same region as the gateway listener worker role to keep the communications within the same data center. As the name suggests, the database consists of pricing and gateway information.

■ **Tip** I recommend that you design your SQL Azure database in SQL Server on-premise and then migrate the database to SQL Azure. When SQL Azure fully supports SQL Server Management Studio, you can work directly with the SQL Azure database.

The Pricing and Gateway database consists of four tables, as shown in Figure 10-37.

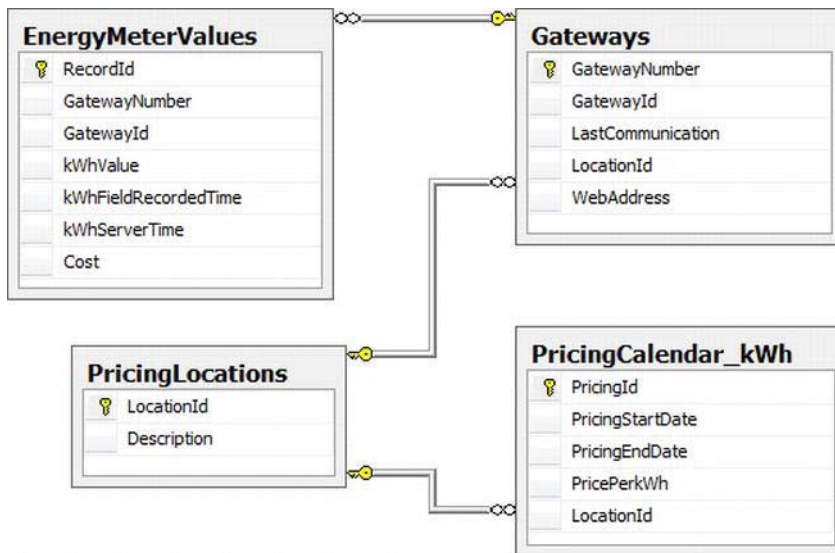


Figure 10-37. Pricing and Gateway database design

The Gateways table maintains the list of gateways in the field along with their location, which is referenced by LocationId from the PricingLocations table. The PricingLocations table consists of address locations that are mapped as pricing zones. Each zone has a price per kWh stored in the PricingCalendar_kWh table. The PricingCalendar_kWh table is updated periodically by the utility companies with the latest pricing. The EnergyMeterValues are the kWh values that gateways send periodically to the Dem-Res system.

The steps to create and testing the Dem-Res database system are as follows:

1. Create a database named proazuredemres.
2. Create table-creation scripts.
3. Create stored-procedure scripts.
4. Upload sample data into the tables.
5. Create data synchronization for the PricingCalendar_kWh table.

Creating the proazuredemres Database

To create this database, follow these steps:

1. Open SQL Server Management Studio.
2. Connect to the SQL Azure master database as an administrator, as shown in earlier sections.
3. Execute the following query in the New Query window to create the proazuredemres database:

```
CREATE DATABASE proazuredemres;
```

4. Create a new login named demresadmin in the db_owner role for the proazuredemres database. Follow the same procedure as shown in the “Creating Logins” section earlier in this chapter.
5. Log in to the proazuredemres database as the demresadmin user, and execute the following query to test if the login was created successfully:

```
select * from sys.databases;
```

Creating Database Tables

To create database tables, I recommend that you first create the tables and other database objects in your local SQL Server Express and then generate scripts to upload to the SQL Azure database. In this section, you directly create SQL Server objects in SQL Azure to keep the content relevant to SQL Azure only.

Listing 10-4 shows the script and schema to create the Dem-Res database tables.

Listing 10-4. Dem-Res Table-Creation Script

```
CREATE TABLE [dbo].[PricingLocations](
    [LocationId] [varchar](50) NOT NULL PRIMARY KEY CLUSTERED,
    [Description] [varchar](100) NOT NULL);
GO
CREATE TABLE [dbo].[PricingCalendar_kWh](
    [PricingId] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY CLUSTERED,
    [PricingStartDate] [datetime] NOT NULL,
    [PricingEndDate] [datetime] NOT NULL,
    [PricePerkWh] [float] NOT NULL,
    [LocationId] [varchar](50) NOT NULL);
GO
CREATE TABLE [dbo].[Gateways](
    [GatewayNumber] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY CLUSTERED ,
    [GatewayId] [varchar](50) NOT NULL,
    [LastCommunication] [datetime] NULL,
    [LocationId] [varchar](50) NOT NULL,
    [WebAddress] [varchar](100) NOT NULL);
GO
CREATE TABLE [dbo].[EnergyMeterValues](
    [RecordId] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY CLUSTERED,
    [GatewayNumber] [int] NOT NULL,
    [GatewayId] [varchar](50) NOT NULL,
    [kWhValue] [float] NOT NULL,
    [kWhFieldRecordedTime] [datetime] NOT NULL,
    [kWhServerTime] [datetime] NOT NULL,
    [Cost] [money] NOT NULL);
GO
ALTER TABLE [dbo].[EnergyMeterValues]
    WITH CHECK ADD CONSTRAINT [FK_EnergyMeterValues_Gateways]
    FOREIGN KEY([GatewayNumber])
```

```

REFERENCES [dbo].[Gateways] ([GatewayNumber])
GO
ALTER TABLE [dbo].[EnergyMeterValues] CHECK CONSTRAINT
[FK_EnergyMeterValues_Gateways]
GO
ALTER TABLE [dbo].[Gateways] WITH CHECK ADD CONSTRAINT
[FK_Gateways_PricingLocations] FOREIGN KEY([LocationId])
REFERENCES [dbo].[PricingLocations] ([LocationId])
GO
ALTER TABLE [dbo].[Gateways] CHECK CONSTRAINT [FK_Gateways_PricingLocations]
GO
ALTER TABLE [dbo].[PricingCalendar_kWh] WITH CHECK ADD CONSTRAINT
[FK_PricingCalendar_kWh_PricingLocations] FOREIGN KEY([LocationId])
REFERENCES [dbo].[PricingLocations] ([LocationId])
GO
ALTER TABLE [dbo].[PricingCalendar_kWh]
CHECK CONSTRAINT [FK_PricingCalendar_kWh_PricingLocations]
GO

```

The first part of Listing 10-4 defines the tables, and then second part defines foreign key relationships between the tables.

To create tables in SQL Azure, follow these steps:

1. Connect to the proazuredemres database using SQL Server Management Studio.
2. Log in as the demresadmin user.
3. Open the createtables_proazuredemresdb.sql script file window from the DbScript folder of the Chapter 10 code directory in a NewQuery.
4. Click the Execute button to create the tables in the proazuredemres database.
5. Execute the following query to check if the tables and constraints were successfully created:

```
select * from sys.objects
```

Creating Stored Procedures

One of the database design best practices is to locate data-processing logic closer to the database as much as possible. This is the reason stored procedures are recommended for data-processing logic rather than inline code. Stored procedures are also easier to modify and maintain than code because each stored procedure is an atomic unit containing data-processing logic that can be easily modified without having to recompile the code. For the Dem-Res system, I identified the stored procedures described in the following sections.

InsertPricingLocations

The InsertPricingLocations stored procedure inserts a new record in the PricingLocations table. This stored procedure is called by utility companies to add locations that are used to set energy prices. Listing 10-5 shows the create script for the InsertPricingLocations stored procedure.

Listing 10-5. InsertPricingLocations

```
CREATE PROCEDURE [dbo].[InsertPricingLocations]
    @locationId varchar(50),
    @description varchar(100)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    INSERT INTO PricingLocations(LocationId, [Description])
    VALUES (@locationId, @description);
END
```

The stored procedure consists of a simple insert statement. If you want to modify the stored procedure after it's installed in the database, replace CREATE PROCEDURE with ALTER PROCEDURE in the stored procedure body.

InsertPricingCalendar_kWh

The InsertPricingCalendar_kWh stored procedure inserts a new record in the PricingCalendar_kWh table. The stored procedure is called by the utility companies to update the kWh pricing for a particular period of time. Listing 10-6 shows the create script for the InsertPricingCalendar_kWh stored procedure.

Listing 10-6. InsertPricingCalendar_kWh

```
CREATE PROCEDURE [dbo].[InsertPricingCalendar_kWh]
    @pricingStartDate datetime,
    @pricingEndDate datetime,
    @pricePerkWh float,
    @locationId int
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO PricingCalendar_kWh
    (PricingStartDate, PricingEndDate, PricePerkWh, LocationId)
    VALUES (@pricingStartDate, @pricingEndDate, @pricePerkWh, @locationId);
END
```

InsertGateway

The InsertGateway stored procedure inserts a new record in the Gateways table. This procedure is called when a new gateway is added to the Dem-Res database by the utility company or when the gateway

communicates with the Dem-Res system for the first time. Listing 10-7 shows the create script for the InsertGateway stored procedure.

Listing 10-7. InsertGateway

```
CREATE PROCEDURE [dbo].[InsertGateway]
    @gatewayId varchar(50),
    @locationId int,
    @webAddress varchar(100)
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO Gateways(GatewayId, LocationId, WebAddress, LastCommunication)
VALUES (@gatewayId, @locationId, @webAddress, getdate());
END
```

InsertEnergyMeterValues

The InsertEnergyMeterValues stored procedure inserts a new record in the EnergyMeterValues table. This stored procedure is called when the gateway sends the energy meter value to the Dem-Res server. Listing 10-8 shows the create script for the InsertEnergyMeterValues stored procedure.

Listing 10-8. InsertEnergyMeterValues

```
CREATE PROCEDURE [dbo].[InsertEnergyMeterValues]
    @gatewayId varchar(50),
    @kWhValue float,
    @kWhFieldRecordedTime datetime,
    @kWhServerTime datetime
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @gatewayNumber int
    DECLARE @cost float
    DECLARE @locationId int
    SELECT @gatewayNumber = GatewayNumber, @locationId=LocationId
FROM Gateways WHERE GatewayId = @gatewayId;
    SELECT @cost=PricePerKWh FROM PricingCalendar_kWh WHERE
LocationId = @locationId;
    SET @cost = @cost * @kWhValue;
    INSERT INTO EnergyMeterValues(GatewayNumber, GatewayId,
kWhValue, kWhFieldRecordedTime, kWhServerTime, Cost)
VALUES (@gatewayNumber, @gatewayId, @kWhValue,
@kWhFieldRecordedTime, @kWhServerTime, @cost);
END
```

In Listing 10-8, PricePerKWh is retrieved from the PricingCalendar_kWh table to calculate the cost of energy for the kWh value at the location where the gateway is located. The cost is calculated by multiplying the kWh value by the price per kWh sent by the gateway. The cost value is then inserted into the record along with all the other fields of the table.

UpdateGatewayLastCommunication

The UpdateGatewayLastCommunication stored procedure updates the last communication time field in the Gateways table. This stored procedure is called when the gateway sends the energy meter value to the Dem-Res server. Listing 10-9 shows the create script for the UpdateGatewayLastCommunication stored procedure.

Listing 10-9. UpdateGatewayLastCommunication

```
CREATE PROCEDURE [dbo].[UpdateGatewayLastCommunication]
    @gatewayId varchar(50),
    @locationId int,
    @webAddress varchar(100)
AS
BEGIN
    SET NOCOUNT ON;
    UPDATE Gateways SET LastCommunication = getdate() WHERE GatewayId = @gatewayId
END
```

To install the stored procedures, open SQL Server Management Studio, connect to the proazuredemres database, and execute the CREATE PROCEDURE scripts as shown in Figure 10-38.

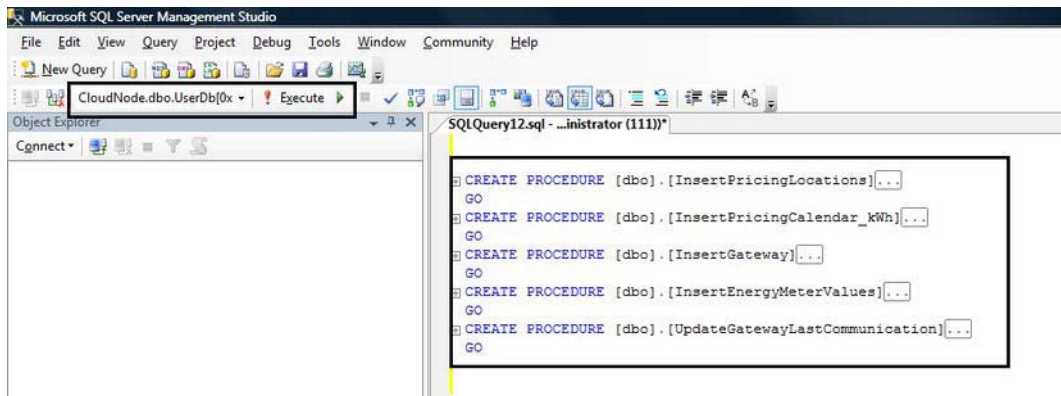


Figure 10-38. Creating stored procedures

Uploading Sample Data

In any database system design, you need sample data to test different scenarios and conditions that will affect the system in general. You also need sample data to test the business logic in application and stored procedures. In Ch10Solution in this chapter's source code directory, there is a Windows Forms project called ProAzureDemResDbApp that uploads sample data to PricingLocations, PricingCalendar_kWh, Gateways, and EnergyMeterValues in the SQL Azure Dem-Res database. The application calls the stored procedures discussed in the previous section to insert the data. The data is randomly generated based on some hard-coded parameters in the code. For example, the pricing locations are between the ZIP codes 95147 and 94583. Similarly, the gateway numbers are between 1 and 300. These values are also used to generate a web URL for the gateway, which is of the format `sb://proazure.servicebus.windows.net/gateways/{location_id}/{gateway_id}`.

Figure 10-39 illustrates the user interface of the ProAzureDemResDbApp application.

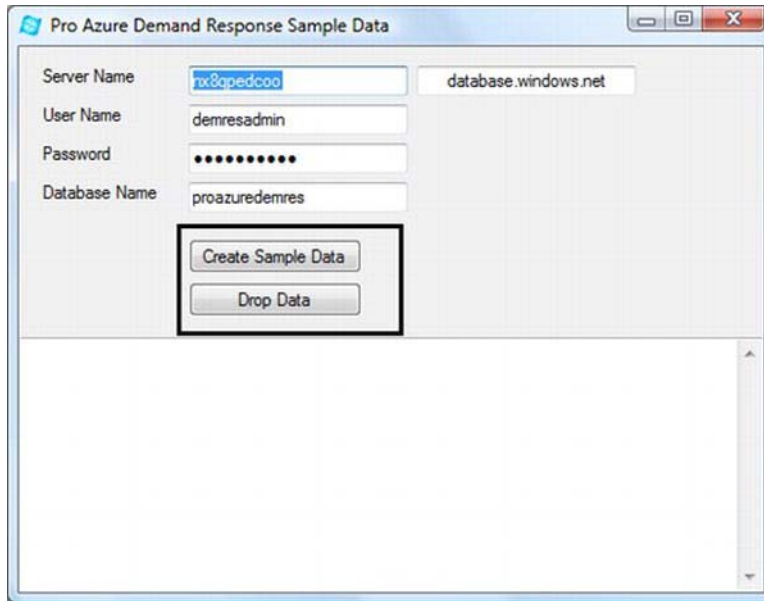


Figure 10-39. *ProAzureDemResDbApp user interface*

To upload sample data, follow these steps:

1. Run the ProAzureDemResDb application.
2. Enter your server name, username, password, and database name.
3. Click the Create Sample Data button.

■ **Note** It may take some time to run the query, depending on your network connection.

4. To delete all the data in the tables, click the Drop Data button. Dropping data is useful if you want to re-create the sample data from scratch.

Listing 10-10 shows the code to insert data in the PricingCalendar_kWh table.

Listing 10-10. *Insert Pricing Calendar Data*

```
using (SqlConnection conn = new SqlConnection(GetUserDbString()))
{
    conn.Open();
    for (int j = START_LOCATIONID; j < END_LOCATIONID; j++)
```

```

{
    using (SqlCommand command = conn.CreateCommand())
    {
        command.CommandText = "InsertPricingCalendar_kWh";
        command.CommandType = CommandType.StoredProcedure;
        string lid = j.ToString();
        Random r = new Random();
        double price = r.NextDouble();
        SqlParameter pricingStartDate = command.CreateParameter();
        pricingStartDate.ParameterName = "@pricingStartDate";
        pricingStartDate.Value = PRICINGCALENDAR_STARTDATE;
        command.Parameters.Add(pricingStartDate);
        SqlParameter pricingEndDate = command.CreateParameter();
        pricingEndDate.ParameterName = "@pricingEndDate";
        pricingEndDate.Value = PRICINGCALENDAR_ENDDATE;
        command.Parameters.Add(pricingEndDate);
        SqlParameter pricePerkWh = command.CreateParameter();
        pricePerkWh.ParameterName = "@pricePerkWh";
        pricePerkWh.Value = price;
        command.Parameters.Add(pricePerkWh);
        SqlParameter locationId = command.CreateParameter();
        locationId.ParameterName = "@locationId";
        locationId.Value = lid;
        command.Parameters.Add(locationId);
        command.ExecuteNonQuery();
    } //using
} //for
} //using

```

Listing 10-10 demonstrates calling the `InsertPricingCalendar_kWh` stored procedure with parameterized values. In database programming, parameterized values are recommended over plain-text query strings, because there is a SQL injection risk when you use plain text queries. Parameterized queries reduce this risk because they don't append the value of the parameter to the SQL query, which gives a clear separation between the SQL query and its parameters. In Listing 10-10, note that the price per kWh is generated randomly. In the real world, the utility company provides the Dem-Res application with the price per kWh.

To test the creation of the data, you can login to `proazuredemres` database in your account using SQL Server Management Studio and execute a `select *` query on all the database tables.

Optimizing SELECT Queries

In the Dem-Res system, the most commonly used SELECT query selects `PricePerkWh` by `LocationId` from the `PricingCalendar_kWh` table, because for every message that comes in from the gateway, you have to calculate the cost. So, it's important that the performance of the SELECT query is optimized by appropriate indexes on the table. Depending on the other queries in the system, you may choose to create a clustered index on the `LocationId` field. But you can have only one clustered index on a table, which in this case is `PricingId`. In this example, you create a simple index on the `LocationId` field by executing this query:

```
CREATE INDEX INDEX_PricingCalendar_kWh_LocationId
```

```
ON PricingCalendar_kWh(LocationId);
```

To test the index scan, you need sufficient data in the PricingCalendar_kWh table; otherwise, the SQL Server optimizer scans only the clustered index because the SQL Server optimizer may choose a different execution plan that yields better results. You can generate more test data by executing the stored procedure shown in Listing 10-11.

Listing 10-11. *AddSampleData Stored Procedure*

```
CREATE PROCEDURE AddSampleData
@NumRows int
AS
DECLARE @counter int
DECLARE @locationId int
DECLARE @locationIdStr varchar(50)
DECLARE @desc varchar(50)
DECLARE @pricingStartDate datetime
DECLARE @pricingEndDate datetime
DECLARE @pricekWh float
DECLARE @gatewayUrl varchar(100)
DECLARE @gatewayId varchar(50)
DECLARE @kWhValue float
DECLARE @now datetime

SELECT @counter = 1
WHILE (@counter < @NumRows)
BEGIN

SET @locationId = 10000 + @counter;
SET @locationIdStr = CAST(@locationId as varchar);
SET @desc = @locationIdStr + '-' + CAST(@counter as nvarchar)+'-description';
SET @pricingStartDate = DATEADD(m, 2, getdate());
SET @pricingEndDate = DATEADD(m, 3, getdate());
SET @pricekWh = CAST(@counter as float) * 0.00052;
SET @gatewayId = 'MyGateway' + @locationIdStr;
SET @gatewayUrl = 'sb://proazure.servicebus.windows.net/gateways/' +
@locationIdStr + '/' + @gatewayId;
SET @kWhValue = @pricekWh * 5.2;
SET @now = getdate();

EXEC InsertPricingLocations @locationId, @desc;
EXEC InsertPricingCalendar_kWh @pricingStartDate, @pricingEndDate,
@pricekWh, @locationId;
EXEC InsertGateway @gatewayId, @locationId, @gatewayUrl;
EXEC InsertEnergyMeterValues @gatewayId, @kWhValue, @now, @now;

SELECT @counter = @counter + 1;

END
```

The AddSampleData stored procedure creates sample data in all the database tables similar to the ProAzureDemResDbApp Windows application you saw earlier. Execute the stored procedure with the following query to enter 10,000 entries in the database tables:

```
EXEC AddSampleData 10001;
```

■ **Note** I demonstrate two different ways of creating sample data so you can choose the approach you feel comfortable with and understand the advantage of having the data-processing logic closer to the data. You can easily modify the AddSampleData stored procedure without recompiling any code as you would have to do with the Windows application shown earlier.

Next, to view the query execution plan, execute the query shown in Listing 10-12.

Listing 10-12. Show Query Plan

```
SET SHOWPLAN_ALL ON
GO
SELECT PricePerKWh FROM PricingCalendar_kWh WHERE LocationId = 95148;
GO
SET SHOWPLAN_ALL OFF
```

SET SHOWPLAN_ALL ON enables you to see the output of the query execution plan, as shown in Figure 10-40.

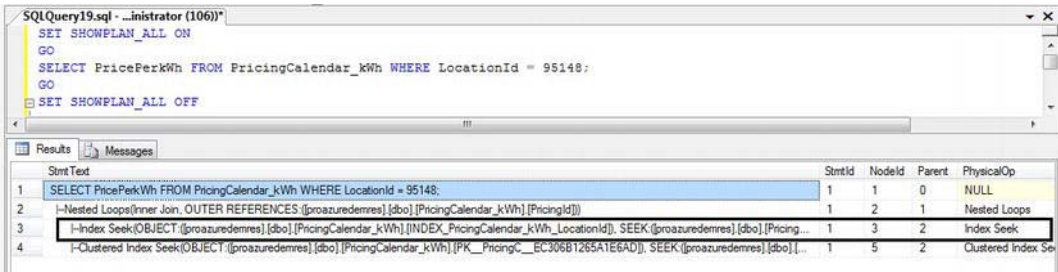


Figure 10-40. Query plan output

The query plan shows the steps followed by the SQL Server query optimizer to execute your query. The information shown in the plan is valuable for optimizing queries or debugging slow-running queries. Figure 10-41 shows the query optimizer using the index scan of the PricingCalendar_kWh_LocationId index you created earlier. To see the execution of the plan graphically, go to Query ► Display Estimated Execution Plan in SQL Server Management Studio.

Query 1: Query cost (relative to the batch): 0%
 SET SHOWPLAN_ALL ON

T-SQL
 SET ON/OFF
 Cost: 0 %

Query 2: Query cost (relative to the batch): 100%
 SELECT PricePerkWh FROM PricingCalendar_kWh WHERE LocationId = 95148;

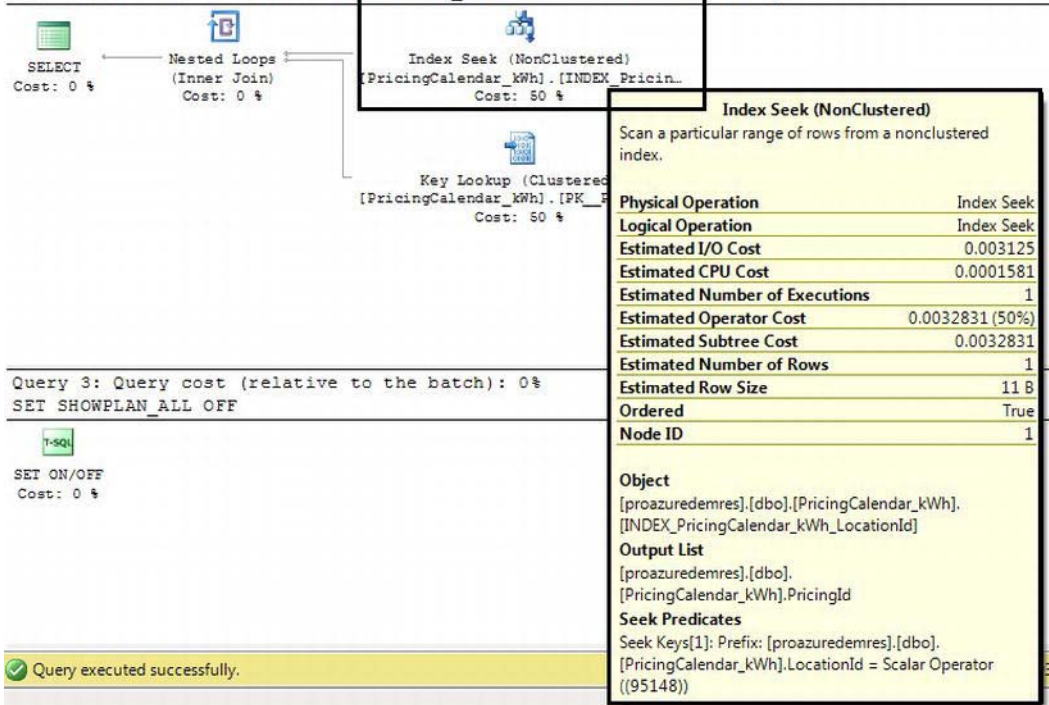


Figure 10-41. Graphical query plan output

Pricing Table Synchronization

The values in the PricingCalendar_kWh table are provided by the utility companies. There are several ways to synchronize data in the cloud PricingCalendar_kWh table with an on-premise database table, such as creating a custom web service that can be called by utility companies, having an FTP server for data transfer between utility companies and the Dem-Res system, SQL Server Integration Services (SSIS), SQL Azure Data Sync, and so on.

SQL Server Integration Services

SSIS is an Extract-Transform-Load (ETL) tool that comes with higher SQL Server editions. You can use SSIS to do the following

- Extract data from a structured or unstructured data source.
- Clean up the data or apply business rules to the data.
- Upload the clean data to the destination database tables.

SSIS is popular in business intelligence (BI) applications for extracting data from different kinds of sources and uploading the aggregated and clean data to a data warehouse for analysis and reporting. But the application of SSIS isn't limited to BI applications: many organizations use SSIS for simple cross-database data transfer. Figure 10-42 illustrates the use of SSIS in different kinds of applications.

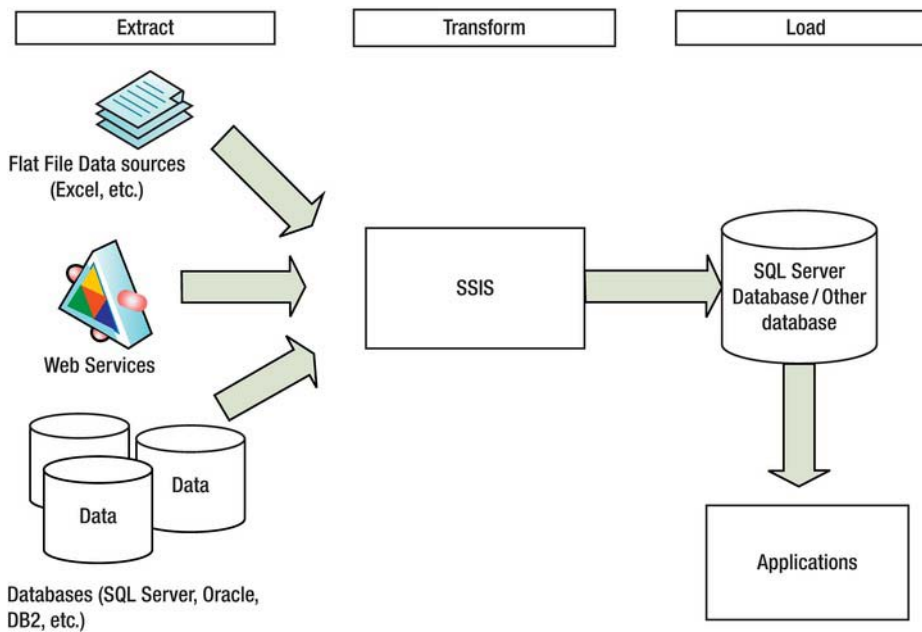


Figure 10-42. SSIS applications

■ **Note** Since SSIS is outside the scope of this book, we've created an online companion with a section that shows you how to use SSIS to synchronize data between an on-premise database and the SQL Azure cloud database. Extra content can be downloaded with the source code at <http://azureplatformbook.codeplex.com>.

Gateway Application Design

A *gateway application* runs on gateways. This example builds it as a simple Windows application, but in the real world such applications are background processes that run on embedded operating systems like Windows CE. The gateway application calls the DemRes service in the cloud via the AppFabric Service Bus using the `netTcpRelayBinding`. In `Ch10Solution`, the `DemResGateway` project represents the gateway application. `DemResGateway` is a Windows application that implements the callback interface that the DemRes service can call to curtail the load. Figure 10-43 illustrates the `DemResGateway` architecture.

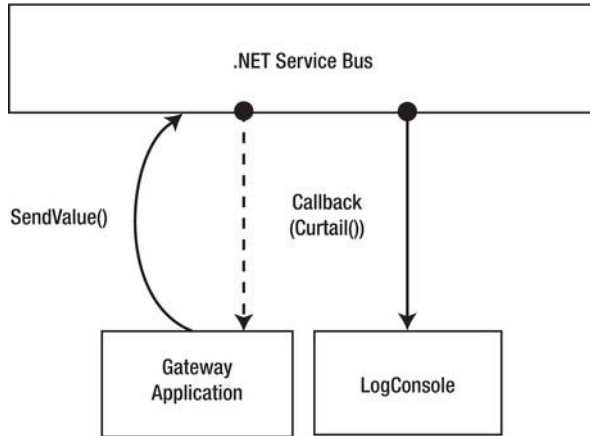


Figure 10-43. *DemResGateway application architecture*

Running the ProAzure Demand-Response Service

The steps required to run the Dem-Res system are outlined next. Begin by performing the following prerequisites:

1. Register for a Windows Azure account.
2. Create a project in the AppFabric portal.
3. Create a username and password in the AppFabric portal.
4. Create a new SQL Azure database called `proazuredemres`.
5. Log in to `proazuredemres` using SQL Server Management Studio.
6. Open the `proazuredemres_allobjects.sql` file in a New Query window, and execute the script to create objects in the `proazuredemres` database.

■ **Note** Make sure you've chosen the correct database before executing the script. You may also use SQLCMD to execute the script from the command line.

7. Create sample data by executing the following stored procedure:

```
EXEC AddSampleData 10001;
```

Now, follow these steps to run the application:

8. Open Ch10Solution in Visual Studio.
9. Add your own usernames, passwords, endpoints, SQL Azure connection strings, and other configuration parameters in the configuration files of the DemResWorker, DemResGateway, and LogReceiverConsole projects.
10. Build and deploy the DemResWorker cloud service project to Windows Azure.

■ **Tip** Run the complete end-to-end application locally in the development fabric before deploying it to Windows Azure.

11. Start the LogReceiverConsole application. This application receives log messages from the DemResWorkerRole.
12. Start the DemResWorker service in Windows Azure or in the development fabric.
13. Start the DemResGateway application on your local machine.
14. Click the Send kWh Value button to send a single kWh value to the DemResWorker service.
15. Click the button several times to get a value that results in a price greater than one dollar. If the cost of energy per unit is more than one dollar, the DemResWorker service sends a curtail command to the gateway using a callback interface. When this happens, you see a message box with the curtail value, as shown in Figure 10-44.

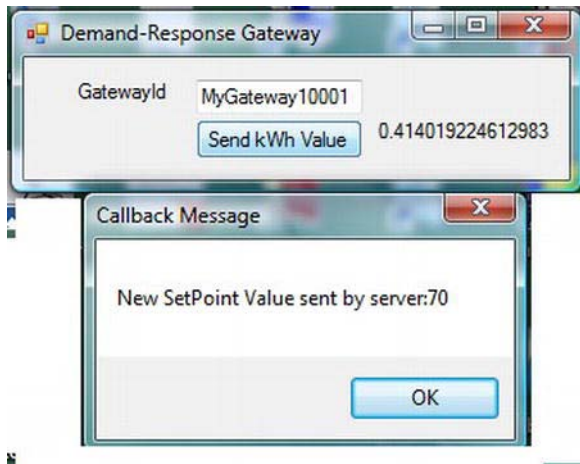


Figure 10-44. DemRes callback

Database-Migration Strategies

When you're creating a database from scratch, designing and deploying it in SQL Azure shouldn't be difficult. All the limitations and constraints in SQL Azure are published, and the database is a subset of your on-premise SQL Server database. So, any database you design for SQL Azure can be easily migrated to an on-premise SQL Server. But SQL Server is a mature database server used in enterprises of all sizes. Migrating these legacy databases to the cloud may require a complete redesign because of the wide range of rich features like Server Broker, CLR stored procedures, replication, mirroring, and so on that are supported by on-premise databases but aren't yet supported in SQL Azure.

A database migration involves migrating not only the data and its schema but also the business logic and applications that depend on that database. Thus the database-migration strategy to SQL Azure involves the following four actions:

1. Data definition migration
2. Data migration
3. Business logic migration
4. Application migration

Data Definition Migration

The *data definition* refers to the design of your database schema, which may include storage-specific objects like tables, views, indexes, constraints, and so on. The data definition is tightly coupled to the type of data stored in the database to achieve optimal performance.

A particular database's data definition can be easily represented by a script that can be automatically generated in SQL Server Management Studio. With minor modifications, these scripts can be executed on SQL Azure to migrate the data definition from on-premise SQL Server to SQL Azure. So, other than execution tools like SQL Server Management Studio and SQLCMD, you don't need any specific tools to migrate data definition from on-premise SQL Server to SQL Azure when you have the

script representing the data definition. This is the recommended approach to migrate data definitions from on-premise to SQL Azure, because this approach gives you more control over the definition of the data.

You can also use SSIS to replicate data definitions on the fly between an on-premise database and SQL Azure. But this approach may require more work in designing, building, testing, and deploying packages.

Typical steps required to migrate data definition from on-premise SQL Server to SQL Azure are as follows:

1. Log in to your on-premise SQL Server database.
2. Generate a script for all the data definition objects, which include tables, views, indexes, and constraints (see Figure 10-45).

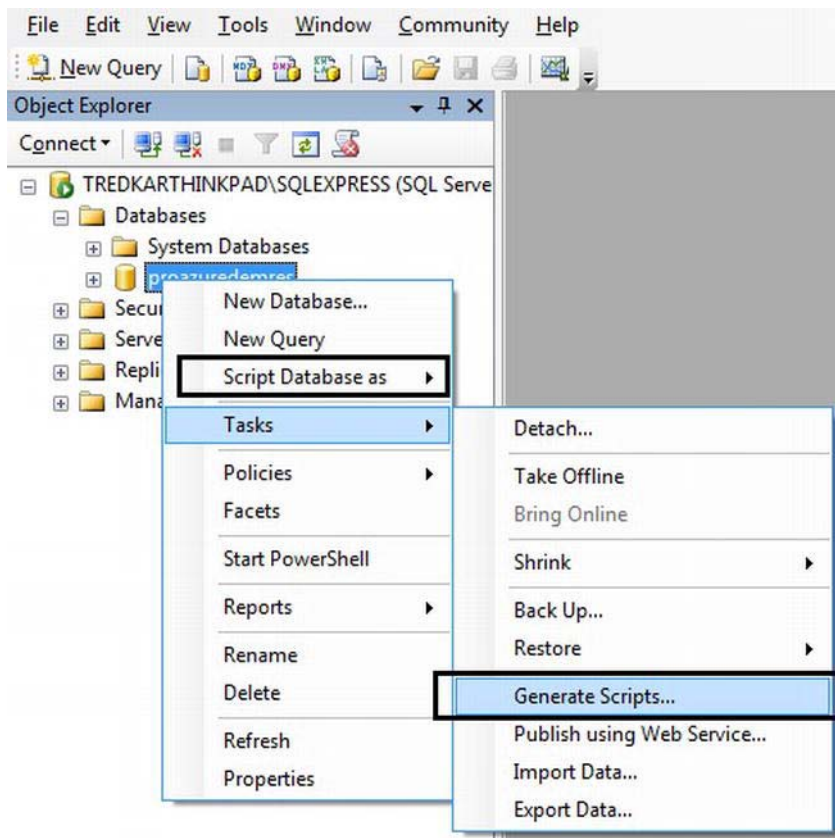


Figure 10-45. Generate a script

3. Modify the script to remove the features or commands not supported by SQL Azure.

■ **Note** For commands not supported in SQL Azure, please refer to the SQL Azure documentation. For a full list of unsupported commands, go to <http://msdn.microsoft.com/en-us/library/windowsazure/ee336253.aspx>.

4. Save the script as a SQL Azure database definition script.
5. Connect to the SQL Azure database to which you want to migrate the data definition.
6. Open a New Query window, and copy and paste the content of the SQL Azure database script into it.
7. Execute the script to install the data definition objects in the SQL Azure database.

Data Migration

Data migration refers to the actual data stored in SQL Server. An on-premise SQL Server supports several tools to migrate data across different SQL Server instances as well as heterogeneous databases like Oracle, DB2, Access, and so on. Some of the popular data-migration tools are as follows:

- *SQL Server BCP Utility*: Used for bulk copying data between SQL Server instances and/or file systems.
- *SQL Server Management Studio*: Used to back up and restore SQL Server databases.
- *Database mirroring*: Supports real-time data mirroring across SQL Server databases.
- *Log shipping*: Used for real-time backup and restore functionality.
- *Replication*: Supports real-time data mirroring across SQL Server databases.
- *SQL Server Integration Services (SSIS)*: Includes built-in backup and restore tasks that can be included in packages and executed in a standalone manner or coupled with other business logic.

■ **Note** This isn't an exhaustive list but just the most commonly used tools that are included with SQL Server. Most companies use advanced data-replication and -migration tools built by Microsoft's partner companies. I discuss only the most popular out-of-the-box SQL Server tools that you can use for data migration.

Most of the tools from this list need both the source database and the destination database supporting the tool. As of this writing, other than SSIS and the BCP Utility (supported in future releases of SQL Azure), SQL Azure doesn't support any of these tools. Even within SSIS, some of the maintenance tasks aren't supported by SQL Azure, so your best option is to use tasks that support ADO.NET

connections. The BCP tool is the simplest to use and the best option for quickly scripting and/or scheduling the data migration on a periodic basis. On the other hand, SSIS gives you the most flexibility because you can design workflows and/or data transformations within your data-migration package. The BCP Utility is the best option for simple, quick, no-code data migrations, whereas SSIS is the best option for data migrations involving workflows and/or transformations.

Business Logic Migration

In the simplest terms, the *business logic* refers to the logic that is applied to the data before it's stored in the database or retrieved from the database for viewing. The business logic may also consist of business rules applied to inbound as well as outbound data in specific conditions. In some distributed systems, the business logic is embedded in the middle tier; in other cases, it's embedded in stored procedures closer to the database. There are also some client/server systems where the business logic is embedded in the client tier. Microsoft Excel is a very good client example in which you can connect to a database and add business logic to the data retrieved in Excel.

When you're planning a database migration, migrating the business logic associated with the data is equally important. In cases where the business logic is programmed in stored procedures, you can follow the same procedure as in the data-definition migration discussed earlier. You can generate a script defining the stored procedures and execute the script in SQL Azure. SQL Azure doesn't support CLR stored procedures yet, so you have to reprogram the stored procedures in .NET middle-tier components and TSQL stored procedures.

When the business logic is embedded in the middle tier, you must identify the SQL Server-specific features used in the business logic and verify their supportability in SQL Azure. If they aren't supported, then you have to redesign an alternative.

Your business logic migration strategy will change depending on the tier that owns the business logic. Typically, in large-scale enterprise systems, the business logic is programmed in the middle tier, so multiple applications can share the same data and have their own business logic components. In these cases, migration may require a detailed analysis and planning exercise. In small- to medium-scale databases, the business logic tier is typically programmed in stored procedures and closer to the data. In these cases, if the business logic is in TSQL stored procedures, the process is easier—assuming the stored procedures access objects supported by SQL Azure. If the business logic is in CLR stored procedures, you need a detailed planning and analysis exercise similar to that used with middle-tier components.

Application Migration

All databases provide data-retrieval and -modification services to one or more applications that process inbound and outbound data from the database. Without applications, databases are simply silos of isolated data providing no value to the business. You don't need a database to create a silo of data; you can store the data in a file system, on tape, or on a storage area network in its raw format. Enterprises store data in databases to make it available to applications. Applications then retrieve data from the databases and present it to end users in a readable and business-friendly format.

When you're designing a strategy for a database migration to SQL Azure, you have to consider all the applications that are actively using the database and supporting business functions. In your migration strategy, you must design a business continuity plan in which the database is migrated to SQL Azure without affecting the business continuity of the applications and the database itself. In some cases, you may also have to migrate the applications to Windows Azure along with the database to SQL Azure. Business continuity is critical to enterprises, and all migration strategies must be designed so that application downtime is zero.

Database Growth-Management Strategies

When your data is on-premise, you can manage your SQL Server database's growth by adding more storage capacity. Typically, an on-premise storage area network is shared across multiple databases and applications, and it's only a matter of acquiring an extra block of storage from the company's storage-management team. Even though a cost is associated with the storage, you still have control over how you distribute your database growth.

When your data is in SQL Azure, there is a storage constraint of 10GB per database, and you don't have control over how the data files are stored or distributed across the storage area network. Microsoft's argument behind this constraint is that according to the company's analysis, 90% of the SQL Server databases in the world are less than 9GB in size.

With this constraint in mind, how do you architect your database for growth beyond 50GB? The following are a few strategies I have designed for SQL Azure customers:

- Partition data by location, and distribute it across multiple SQL Azure data centers.
- Partition data by date into multiple databases.
- Partition data by business functions into bucket databases.
- Partition data by tenant, with one configuration and one content database per tenant.
- Partition data between on-premise and SQL Azure databases.

■ **Note** Because of the SQL Azure size restrictions, all these strategies revolve around creating multiple SQL Server databases in SQL Azure and partitioning data across these databases.

In all the partitioning options, typically a centrally located or replicated configuration database maintains the references and boundary parameters of the content databases. The content databases contain the actual content partitioned by the appropriate boundary condition. These boundary conditions may be one of more of the following: location, date, business function, tenant, and premise. Figure 10-46 illustrates some of these partitioning strategies.

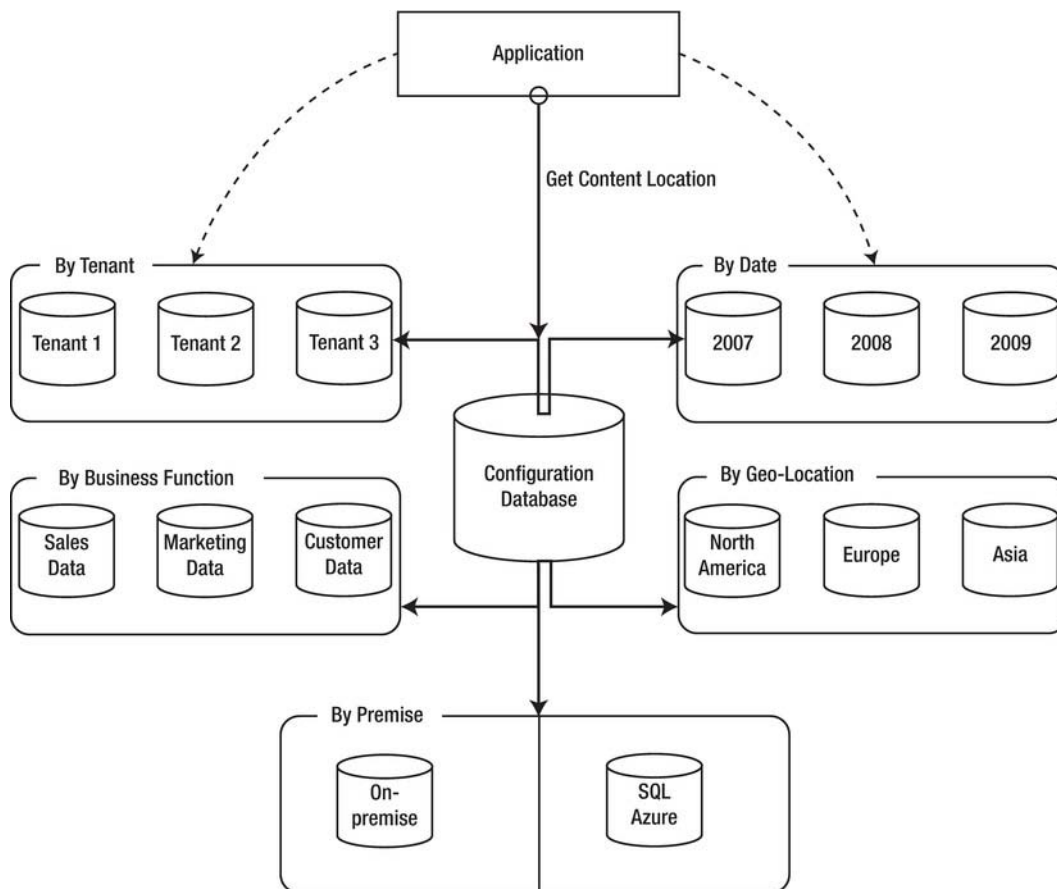


Figure 10-46. Partitioning strategies

In Figure 10-46, the configuration database contains the partition information of the content databases. The application queries the configuration database with query parameters and retrieves the list of content databases that fall within the specified parameters. For example, if your query is for dates in 2007, then the configuration database sends references to the 2007 database. The application can then connect to the appropriate database to execute the query.

Because of the two hops between the application and the databases, there is a performance impact on the data retrieval from the content databases. The configuration database isn't expected to change frequently because it depends on the partitioning parameters, which don't change often. Therefore, you can cache the configuration data in the application and synchronize only when it changes, bypassing an additional hop to the configuration database.

SQL Azure Reporting

Most applications need to provide some reporting functionality. Many on-premise applications have used SQL Server Reporting Services (SSRS) as their reporting platform. This server-based platform provides reporting functionality and tools that make it possible to create reports in many formats. It also serves as a delivery engine, delivering reports over standard http protocols.

In addition, SSRS offers the built-in ability to export your reports in other formats, such as Excel, Word, PDF, CSV, and others.

In an on-premise scenario, in order to get SSRS up and running, you would need to provision and configure one or more application-tier servers for report hosting and delivery, as well as another database (and perhaps server) to host report server data. For many non-enterprise customers, the additional cost and maintenance overhead has been a blocker to adoption. SQL Azure Reporting solves this problem by providing SSRS functionality as a cloud service.

■ **Note** At the time of writing, SQL Azure Reporting was released in Community Technology Preview (CTP) mode. It is not recommended for production applications yet.

The benefits of SQL Azure Reporting are obvious. Freeing you from the provisioning and maintenance of additional servers enables you to focus on building your reports. Additionally, your reporting services engine will be capable of scaling with your application without needing to purchase and maintain additional equipment. Finally, since Reporting Services is provided as a cloud service, you can consume your reports from either on-premise or cloud applications.

Sample Report

As part of our project we're going to assume that one of the requirements is to provide a report of the most expensive locations. We want to be able to deliver this information in a variety of formats, and we need it up quickly, so we've chosen to use SQL Azure Reporting.

Creating Reports

In-depth details around creating reports is outside the scope of this book, so I will be focusing on the differences between building for SQL Server Reporting Services and SQL Azure. For more information on building reports, go to: <http://msdn.microsoft.com/en-us/library/bb522683.aspx>

The steps required to implement SQL Azure Reporting are:

1. Ensure you have installed "SQL Server Business Intelligence Studio", as part of your SQL Server client tools installation. This is a Visual Studio Shell environment that provides a graphical IDE for creating reports.
2. Provision your report server on Azure. Navigate to the Azure Management Portal, click 'Reporting' in the left-hand navigation, and follow the instructions to request CTP access.
3. Create the SQL Azure database that will serve as the source for report data.
4. Author the report using the Visual Studio Business Intelligence IDE.

5. Deploy the resulting .RDL file to the report server.

Once you have completed steps 1, 2, and 3, you can begin creating your report. Open SQL Server Business Intelligence Studio and create a new Report Server project. Choosing Add New Report will bring up a wizard that will walk you through the next steps: defining your data source, and your dataset.

Define Data Source

The first step in creating the report is to define your data source. Retrieve your connection string from the Azure portal, and paste it into the Connection String field. Click the Credentials button. Here you have the choice of entering credentials, prompting for credential, or requiring no credentials. Also, choose whether you want this to be a shared data source for all reports to access. Typically you would want to do this, and the data source will be deployed to the report server. ***The final part is the most important: change the type to 'Microsoft SQL Azure' in the 'Type' drop-down.*** Otherwise you will not be able to deploy the data source to the report server—it will return a type mismatch error.

Define Query/Dataset

The next step is to define the query that will populate the dataset for the report. For our report, we are going to use the following query:

```
SELECT TOP (@NumRecords) pl.Description, k.PricePerkWh
FROM      dbo.PricingCalendar_kWh AS k INNER JOIN
           dbo.PricingLocations AS pl ON k.LocationId = pl.LocationId
ORDER BY k.PricePerkWh DESC
```

@NumRecords is an input parameter that defines the number of records that you want returned. This query will return the X highest priced locations. Paste this query into the text area of the query designer, and click next to choose options for the report layout. There are only two fields, so it's not complex. Use tabular layout. Preview the report by clicking the Preview tab. If the report looks the way you want, then we're ready to deploy.

Deploy reports

Navigate to the Reporting section of the Azure portal, and copy the URL of your SQL Azure Report Server. Right-click the project and select Properties. In the project properties dialog, paste your copied URL into the TargetServerURL property. For example:

```
https://<instance>.ctp.reporting.database.windows.net/ReportServer/
```

Deploy the project by right-clicking the project and choosing Deploy. You will be prompted for report server credentials during the deployment. Use the report server admin credentials that you set up in the Azure Management portal.

View Reports

Open your browser and navigate to your report server URL (i.e., <https://<instance>.ctp.reporting.database.windows.net/ReportServer/>). Sign in to the portal using the user id and password you obtained in the Azure Management portal. Navigate to your report and click on the name to view it (Figure 10-47).

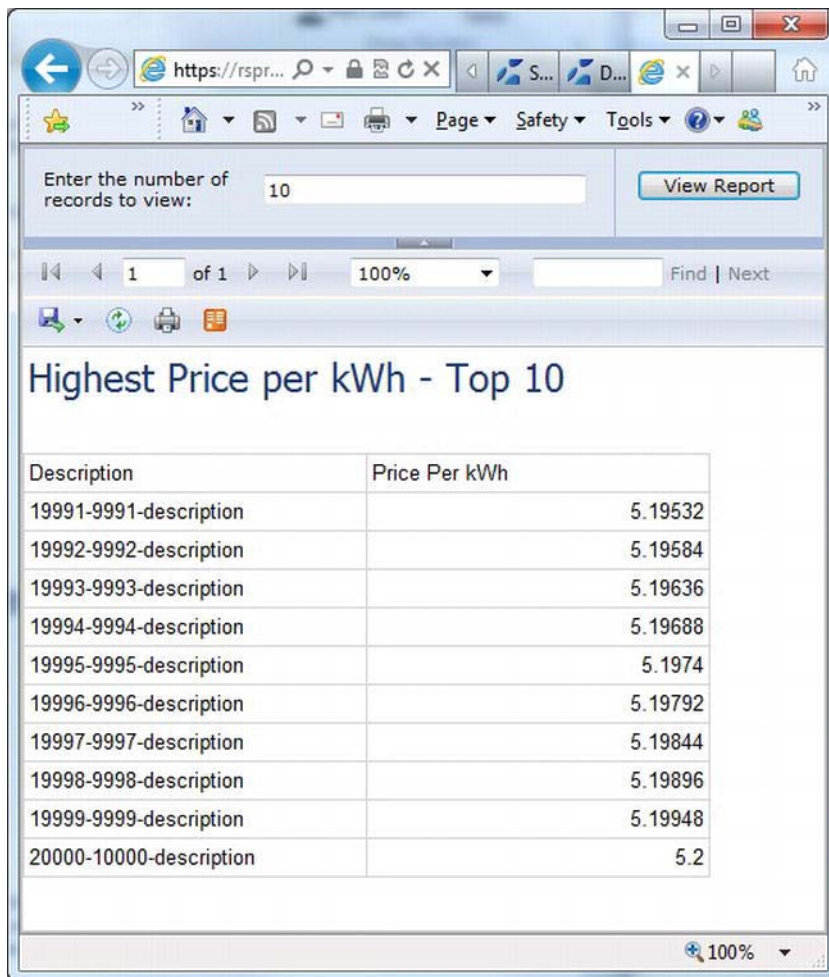


Figure 10-47. Report delivered via SQL Azure Reporting

■ **Note** For additional information and more samples, see SQL Azure Reporting samples (<http://go.microsoft.com/fwlink/?LinkId=207630>).

SSRS Feature Fidelity

Not all features of SSRS are baked into SQL Azure Reporting yet. Table 10-1 provides a non-inclusive list of some of the key feature differences. For the full list, go to <http://msdn.microsoft.com/en-us/library/gg430132.aspx>.

Table 10-1. SSRS Feature Fidelity

Feature	SQL Azure Reporting	SQL Server Reporting Services
Data Sources	SQL Azure only	Built-in or customizable, including SQL Azure
Report Management	SQL Azure Development Portal	Report Manager or SharePoint
Extensibility	No extensions	Custom extensions for processing, rendering, delivery, and security
Security	Username/password	Username/password, Windows authentication, other authentications
Subscription and scheduling	Not available	Available
CLR/Custom assemblies or code	Not supported, no code in a Code element	Supported
SharePoint Integrated Mode	Not supported	Supported

Data Sync

SQL Azure Data Sync provides the ability to synchronize SQL Azure databases with other SQL Azure databases, or even on-premise databases. Some of the possible scenarios include:

- *Geographically-dispersed applications:* For performance or other reasons, you may choose to deploy your application in multiple data centers. You will want to keep the application server and the database server together to avoid additional network bandwidth costs.
- *Backup:* Some application owners are extra careful, or perhaps they have legal compliance issues requiring them to have a copy of data on-premise. They want to know that if the worst-case scenario ever happens, as a last resort they will have their data synchronized to multiple environments, or even on-premise.
- *Scale-out within a datacenter:* Do this to support elastic spikes in demand.

■ **Note** SQL Azure Sync is currently released in CTP2. It was announced that Microsoft is planning to release a CTP3 with new and different features (<http://blogs.msdn.com/b/sync/archive/2011/03/08/sql-azure-data-sync-update.aspx>). Rather than go deep into examples for an already-deprecated version, I will be focusing on Data Sync at a high level. For more information on SQL Azure Data Sync, go to <http://blogs.msdn.com/b/sync/>.

Data Sync Design

The design starts with the hub. One database must serve as the hub for all transactions. Depending on your scenario, the hub might be a central database you use for transactional processing. Or, you might choose to have the hub serve as the central point for serving data to other processing or reporting servers.

Synchronization Options

All synchronization happens between the hub and a database. When you set up your synchronizations, you will set up a separate sync for each database. You have the following three synchronization options:

- *Sync to the Hub*: One-way sync from the edge database to the hub.
- *Sync from the Hub*: One-way sync from the hub to the edge database.
- *Bi-Directional*: Two-way sync of data, after the sync the tables being synchronized should be identical

These sync options provide the functionality to cover most scenarios for synchronization. Currently the smallest synchronization interval is 5 minutes. This means that a true real-time sync is not possible yet. You will need to take this into account in your planning processes.

Summary

In this chapter, you learned to work with SQL Azure, the SQL Server database in the cloud. Because SQL Azure is based on SQL Server, you can use your existing SQL Server knowledge to work with SQL Server.

SQL Azure is a subset of SQL Server and supports only limited features specifically geared toward storage and retrieval of relational data. Many administration- and application-level features like mirroring, replication, the BCP Utility, Service Broker, CLR, replication, and so on aren't available in SQL Azure.

The biggest benefit of SQL Azure is its accessibility to any application from anywhere as long as the platform supports the TDS protocol. You can write cloud and on-premise applications to seamlessly query data in SQL Azure. In the Demand Response example, you saw the flexibility of accessing SQL Azure from Windows Azure, as well as seamless integration between SQL Azure, Windows Azure, and AppFabric.

Also, I introduced some database-migration and database growth-management strategies for SQL Azure databases. Finally, I discussed new technologies not yet released, SQL Azure Reporting Services and SQL Azure Data Sync. These will be key elements of the SQL Azure platform going forward.

Bibliography

Microsoft Corporation. (n.d.). *SQL Azure Team Blog*. Retrieved from Windows Azure Team Blog:
<http://blogs.msdn.com/b/windowsazure/>.

Robinson, D. (2009). The Relational Database of the Azure Services Platform. *MSDN*, 71-74.

Microsoft Corporation. (n.d.). *Sync Framework and SQL Azure Data Sync Team Blog*.
<http://blogs.msdn.com/sync/>.

Index

■ Numbers and Symbols

- (dash) character, 210
- # (number sign) character, 252
- \$filter option, 254, 278–279
- \$MetricsCapacityBlob, 297
- \$MetricsTransactionsBlob, 297
- \$MetricsTransactionsQueue, 298
- \$MetricsTransactionsTable, 297
- \$top option, 254, 274, 279–280
- / (forward slash) character, 252
- ? (question mark) character, 252

■ A

- Abandon() method, 468
- Access control, 32
- Access Control List (ACL), 140
- Access Control Service. *See* ACS
- account name, 207, 209–211, 269–271, 278–279, 287
- account operations, 218–222, 269
- ACL (Access Control List), 140
- ACS (Access Control Service), 327–380
 - claims, 328–333
 - digital identities, 327–328
 - integration
 - relay authentication with, 389–392
 - securing requests with, 469
 - Management Portal for, 342–365
 - ADFS 2.0, 345–348
 - Application Integration section, 362
 - Certificates and Keys section, 359
 - digital identities, 345, 350–355
 - Endpoint References section, 365
 - Login Pages section, 362–365
 - Management Service section, 362
 - Portal Administrators area, 361
 - provisioning ACS service namespace, 342–345
 - Relying Party applications, 355–357
 - Rule Groups section, 357–358
 - SDKs and Documentation section, 365
 - Service Identities section, 361
 - tokens, 359–360
- programming applications for, 366–377
 - claims-based authorization, 374–377
 - configuring ACS using Management Portal, 367
 - designing ACS rules, 373
 - identity providers, 367
 - Passive Federation case using ACS, 367
 - relying party, 367–369, 371–373
 - rule groups and rules for mapping claims, 368
 - WS-Federation provider, 371–372
- usage scenarios for, 333–342
 - cross-enterprise application, 337–339
 - enterprise cloud application, 334–336
 - ISV cloud service, 339–341
 - retrieving tokens from ACS, 341–342

- ACS-hosted login page, 362–363, 375
- ACS-hosted login screen, 375
- Activate Windows Azure Connect checkbox, 322
- Activated Endpoints folder, and Groups and Roles folder, 319
- Active Directory Federation Services (ADFS), 345–348, 379
- Add Certificate button, 124
- Add Roles dialog, 313
- Add STS Reference, 370
- AddMessage() method, 217–218, 232
- AddProAzureReader() method, 289–290
- AddRecord method, 269
- ADFS (Active Directory Federation Services), 345–348, 379
- Admin.aspx page, 375
- administration features, of SQL Azure service, 503
- ADO.NET Data Services, 249, 256, 259, 265–266, 274, 279, 287, 289, 306

- ADO.NET framework, connecting to SQL Azure service database with, 527–530
- affinity, geographic, 121–123
- Affinity group, 121, 123
- AllowPartiallyTrustedCallers attribute, 87
- API structure, 59
- APIs (Application Programming Interfaces)
 - asynchronous, 238–240
 - storage client
 - for Queue service applications, 215–218
 - for Table service, 260–263
- APP (Atom Publishing Protocol), 396
- App.config file, 63, 126, 417, 437
- AppendOnly option, 294
- AppFabric Caching service, 485–495
 - ASP.NET framework
 - enabling output cache in AppFabric Caching service, 494
 - Session State provider, 493–494
 - cache clients, 487–493
 - assembly references, 487–488
 - configuring, 488
 - programming AppFabric Caching service, 492–493
 - vs. other cache providers, 486–487
 - provisioning caches, 487
- AppFabric contracts
 - netEventRelayBinding binding, 425
 - NetOnewayRelayBinding binding, 407–408
 - NetTcpRelayBinding binding, 434–435
- AppFabric platform, 32–34
- AppFabric Service Bus, 381–483
 - background of, 381–382
 - ESB, 382–383
 - connectivity infrastructure, 383
 - enterprise naming scheme, 383
 - interface contracts, 383
 - Security and Access Control service, 383
- introduction to, 387–401
 - messaging fabric, 397–401
 - Naming service, 393–395
 - security, 388–393
 - service registry, 395–397
- ISB, 384–387
- message buffer, 452–458
- messaging, Queues and Topics, 458–459
- programming with, 401–452
 - HTTP relay bindings, 442–452
 - netEventRelayBinding binding, 423–429
 - NetOnewayRelayBinding binding, 406–422
 - NetTcpRelayBinding binding, 431–442
 - ProAzure Energy service, 403–406
 - Queues and Topics, 462–482
 - Queues, 459–461
 - Topics, 461–462
- application configuration files, configuring
 - cache clients using, 488–489
- application features, of SQL Azure service, 502–503
- Application Integration option, 362, 369
- Application Integration section, ACS
 - Management Portal, 362
- application migration, 552
- Application Migrations, 31, 44
- Application Programming Interfaces. *See* APIs
- application roles, 52–53
- applications
 - cross-enterprise, 337–339
 - enterprise cloud, 334–336
 - gateway, design, 547
 - hosting login pages as part of, 364–365
 - Java server, 19
 - netEventRelayBinding binding, 429
 - NetOnewayRelayBinding binding, 420–422
 - NetTcpRelayBinding binding, 440–442
 - programming for ACS, 366–377
 - claims-based authorization, 374–377
 - configuring ACS using Management Portal, 367
 - designing ACS rules, 373
 - identity providers, 367
 - Passive Federation case using ACS, 367
 - relying party, 367–369, 371–373
 - rule groups and rules for mapping claims, 368
 - WS-Federation provider, 371–372
 - programming for message buffers, 454–458
 - creating and deleting, 455–456
 - policies, 454–455
 - sample application, 457–458
 - sending messages to, 456
 - Queue service, storage client API for, 215–218
 - runtimes, 52
 - upgrade and fault domains in context of, 55–57
- architecture, advice on, 128–129
- ASP.NET framework
 - enabling output cache in AppFabric Caching service, 494
 - Session State provider, 493–494

- ASP.NET Membership, 305
- assembly references, for cache clients, 487–488
- asynchronous APIs, 238–240
- Atom feed, 256–257, 271–272, 282
- Atom Publishing Protocol (APP), 396
- Attach Debugger, 91–92
- Authentication attribute, 93
- authentication relay, with ACS integration, 389–392
- authorization, claims-based, 374–377
- Authorization SharedKey, 144
- Auto-scale toolkit, 28
- Azure database, 27, 39, 344, 512, 527, 529, 536, 547, 553, 558
- Azure platform, 1, 8, 11, 14, 30, 33, 38, 40, 43, 45
- Azure Storage service queues, AppFabric
 - Service Bus Queues vs., 460–461
- Azure Table storage, 263, 300, 304

■ B

- backslash character, 252
- bandwidth costs, 318
- BeginGetMessage() method, 238
- BI (Business Intelligence), 45, 546
- bin directory, 91, 93
- bindings
 - AppFabric Service Bus, 399–400
 - relay, HTTP, 442–452
 - service, 409–411
 - netEventRelayBinding binding, 426
 - NetTcpRelayBinding binding, 436
- blade servers, 51
- Blob API, 147–154
- Blob class, 148
- Blob method, 173
- Blob service
 - architecture of, 138–142
 - blobs, 140
 - containers, 139
 - storage account, 139
 - limitations and constraints of, 138
- blobs, 140
 - operations of, 172–186
 - Copy Blob, 184–186
 - Get Blob, 180–183
 - Put Blob, 175–180
 - storage scenarios, 200–204
 - Enterprise File Sync tool, 203–204
 - massive data uploads, 200–201

- Storage as a Service model in cloud, 201–203
- types of, 140–142
 - Block blob, 141–142
 - Page blob, 141
- Block blob, 141–142
- BPOS (Business Productivity Online Suite), 10
- BrokeredMessage object, 465, 467–468
- btnSubmit_Click event, 289
- buffers, message, 401
- bursts, 46
- Business Intelligence (BI), 45, 546
- business logic migration, 552
- Business Productivity Online Suite (BPOS), 10
- businesses, and cloud services, 9

■ C

- C# class, 149, 217, 262
- cache clients, 487–493
 - assembly references, 487–488
 - configuring, 488
 - programmatically, 490–491
 - using application configuration file, 488–489
 - programming AppFabric Caching service, 492–493
- caches
 - output, ASP.NET framework, 494
 - provisioning, 487
- caching services, 32–33. *See also* AppFabric Caching service
- CanonicalizedHeaders, 213
- CanonicalizedResource, 213, 255
- CDA (cost-driven architecture), 34
- CDN (Content Delivery Network), 187–189
- Cerebrata, 26
- certificates, 342, 345, 359–360
 - installing, 310
 - management of, 88
- Certificates and Keys section, of ACS
 - Management Portal, 359
- Certificates folder, 124
- Certificates tab, 67
- CGI Web role, 61
- City property, 269
- claims, 328–330
 - claims-based authorization, 374–377
 - claims-based identity model, 330–333
 - mapping, rule groups and rules for, 368

- cleanOnRoleRecycle attribute, 83
- Clear Messages operation, 229
- Clear Results button, 227
- client designs
 - netEventRelayBinding binding, 428–429
 - NetOnewayRelayBinding binding, 418–420
 - NetTcpRelayBinding binding, 438–439
- client layer, of SQL Azure service, 501
- Client object, 456
- ClientCertificate property, 124
- ClientProxy object, 109
- ClientProxy.cs file, 108
- cloud applications, 4–5, 9, 43–44
- Cloud bursts, 46
- cloud computing, Storage as a Service model in, 201–203
 - encryption and decryption, 203
 - integration with enterprise domain accounts, 202
 - storage taxonomy design, 202–203
- Cloud platform, 5, 9
- cloud service roles, 69
- cloud services, 1–6
 - ecosystem of
 - businesses, 9
 - enablers, 9
 - independent software vendors, 8–9
 - service providers, 8
 - software vendors, 8
 - enterprise cloud application, 334–336
 - ISV, 339–341
 - Microsoft strategy for, 9–10
 - providers of, 5–6
 - shifting to, 6–7
 - terminology, of cloud services industry, 2–6
 - types of, 3
- Cloud Storage Studio, 26
- CloudBlob class, 190
- CloudBlobClient class, 150
- CloudBlobContainer object, 166
- CloudBlob.Create() method, 191
- CloudBlobDirectory class, 171
- CloudDrive class, 190
- CloudDrive.Copy() method, 192
- CloudDrive.Mount() method, 191
- CloudDrive.Snapshot() method, 191
- CloudDrive.Unmount() method, 192
- CloudQueue class, 217–218, 222, 225, 228, 232, 236, 238–239
- CloudQueue object, 225, 232
- CloudQueueClient class, 217, 222
- CloudQueueMessage class, 217–218, 232, 236, 238–239
- CloudQueueQueue object, 228
- CloudStorageAccount class, 149–150, 217, 261–262, 282–283, 302–303
- CloudTableClient class, 262, 274, 277, 282
- CloudTableClient object, 277
- CloudTableQueryTElement class, 262, 304
- CLR (Common Language Runtime), 252, 498
- code-far connectivity, 505–506
- code-near connectivity, 503–505
- Common Language Runtime (CLR), 252, 498
- Community Technology Preview (CTP), 497–498, 555
- compute architecture, 13–14
- Compute service, 18
 - application roles, 52–53
 - application runtimes, 52
 - architectural advice, 128–129
 - bibliography, 130
 - blade servers, 51
 - developing services, 58–93, 100–120
 - API structure, 59
 - developer environment, 59–77
 - development fabric, 90–92
 - development storage, 92–93
 - diagnostics and inter-role communication, 101–120
 - objectives of, 101
 - publishing, 111–120
 - running HelloAzureCloud service, 109–111
 - SDK tools, 77–78
 - service models, 78–88
 - diagnostics, 94–100
 - Fabric Controller service, 51
 - geo-location, 120–123
 - Hypervisor program, 51
 - networking components, 51
 - operating systems, 51
 - security, 57–58
 - service development life cycle, 127–128
 - Service Management API, 123–126
 - programming with, 124–126
 - structure of, 123–124
 - service model, 52
 - upgrade domains and fault domains, 53–57
- ComputeHash() method, 144, 213, 255
- configuration files, application, 488–489
- Configuration tab, 64–65
- ConfigurationSettings element, 80

- Configure Remote Desktop Connections link, 316
- Connect feature, 22–23, 317–324
 - Activated Endpoints and Groups and Roles folders, 319
 - enabling for role, 322–323
 - groups, 324
 - installing and activating endpoint on local machine, 320–321
 - protocols and ports, 322
 - provisioning, 318–319
 - vs. Service Bus feature, 318
- connectivity
 - code-far, 505–506
 - code-near, 503–505
- connectivity infrastructures, 383
- ConnectivityMode property, 410
- containers
 - operations of, 160–171
 - Create Container, 162–164
 - List Blobs, 166–171
 - Set Container Metadata, 164–166
 - overview, 139
- Content Delivery Network (CDN), 187–189
- contracts
 - AppFabric, 407–408
 - netEventRelayBinding binding, 425
 - NetTcpRelayBinding binding, 434–435
 - interface, 383
- Coordinated Universal Time (UTC), 143, 212, 255
- Copy Blob operation, 184–186
- Copy method, 184
- CopyBlob() method, 186
- CopyTo() method, 198
- CorrelationFilterExpression expression, 462
- cost-driven architecture (CDA), 34
- crash dumps, 95, 99
- Create Container operation, 162–164
- Create Queue operation, 218, 223–226
- Create table, 528, 535
- Create Table operation, 256, 262, 269–274
- CreateContainer() method, 164
- CreateKeys() method, 269
- CreateQueue() method, 225–226
- CreateTable() method, 274
- CredentialType property, 390
- cross-enterprise application, 337–339
- csmanage.exe, 59, 124
- CSPack.exe, 78
- CSRun.exe, 78, 91

- Csupload.exe, 78
- CTP (Community Technology Preview), 497–498, 555
- curtailment, processes for, 531

■ D

- DaaS (Data as a Service), 3
- dash (-) character, 210
- Data Access component, 29
- data access, connectivity
 - code-far, 505–506
 - code-near, 503–505
- Data as a Service (DaaS), 3
- data definition (DDL), 520, 549–551
- data definition migration, 549–551
- data manipulation (DML), 520
- data migrations, 44, 551–552
- data sources, defining, 556
- Data Sync service, 29–30, 558–559
- data types, 252
- database features, of SQL Azure service, 502
- Database Manager tool, connecting to SQL
 - Azure service database with, 520–524
- database mirroring, 551
- DATABASE statement, 516
- database tables, 536–537
- databases
 - growth-management strategies for, 553–554
 - migration strategies for, 549–552
 - application, 552
 - business logic, 552
 - data, 551–552
 - data definition, 549–551
 - SQL Azure service, 510–512, 530
- dataCacheClients, 489, 491
- DataCacheFactory, 489–492
- DataCacheSecurity, 491
- DataCacheServerEndpoint, 490–491
- DataGridView control, 236
- DataMarket, 13, 40–42
- DataServiceContext class, 262, 265–266, 277, 290, 294
- DataServiceContext object, 277, 294
- DataServiceQuery class, 277–278
- datasets, query and defining, 556
- DateTime property, 250
- DDL (data definition), 520, 549–551
- DeadLetter() method, 467

- debugging, in Windows Azure Tools for Visual Studio, 71–73
- decryption, encryption and, 203
- Default.aspx file, 63, 99, 108, 114, 116
- default.aspx page, 376
- Default.aspx.cs file, 289
- Definition file, 116
- Delete Entity operation, 278–279
- Delete Message from Subscription command, 475
- Delete Message operation, 228–229
- Delete Queue operation, 218, 223
- Delete Table operation, 262, 269–270, 274
- DeleteTable() method, 274
- Dem-Res database, 535–536, 538, 540
- Dem-Res (Demand-Response) service, 547–548
- deployment phase, 128
- DequeueCount property, 238
- Description object, 463
- design and development phase, 127
- Destructive Read command, 475
- developer environment, 59–77
 - packaging service, 73–77
 - role settings and configuration, 63–68
 - Certificates tab, 67
 - Configuration tab, 64–65
 - Endpoints tab, 66–67
 - Local Storage tab, 67
 - Settings tab, 65–66
 - Virtual Network tab, 68
 - Windows Azure Tools for Visual Studio
 - debugging in, 71–73
 - project actions, 69–70
 - project types, 59–63
- development fabric, 90–92
- development, of services. *See* service development
- development storage, 92–93
- devstorageaccount, 219, 223–224, 229, 270, 278–279
- DFS (Distributed File System), 135
- DFUI.exe, 91
- Diagnostic Monitor service, 99
- DiagnosticMonitorConfiguration class, 98
- diagnostics, 94–100
 - and inter-role communication, 101–120
 - service model, 102–104
 - Web role, 108–109
 - Worker role, 104–108
 - logging, 96–100
 - configuring trace listener, 96–97
 - diagnostics service, 97–100
- Dictionary object, 466
- digital identities
 - overview, 327–328
 - providers of, 345, 350–355
- DiscoveryMode property, 396
- DisplayResultsSqlCommand command, 529
- disruptive technology, 1
- Distributed Caching, 43
- Distributed File System (DFS), 135
- DML (data manipulation), 520
- DNS (Domain Name System), 393
- documentation, of ACS Management Portal, 365
- Domain Name System (DNS), 393
- domains
 - accounts, enterprise, 202
 - upgrade and fault, 53–57
- drives
 - creating locally, 192–195
 - mounting, 196–197
 - snapshotting, 197–198
 - uploading, 196
 - Windows Azure. *See* Windows Azure Drives
- DSInit.exe, 78, 93
- Dynamic Scaling, 43

■ E

- E-mailAddressSuffixes element, 364–365
- Each database, 29
- EAV (Entity-Attribute-Value), 497
- Edit Claim Rule page, 358
- Edit Group button, 324
- EnableAdminAccess() method, 375
- enablers, of cloud services, 9
- encryption, and decryption, 203
- Encryption attribute, 93
- Endpoint References section, of ACS Management Portal, 365
- endpoints
 - installing and activating on local machine, 320–321
 - overview, 80–83
 - service
 - netEventRelayBinding binding, 427
 - NetOnewayRelayBinding binding, 417
 - NetTcpRelayBinding binding, 438
- Endpoints tab, 66–67, 315
- EnergyMeterValues table, 539

- enterprise cloud application, 334–336
- enterprise domain accounts, integration of
 - Storage as a Service model with, 202
- Enterprise File Sync tool, 203–204
- Enterprise Identity Integration, 44
- enterprise naming schemes, 383
- enterprise scenario, 44
- Enterprise Service Bus (ESB), 382–383
- entities, 249
- Entity-Attribute-Value (EAV), 497
- entity operations, 278–295
 - Insert Entity, 287–291
 - Merge Entity, 291–295
 - Query Entities, 279–286
- EntryDate property, 265, 267, 269
- EnumerationResults element, 221
- errors, during message processing, 468
- ESB (Enterprise Service Bus), 382–383
- ESB interface, 383
- ETag value, 256, 292, 294
- Event Log data, 99
- Events table, 251
- evt.Set() method, 240
- Execute() method, 278
- Execute Operation, 126
- ExpirationTime property, 217, 233–234

■ F

- Fabric Controller service, 18, 20, 51–52, 54, 56, 78, 116, 120, 310
- Facebook, 332–334, 341, 350–351, 353–354, 357, 379
- Factory class, 464
- FastCGI, 19
- Fault Domain, 115–116
- fault domains, upgrade domains and, 53–57
- federated identities, 379
- FederationMetadata.xml file, 371–372
- Feedback property, 264
- Fiddler Tool, 215, 259
- FIFO (First In First Out), 207
- file management, 308
- First In First Out (FIFO), 207
- Flexibility attribute, 93
- forward slash (/) character, 252
- foundational message components,
 - foundational components of, 465
- foundational scenario, 43
- front-end servers, 135

- FTP-based interface, 383
- FTP interface, 382
- Full IIS (Internet Information Services) support, 85–86
- full trust execution, 86–88

■ G

- gateway applications, design, 547
- Gateway database, 533–535
- Gateways table, 535, 538, 540
- GatewayService class, 448
- Generate link, 368
- geo-location, 120–123
- Geo-replication, 30–31, 43
- geographic affinity, enabling, 121–123
- Get Blob operation, 180–183
- Get Machine Info button, 108–110
- Get Messages operation, 214, 228–230, 232–238
- GET operation, 141, 255
- Get Queue command, 473–474
- Get Queue Metadata operation, 223
- Get Rule command, 482
- Get Storage Service Properties, 299
- Get Subscription command, 480
- Get Topic command, 478
- GetBlobContentsAsFileIfModified() method, 183
- GetDefaultCache() method, 490–492
- GetMessage() method, 217
- GetMessageBuffer() method, 456
- GetMessages() method, 218, 236
- GetQueueReference() method, 217–218, 225, 232, 236, 238–239
- GFS (Global Foundation Services), 9
- Global identity providers, 363
- Google, 332–333, 341, 345, 350, 355, 366–367, 377
- Groups and Roles folder, Activated Endpoints folder and, 319
- groups, Connect feature, 324
- growth-management strategies, for databases, 553–554
- guaranteed processing, 243–245

■ H

- Hash-based Message Authentication Code (HMAC), 143, 207, 255
- headers

- headers (*cont.*)
 - request, 143–145, 212–213, 254–256
 - response, 145, 214, 257
 - Heating Ventilation Air Conditioning (HVAC), 404, 461
 - HelloAzureCloud service, 62, 78, 80, 102, 104, 109–111, 115, 117
 - HelloAzureWorldLocalCache, 79, 104
 - HelloService folder, 101
 - HelloServiceImpl class, 105–108
 - HelloWebRole, 79, 84, 89, 102, 104, 108
 - HelloWorkerRole class, 79, 81, 83, 89, 102, 104–105, 108, 118
 - high-growth sites, 45
 - High Scale compute, 45
 - HMAC (Hash-based Message Authentication Code), 143, 207, 255
 - Hosted Service model, 111, 117–118, 121, 124
 - hosted services
 - deploying, 316–317
 - and Service Definition file, 313–316
 - hosting, service
 - netEventRelayBinding binding, 427
 - NetOnewayRelayBinding binding, 417–418
 - NetTcpRelayBinding binding, 438
 - Host.Open() method, 418
 - HTTP-based interface, 383
 - HTTP (Hypertext Transfer Protocol)
 - relay bindings, 442–452
 - WebHttpRelayBinding, 446–452
 - WS2007HttpRelayBinding, 443–446
 - verbs, 142, 212–254
 - HTTP messages, 253
 - HTTP request, 253–254, 290
 - HttpIn input endpoint, 114
 - HVAC (Heating Ventilation Air Conditioning), 404, 461
 - hybrid applications, 45
 - Hyper-V, 307, 309
 - Hypertext Transfer Protocol. *See* HTTP
 - Hypervisor program, 51
- |
- IaaS (Infrastructure as a Service), 2
 - IAsyncCallback object, 240
 - IBM Global Services, 9
 - IDE (integrated design environment), 94
 - identity federation, 379
 - identity management, 43
 - identity models, claims-based, 330–333
 - identity providers, 367, 377
 - IEnergyServiceGatewayOperations interface, 435
 - IHelloService interface, 105–109
 - IHybridConnectionStatus interface, 442
 - IIS (Internet Information Server), 15
 - images
 - coordination between Fabric Controller service and, 310
 - uploading, 312–313
 - viewing in management portal, 313
 - ImageUrl element, 364–365
 - implementers, 9
 - IMulticastGatewayOperations interface, 425
 - Independent Software Vendor (ISV), 45, 339–341
 - information technology (IT), 1
 - Infrastructure as a Service (IaaS), 2
 - infrastructure layer, of SQL Azure service, 499
 - Infrastructure logs, 94, 99
 - Initialize() method, 20
 - input claims, 378–379
 - Insert Entity operation, 278–279, 287–291
 - InsertEnergyMeterValues stored procedure, 539
 - InsertGateway stored procedure, 538–539
 - InsertionTime property, 217, 233–234
 - InsertPricingCalendar_kWh stored procedure, 538
 - InsertPricingLocations stored procedure, 538
 - Install Local Endpoint option, 320
 - instance count, 65
 - integrated design environment (IDE), 94
 - Integration components, 309–311
 - IntelliTrace feature, 15, 71, 76, 94, 114, 117–118
 - inter-role communication, diagnostics and, 101–120
 - service model, 102–104
 - Web role, 108–109
 - Worker role, 104–108
 - interface contracts, 383
 - internal endpoint, 66–67, 79, 81–82, 106, 108–109
 - Internet Information Server (IIS), 15
 - Internet Information Services (Full IIS) support, 85–86
 - Internet Service Bus (ISB), 384–387
 - interoperable messaging, 242–243
 - invalid messages, 467
 - IOnewayEnergyServiceOperations interface, 409

- IPAddress, 107, 115
- IPEndPoint object, 83
- IPv6 link, 322
- ISB (Internet Service Bus), 384–387
- .iso file, 309, 311
- IsOneWay=true property, 408
- ISV (Independent Software Vendor), 45, 339–341
- IT (information technology), 1

■ J, K

- Java server applications, 19

■ L

- LastUpdatedBy, 226
- life cycles, of service development, 127–128
- LINQ statement, 280
- List Blobs operation, 166–171
- List Queues command, 474
- List Queues operation, 214, 218–219, 222
- List Rules command, 482
- List Subscriptions command, 480
- List Topics command, 478
- ListContainers() method, 158
- ListContainersSegmented() method, 158
- ListQueue() method, 222
- ListQueues() method, 217–218, 222
- ListTables() method, 277
- load distribution, Worker roles, 241–242
- local endpoint software, 320, 322
- local storage, 83, 311
- Local Storage tab, 67
- LocalResource class, 83
- LocalStorage element, 80, 83
- LocalSystem, 310
- log shipping, 551
- logged fields, 297
- logging, 96–100
 - configuring trace listener, 96–97
 - diagnostics service
 - defining storage location for, 97
 - starting, 98–100
- Logging feature, 296–297
- login pages
 - hosting as part of application, 364–365
 - linking to ACS-hosted, 362–363
- Login Pages section, of ACS Management Portal, 362–365
- logins, 516–520
- LoginUrl element, 364–365
- LogoutUrl element, 364–365
- logs, IntelliTrace feature, 117–118

■ M

- maintenance phase, 128
- Management API, 27–28
- management commands, 471–475, 480–482
 - creating REST API Queue, 472
 - creating REST API Topic, 477
 - creating rule, 481
 - creating subscription, 479
 - deleting REST API Queue, 473
 - deleting REST API Topic, 477–478
 - deleting rule, 482
 - deleting subscription, 480
 - Get Queue, 473–474
 - Get Rule, 482
 - Get Subscription, 480
 - Get Topic command, 478
 - List Queues, 474
 - List Rules, 482
 - List Subscriptions, 480
 - List Topics command, 478
 - QueueDescription, 471
 - Rule Description document, 481
 - SubscriptionDescription, 478
 - subscriptions, 478
 - TopicDescription, 476
- Management Portals
 - for ACS, 342–365
 - ADFS 2.0, 345–348
 - Application Integration section, 362
 - Certificates and Keys section, 359
 - digital identities, 345, 350–355
 - Endpoint References section, 365
 - Login Pages section, 362–365
 - Management Service section, 362
 - Portal Administrators area, 361
 - provisioning ACS service namespace, 342–345
 - Relying Party applications, 355–357
 - Rule Groups section, 357–358
 - SDKs and Documentation section, 365
 - Service Identities section, 361
 - tokens, 359–360
 - configuring ACS using, 367
 - overview, 38–40

- Management Portals for ACS (*cont.*)
 - viewing images in, 313
- Management Service section, of ACS
 - Management Portal, 362
- Manager table, 523
- mapping claims, 368
- marker parameter, 219
- Marketplace DataMarket broker, 41–42
- massive data uploads, 200–201
- maxresults parameter, 219
- Merge Entity operation, 278–279, 291–295
- MESQ (Microsoft Messaging Queue), 459
- Message Body text box, 231
- message buffers, programming applications
 - for, 452–458
 - creating and deleting message buffer, 455–456
 - message buffer policies, 454–455
 - message buffer sample application, 457–458
 - sending messages to message buffer, 456
- message commands, 469–471, 474–475
 - Delete Message from Subscription, 475
 - deleting from REST API Queue, 471
 - Read and Delete Message from Subscription (Destructive Read), 475
 - Read Message from Subscription with Non-Destructive Peek-Lock, 474
 - receiving from REST API Queue, 470
 - sending to REST API Queue, 470
 - sending to REST API Topic, 474
 - unlocking message from subscription, 475
- message operations, 228–238
 - Get Messages, 232–238
 - Put Message, 230–232
- message security
 - netEventRelayBinding binding, 426
 - NetOnewayRelayBinding binding, 413–417
 - NetTcpRelayBinding binding, 437
 - overview, 392–393
- MessageBufferClient class, 453–454, 456
- MessageBufferClient.CreateMessageBuffer()
 - method, 454–455
- MessageBufferPolicy class, 453–454
- MessageID attribute, 210
- MessageReceived event, 236–237
- MessageReceivedEventArgs class, 236–237
- MessageReceivedEventHandler, 236–237
- MessageReceiver object, 466
- messages, 210–211, 465–467
 - buffers, 401
 - creating and sending to AppFabric Service Bus Queue, 466
 - creating and sending to Topic, 466
 - foundational components of, 465
 - problems and abandonment, 467–468
 - invalid or poison messages, 467
 - server error during message processing, 468
 - retrieving from AppFabricService Bus Queue, 466
 - retrieving using subscriptions, 466–467
 - sending to message buffer, 456
 - unlocking from subscription, 475
- MessageText element, 230
- Messagettl attribute, 211
- messagettl parameter, 211, 230
- messaging
 - fabric, 397–401
 - AppFabric Service Bus bindings, 399–400
 - message buffer, 401
 - Queues and Topics features, 401
 - interoperable, 242–243
 - Queues and Topics, 458–459
- Metrics feature, 297–298
- Microsoft Consulting Services, 9, 28
- Microsoft datacenters, 9
- Microsoft Message Queuing (MSMQ), 207, 384
- Microsoft Messaging Queue (MESQ), 459
- Microsoft, strategy for cloud services, 9–10
- Microsoft Technology Center (MTC), 201
- Microsoft Windows Azure platform
 - AppFabric platform, 32–34
 - cloud services, 1–6
 - ecosystem of, 8–9
 - industry terminology, 2–6
 - Microsoft strategy for, 9–10
 - providers of, 5–6
 - shifting to, 6–7
 - terminology, 4
 - types of, 3
 - Compute service, 18
 - Connect feature, 22–23
 - description of, 10–14
 - Management API, 27–28
 - management portal, 38–40
 - Marketplace DataMarket broker, 41–42
 - pricing of, 34–38
 - scenarios found in, 43–46
 - enterprise, 44
 - foundational, 43

- ISV, 45
- SQL Azure service, 28–31
- storage service, 24–26
- VM role, 21
- Web role, 19
- Worker role, 19–21
- Microsoft.ApplicationServer.Caching.Client.dll, 487
- Microsoft.ApplicationServer.Caching.Core.dll, 488
- microsoft.identitymodel section, 371
- Microsoft.Samples.WindowsAzure.ServiceManagement.dll file, 124
- Microsoft.ServiceBus namespace, 462–463
 - adding session state to Queue, 463
 - NamespaceManager class, 463
 - NamespaceManagerSettings class, 462
- Microsoft.ServiceBus.Messaging namespace, 464
- Microsoft.ServiceBus.ServiceRegistrySettings object, 396
- Microsoft.Web.DistributedCached.dll, 488
- Microsoft.WindowsAzue.ServiceRuntime.dll, 105
- Microsoft.WindowsAzure.Diagnostics
 - assembly, 59, 96–97, 99, 105
- Microsoft.WindowsAzure.Diagnostics
 - namespace, 59
- Microsoft.WindowsAzure.ServiceRuntime, 59, 62–63, 105–106
- Microsoft.WindowsAzure.ServiceRuntime.Role
 - EntryPoint class, 20
- Microsoft.WindowsAzure.ServiceRuntime.Role
 - Environment class, 83
- Microsoft.WindowsAzure.StorageClient
 - assembly, 26, 59
- Microsoft.WindowsAzure.StorageClient library, 105, 215, 217, 236
- migration strategies, for databases, 549–552
 - application migration, 552
 - business logic migration, 552
 - data definition migration, 549–551
 - data migration, 551–552
- MonAgentHost.exe, 95
- mounting drives, 196
- m:properties element, 292
- MSMQ (Microsoft Message Queuing), 207, 384
- MTC (Microsoft Technology Center), 201
- multi-tenancy, 43
- MyCloudDb database, 516, 526
- MyFirstAzureTable, 271–273

MyInternalEndpoint, 79, 81, 105–106, 108

■ N

- NamespaceManager class, 463–464
- NamespaceManagerSettings class, 462–463
- NamespaceManger object, 462
- namespaces, ACS service, 342–345
- naming schemes, enterprise, 383
- Naming service, 393–395
- NAT (network address translation), 386
- .NET Client API, 462–468
 - foundational message components, 465
 - messages, 465–467
 - creating and sending to Queue, 466
 - creating and sending to Topic, 466
 - problems and abandonment, 467–468
 - retrieving from Queue, 466
 - retrieving using subscriptions, 466–467
- Microsoft.ServiceBus namespace, 462–463
 - adding session state to Queue, 463
 - NamespaceManager class, 463
 - NamespaceManagerSettings class, 462
- Microsoft.ServiceBus.Messaging
 - namespace, 464
- .NET Client library, 253, 259
- .NET Trust Level, 65
- NetDataContractSerializer class, 493
- netEventRelayBinding binding, 423–429
 - AppFabric contract, 425
 - applications, 429
 - client design, 428–429
 - message security, 426
 - relay security, 426
 - service binding, 426
 - service endpoints, 427
 - service hosting, 427
 - service implementation, 425–426
- NetOnewayRelayBinding binding, 406–422
 - AppFabric contract, 407–408
 - applications, 420–422
 - client design, 418–420
 - message security, 413–417
 - relay security, 411–413
 - service binding, 409–411
 - service endpoints, 417
 - service hosting, 417–418
 - service implementation, 409
- NetOnewayRelayServer.exe command, 421
- NetTcpRelayBinding binding, 431–442

NetTcpRelayBinding binding (*cont.*)

- AppFabric contract, 434–435
 - applications, 440–442
 - client design, 438–439
 - message security, 437
 - relay security, 437
 - service binding, 436
 - service endpoints, 438
 - service hosting, 438
 - service implementation, 435–436
- NetTcpRelayBinding.exe command, 440
- network address translation (NAT), 386
- network latency, 318
- networking, components of, 51
- New Hosted Service, 111
- New Virtual Machine Role option, 314
- NextMarker element, 221
- NextMarker property, 262
- NextPartitionKey parameter, 280, 282
- NextRowKey parameter, 280, 282
- NextVisibleTime property, 217
- NoSQL option, 247
- NoTracking option, 294
- number sign (#) character, 252
- numofmessages parameter, 232–233, 236

■ O

- OASIS (Organization for the Advancement of Structured Information Standards), 400
- OData, 41–42
- OnStart() method, 20, 104, 106
- operating systems, 51
- Organization for the Advancement of Structured Information Standards (OASIS), 400
- output caches, ASP.NET framework, 494
- output claims, 379
- outputcache, 489
- OverwriteChanges option, 294

■ P

- PaaS (Platform as a Service), 2
- packaging, service, 73–77
- Page blob, 141
- Page method, 173
- Page_Load method, 374–375
- partition layer, 135, 250–251, 267, 291

- PartitionKey property, 250–251, 253, 264, 267, 280, 283, 285, 291, 294
- PartnerAccess, 337–339
- Partner_Contractor, 337
- Partner_Employee, 337
- Partner_Manager, 337
- Passive Federation case, using ACS, 367
- Peek Messages operation, 228–229
- PeekLock command, 475
- PeekLock() method, 456
- PerfData property, 302
- performance counters, from Table service, 300–304
- PerformanceCounters, 79, 89, 97–98
- PerformanceData class, 301
- PerformanceDataContext class, 301–302
- Platform as a Service (PaaS), 2
- platform layer, of SQL Azure service, 499–500
- poison messages, 467
- policies, message buffer, 454–455
- PopReceipt attribute, 211, 217, 228, 233–234
- Portal Administrators area, of ACS
- Management Portal, 361
- portal commands, 39
- ports, protocols and, 322
- POST HTTP verb, 271
- prefix parameter, 219
- PreserveChanges option, 285, 294
- pricing, 34–38
- Pricing and Gateway database, design, 534–548
- gateway application design, 547
 - optimizing SELECT queries, 542–544
 - pricing table synchronization, 545–546
 - ProAzure Dem-Res service, 547–548
 - proazuredemres database, 535–536
 - stored procedures, 537–540
 - tables, 536–537
 - uploading sample data, 540–542
- pricing tables, synchronization, 545–546
- PricingCalendar_kWh table, 535, 538–539, 541–543, 545
- PricingLocations table, 535, 538
- Pro Azure Reader Tracker, 263, 282
- ProAzure Dem-Res (Demand-Response) service, 547–548
- ProAzure Energy service, example of, 403–406
- ProAzureACSWeb, 368
- ProAzure.cer certificate, 360
- ProAzureCommonLib, 221–222, 225, 227, 236, 273, 276
- proazuredemres database, 535–536

- ProAzureReader class, 263–265, 267, 269
- ProAzureReader() method, 254, 264, 269, 280, 283, 288, 290–291, 294, 304
- ProAzureReader object, 288, 290, 292
- ProAzureReader table, 273, 280, 285, 288
- ProAzureReaderDataContext class, 265–267, 269, 289
- ProAzureReaderDataSource class, 266–267, 282, 289–290, 293, 295
- ProAzureReaderDataSourceContext object, 290
- ProAzureReaderTracker, 270, 272–273, 276, 285–286, 288, 292–293
- ProAzureReaderTracker_WebRole, 270, 272–273, 276, 285–286, 288, 292–293
- ProAzureSignedCA.cer, 360
- processing, guaranteed, 243–245
- Product object, 383
- programming
 - with AppFabric Service Bus, 401–452
 - HTTP relay bindings, 442–452
 - netEventRelayBinding binding, 423–429
 - NetOnewayRelayBinding binding, 406–422
 - NetTcpRelayBinding binding, 431–442
 - ProAzure Energy service, 403–406
 - Queues and Topics, 462–482
 - with Service Management API, 124–126
- project actions
 - cloud service roles, 69
 - storage services, 70
- project types, in Windows Azure Tools for Visual Studio, 59–63
- properties, 250–253
- protocols, and ports, 322
- provisioning
 - ACS service namespaces, 342–345
 - caches, 487
 - Connect feature, 318–319
- publishing, 111–120
 - Remote Desktop Connection application, 118–120
 - viewing IntelliTrace feature logs, 117–118
- PurchaseDate property, 265
- PurchaseLocation property, 265
- PurchaseType property, 265
- Put Blob operation, 175–180
- Put Block List operation, Put Block operation and, 178–180
- Put Block operation, and Put Block List operation, 178–180
- Put Message operation, 214, 228–232

- PutBlob() method, 178
- PutMessage() method, 232

■ Q

- queries
 - and dataset, defining, 556
 - SELECT, optimizing, 542–544
- Query Entities operation, 257, 278–286
- Query Tables operation, 269–270, 274–278
- question mark (?) character, 252
- Queue class, 216
- queue name, 210–211, 223–224, 226, 229–230, 232
- Queue Name text box, 227, 231
- Queue object, 225
- queue operations, 223–228
 - Create Queue, 224–226
 - Set Queue Metadata, 226–228
- Queue Operations tab, 225, 227
- Queue service, 207–245
 - account operations, 218–222
 - architecture of, 208–211
 - messages, 210–211
 - queues, 210
 - storage account, 209
 - asynchronous API, 238–240
 - bibliography, 245
 - limitations and constraints of, 208
 - message operations, 228–238
 - Get Messages, 232–238
 - Put Message, 230–232
 - queue operations, 223–228
 - Create Queue, 224–226
 - Set Queue Metadata, 226–228
- REST API, 211–218
 - request, 211–214
 - response, 214
 - storage client API, 215–218
- scenarios for, 240–245
 - guaranteed processing, 243–245
 - interoperable messaging, 242–243
 - Web and Worker role communications, 240–241
 - Worker role load distribution, 241–242
- QueueAttributes class, 217
- QueueClient object, 222
- QueueDescription command, 471
- QueueDescription object, 463
- QueueErrorCodeStrings class, 217

- QueueListener class, 236–237
- QueueMessage element, 230, 234
- Queues
 - AppFabric Service Bus, 459
 - adding session state to, 463
 - vs. Azure Storage service queues, 460–461
 - creating and sending messages to, 466
 - retrieving messages from, 466
 - and Topics, 458–459
 - overview, 210
 - REST API
 - management commands, 471–474
 - message commands, 469–471
 - and Topics
 - AppFabric Service Bus, 462–482
 - features of, 401
- Queues element, 221
- Queues List Box, 231

R

- Rackspace, 307
- RDL file, 556
- Read and Delete Message, from Subscription command, 475
- Read Message from Subscription with Non-Destructive Peek-Lock command, 474
- ReaderUrl property, 292–293
- references, assembly, 487–488
- Regions method, 173
- registries, service, 395–397
- relational data storage engine, 29
- relay authentication, with ACS integration, 389–392
- relay bindings, HTTP, 442–452
 - WebHttpRelayBinding binding, 446–452
 - WS2007HttpRelayBinding binding, 443–446
- relay security
 - netEventRelayBinding binding, 426
 - NetOnewayRelayBinding binding, 411–413
 - NetTcpRelayBinding binding, 437
- relay services, 386–387
- relying party
 - applications for, 355–357, 367–371
 - designing claims for, 373
- Relying Party (RP), 345, 348, 355–357, 359, 367, 372
- remote access, packaging and enabling, 316
- Remote Desktop Connections, 76, 114, 118–120

- reporting
 - defining data source, 556
 - defining query and dataset, 556
 - deploying, 556
 - sample of, 555
 - viewing, 556
- Representational State Transfer API. *See* REST
- Request for Security Token (RST), 378
- Request Security Token Response (RSTR), 378
- requests, 211–214, 254–256
 - body of, 214–256
 - components of, 142–146
 - body, 145
 - headers, 143–145
 - HTTP verb, 142
 - request URI, 142–143
 - URI parameters, 143
 - headers, 212–213, 254–256
 - HTTP verb, 212–254
 - securing with Access Control Service, 469
 - URI, 212–254
- responses, 214, 256–259
 - body of, 214, 257–259
 - components of, 145–146
 - body, 146
 - headers, 145
 - status code, 145
 - headers, 214–257
 - status code, 214–256
- REST-based interface, 27, 123
- REST command, 472–474, 477, 479–481
- REST interface, 59, 146, 224, 446, 448, 471–472, 475, 479, 481
- REST (Representational State Transfer) API, 142–154, 211–218, 253–263, 468–482
- ADO.NET Data Services library, 259
- Queues
 - management commands, 471–474
 - message commands, 469–471
- request, 211–214
 - body of, 214–256
 - headers, 212–213, 254–256
 - HTTP verb, 212–254
 - URI, 212–254
- request components, 142–146
 - body, 145
 - headers, 143–145
 - HTTP verb, 142
 - request URI, 142–143
 - URI parameters, 143
- response, 256–259

- body of, 214, 257–259
- headers, 214, 257
- status code, 214, 256
- response components, 145–146
 - body, 146
 - headers, 145
 - status code, 145
- rules, management commands, 481–482
- securing requests with Access Control Service, 469
- storage client API, 215–218, 260–263
 - for Queue service applications, 215–218
 - for Table service, 260–263
- StorageClient API, 146–154
 - Blob API, 147–154
- Topics
 - management commands, 475–480
 - and subscriptions, 474–475
- REST Request body, 299
- REST Request header, 299
- REST-style interface, 399, 446
- ResultSegment class, 305
- ResultSegment object, 304
- RetryPolicy property, 305
- ROI (return on investment), 49
- role instances, creating drives from, 196–197
- RoleEntryPoint class, 20, 62–63
- RoleEnvironmentChanging event, 106
- RoleEnvironmentConfigurationSettingChange, 106
- RoleEnvironment.RequestRecycle() method, 106
- RoleEnvironment.StatusCheck event, 106
- RoleEnvironmentTopologyChange, 106
- RoleIds, 115
- RoleInstanceStatusCheckEventArgs, 106
- roles
 - application, 52–53
 - cloud service, 69
 - enabling Connect feature for, 322–323
 - settings and configuration of, 63–68
 - Certificates tab, 67
 - Configuration tab, 64–65
 - Endpoints tab, 66–67
 - Local Storage tab, 67
 - Settings tab, 65–66
 - Virtual Network tab, 68
- RowKey, 250, 252, 257, 265, 268, 279, 281, 285, 291, 294
- RP (Relying Party), 345, 348, 355–357, 359, 367, 372

- RST (Request for Security Token), 378
- RSTR (Request Security Token Response), 378
- Rule Description document, 481
- Rule Groups section, 348, 350, 357, 368
- RuleDDescription object, 481
- rules
 - CorrelationFilterExpression expression, 462
 - designing for ACS, 373
 - for mapping claims, rule groups and, 368
 - REST API, management commands, 481–482
 - SQLFilterExpression expression, 462
- Run() method, 20, 63, 106
- runtimes, application, 52

■ S

- SaaS (Software as a Service), 2
- SalesAccess, 337, 339
- SAML (Security Assertions Markup Language), 377, 391
- SAML token, 332–333, 338, 348, 377–379
- sample data, uploading, 540–542
- SAN (storage area network), 202
- SaveChanges() method, 269, 290, 294–295
- SaveChangesOptions, 284, 289
- Scalability attribute, 93
- scenarios, 43–46
 - enterprise, 44
 - foundational, 43
 - ISV, 45
- SDK class, 106
- SDK Command Prompt, 312
- SDKs and Documentation section, of ACS Management Portal, 365
- SDS (SQL Data Services), 497
- Secure Sockets Layer (SSL), 345, 378, 501
- SecureString, 491
- security, 388–393
 - Compute service, 57–58
 - message, 392–393
 - netEventRelayBinding binding, 426
 - NetOnewayRelayBinding binding, 413–417
 - NetTcpRelayBinding binding, 437
- relay
 - authentication with ACS integration, 389–392
 - netEventRelayBinding binding, 426

- security, relay (*cont.*)
 - NetOnewayRelayBinding binding, 411–413
 - NetTcpRelayBinding binding, 437
- Security and Access Control service, 383
- Security Assertions Markup Language (SAML), 377, 391
- SELECT command, 529
- SELECT queries, optimizing, 542–544
- SelfSTS tool, 371–373
- Send() method, 456
- SendKwhValue() method, 419
- SendLightingValue() method, 419
- Server database, 335, 498, 501, 549–551, 553, 559
- server errors, during message processing, 468
- servers
 - blade, 51
 - Java, applications, 19
 - SQL Azure service, 507–508
- service bindings
 - netEventRelayBinding binding, 426
 - NetOnewayRelayBinding binding, 409–411
 - NetTcpRelayBinding binding, 436
- Service Bus feature, 13, 32, 46, 318
- Service Definition file, hosted services and, 313–316
- service descriptions, 531
- service development, 58–93, 100–120
 - adding diagnostics and inter-role communication, 101–120
 - service model, 102–104
 - Web role, 108–109
 - Worker role, 104–108
 - API structure, 59
 - developer environment, 59–77
 - packaging service, 73–77
 - role settings and configuration, 63–68
 - Windows Azure Tools for Visual Studio, 59, 69–73
 - development fabric, 90–92
 - development storage, 92–93
 - diagnostics and inter-role communication, 101–120
 - service model, 102–104
 - Web role, 108–109
 - Worker role, 104–108
 - life cycle of, 127–128
 - objectives of, 101
 - publishing, 111–120
 - Remote Desktop Connection
 - application, 118–120
 - viewing IntelliTrace feature logs, 117–118
 - running HelloAzureCloud service, 109–111
 - SDK tools, 77–78
 - service models, 78–88
 - certificate management, 88
 - ServiceConfiguration.cscfg file, 88–90
 - ServiceDefinition.csdef file, 78–88
- service endpoints
 - netEventRelayBinding binding, 427
 - NetOnewayRelayBinding binding, 417
 - NetTcpRelayBinding binding, 438
- service hosting
 - netEventRelayBinding binding, 427
 - NetOnewayRelayBinding binding, 417–418
 - NetTcpRelayBinding binding, 438
- Service Identities section, 361
- service implementations
 - netEventRelayBinding binding, 425–426
 - NetOnewayRelayBinding binding, 409
 - NetTcpRelayBinding binding, 435–436
- Service Level Agreement (SLA), 53
- Service Management API, 123–126
 - programming with, 124–126
 - structure of, 123–124
- service models, 52, 78–90, 102–104
 - certificate management, 88
 - ServiceConfiguration.cscfg file, 88–90
 - ServiceDefinition.csdef file, 78–88
 - endpoints, 80–83
 - Full IIS support, 85–86
 - full trust execution, 86–88
 - local storage, 83
 - startup tasks, 83–84
- [Service Name].cspkg, 76
- service namespaces, ACS, provisioning, 342–345
- service providers, of cloud services, 8
- service queues, Azure Storage, 460–461
- service registries, 395–397
- Service Settings section, 345
- ServiceBusEnvironment.CreateServiceUri() method, 438
- ServiceConfiguration.cscfg file, 76, 87–90, 97, 99, 102, 273
- ServiceDefinition.csdef file, 78–88
 - endpoints, 80–83
 - Full IIS support, 85–86
 - full trust execution, 86–88

- local storage, 83
- startup tasks, 83–84
- services layer, of SQL Azure service, 500–501
- services, packaging, 73–77
- session state, adding to AppFabric Service Bus Queue, 463
- Session State provider, ASP.NET framework, 493–494
- sessionState, 489, 493–494
- Set Container Metadata operation, 164–166
- Set Queue Metadata operation, 223–224, 226–228
- SetBusy() method, 106
- SetContainerMetadata method, 166
- SetContainerMetadata() method, 166
- SetMetadata() method, 218, 228
- Settings tab, 65–66
- SetupDiagnostics() method, 98–99, 105–106
- Shared Access Signatures, 152–154
- SharedKey, 212–213, 220–221, 224, 226, 230, 233
- ShowSignInPageType, 365
- Simple Web Token (SWT), 332, 379
- Size attribute, 93
- SLA (Service Level Agreement), 53
- snapshotting, drives, 197–198
- Software as a Service (SaaS), 2
- software modernization, 46
- software vendors, of cloud services, 8–9
- Springsource, 5
- SQL Azure service, 28–31, 497–560
 - architecture of, 498–501
 - data access, 503–506
 - Data Sync service, 558–559
 - database, 510–512
 - connecting to, 512–530
 - growth-management strategies for, 553–554
 - migration strategies for, 549–552
 - developing services that use, 530–548
 - Pricing and Gateway database design, 534–548
 - processes for curtailment, 531
 - service description, 531
 - technical architecture, 532–534
 - limitations of, 501–502
 - overview of, 497–498
 - reporting, 555–556
 - defining data source, 556
 - defining query and dataset, 556
 - deploying, 556
 - sample of, 555
 - viewing, 556
- server, 507–508
- SSRS feature fidelity, 558
- supported features of, 501–503
 - administration, 503
 - application, 502–503
 - database, 502
- SQL command, 522, 525
- SQL Data Services (SDS), 497
- SQL Server BCP utility, 551
- SQL Server Integration Services (SSIS), 545–546, 551
- SQL Server Management Studio tool,
 - connecting with, 512–520
- SQL Server Reporting Services (SSRS), 555, 558
- SQLAzure database, 11, 24, 29, 31, 33–34
- SQLAzure service, Table service vs., 299–300
- SQLCMD tool, connecting to SQL Azure service database with, 524–526
- SqlCommand command, 528, 542
- SqlConnectionStringBuilder class, 528
- SQLFilterExpression expression, 462
- sqlInstance parameter, 93
- SSIS (SQL Server Integration Services), 545–546, 551
- SSL (Secure Sockets Layer), 501
- SSRS (SQL Server Reporting Services), 555, 558
- startup tasks, 83–84
- state management, 307
- stateless role interfaces, 128
- status codes, 145, 214, 256
- Status method, 126
- Stop() method, 20
- storage
 - development, 92–93
 - local, 83, 311
 - locations, defining for diagnostics service, 97
 - scenarios for, 200–204
 - Enterprise File Sync tool, 203–204
 - massive data uploads, 200–201
 - Storage as a Service model in cloud, 201–203
 - taxonomy design of, 202–203
- storage accounts, 139, 154–157, 209–248
- Storage Analytics API, 296–299
 - enabling, 298–299
 - Logging feature, 296–297
 - Metrics feature, 297–298
- storage area network (SAN), 202
- Storage as a Service model, in cloud, 201–203

- Storage as a Service model, in cloud (*cont.*)
 - encryption and decryption, 203
 - integration with enterprise domain accounts, 202
 - storage taxonomy design, 202–203
- Storage Client API, 215, 217, 238
- Storage Client library, 253, 264
- storage clients, API, 260–263
- Storage Emulator, 70, 93
- Storage Management, 44
- Storage Operations.exe application, 221, 234
- Storage service, 24–26, 70, 131–205
 - account operations, 154–157
 - architecture of, 134–137
 - Blob service, 137–138
 - architecture of, 138–142
 - limitations and constraints of, 138
 - blobs
 - operations of, 172–186
 - storage scenarios, 200–204
 - CDN, 187–189
 - container operations, 160–171
 - Create Container, 162–164
 - List Blobs, 166–171
 - Set Container Metadata, 164–166
 - programming example, 157–160
 - REST API, 142–154
 - request components, 142–146
 - response components, 145–146
 - StorageClient API, 146–154
 - taxonomy of, 132–134
 - Windows Azure Drives, 189–200
 - operations of, 190–198
 - overview of, 189–190
 - scenarios for, 198–200
- storage types, 305
- StorageAccountConnectionString, 222
- StorageClient API, Blob API, 147–154
- StorageClient library, 240
- StorageClient methods, 221
- StorageClient project, 221
- stored procedures, 537–540
 - InsertEnergyMeterValues, 539
 - InsertGateway, 538–539
 - InsertPricingCalendar_kWh, 538
 - InsertPricingLocations, 538
 - UpdateGatewayLastCommunication, 540
- Stream object, 465
- STS (Secure Token Service), 345, 378
- subscription rules, 462
- SubscriptionDescription command, 478

- SubscriptionDescription object, 478
- subscriptionId, 125
- SubscriptionId, 126
- subscriptions
 - deleting, 480
 - REST API management commands, 478
 - REST API Topics and, message commands, 474–475
 - retrieving messages using, 466–467
 - unlocking messages from, 475
- Summary of Table Service Functionality, 259
- SWT (Simple Web Token), 332, 379
- Symmetric Key, 359, 362
- synchronization, options for, 559
- system integrators, 9
- System Preparation tool, 310–312
- System.Collection.Specialized.NameValueCollection object, 166
- System.Convert.ToBase64String() method, 144, 255
- System.Data.Services.Client.DataServiceContext class, 262, 265, 277
- System.Data.SqlClient.SqlConnectionStringBuilder class, 527
- System.Diagnostics.Trace class, 97
- System.IO file, 83
- System.Net.HttpWebRequest, 124, 226, 228, 232
- System.ServiceModel.Channels.Message object, 448, 456

■ T

- T-Press, 334, 336
- T-Room, 334–336
- Table class, 261
- Table names, 249
- table operations, 269–278
 - Create Table, 270–274
 - Query Tables, 274–278
- Table service
 - architecture of, 247–253
 - entity, 249
 - property, 250–253
 - storage account, 248
 - table, 249
 - example model, 263–269
 - paging in, 304–306
 - scenarios for, 300–306
 - paging in Table service, 304–306

- reading performance counters from
 - Table service, 300–304
- vs. SQLAzure service, 299–300
- storage client API for, 260–263
- TableClient object, 277
- TableOperations.aspx page, 273
- TableOperations.aspx page", 276
- tables, 247–306
 - account operations, 269
 - bibliography, 306
 - database, 536–537
 - entity operations, 278–295
 - Insert Entity, 287–291
 - Merge Entity, 291–295
 - Query Entities, 279–286
 - pricing, synchronization, 545–546
 - REST API, 253–263
 - ADO.NET Data Services library, 259
 - request, 254–256
 - response, 256–259
 - storage client API, 260–263
 - Storage Analytics API, 296–299
 - enabling, 298–299
 - Logging feature, 296–297
 - Metrics feature, 297–298
 - table operations, 269–278
 - Create Table, 270–274
 - Query Tables, 274–278
 - Table service
 - architecture of, 247–253
 - example model, 263–269
 - scenarios for, 300–306
 - vs. SQLAzure service, 299–300
- TableServiceContext class, 262, 268, 282, 301
- TableServiceEntity class, 262, 264, 267, 301
- TableStorage object, 277
- TableStorageDataServiceContext class, 265
- TableStorageEntity class, 264
- tabular data stream (TDS), 28, 500
- TargetServerURL property, 556
- taskType attribute, 79, 84, 104
- TcpListener, 109
- TDS (tabular data stream), 28, 500
- technical architecture, 532–534
- testing phase, 127
- third-party applications, 44
- TokenProvider object, 463
- tokens
 - decryption, 360
 - encryption, 360
 - retrieving from ACS, 341–342

- signing, 359–360
- TopicDescription command, 476
- Topics, 461–462
 - AppFabric Service Bus Queues and,
 - messaging, 458–459
 - creating and sending messages to, 466
- Queues and
 - AppFabric Service Bus, 462–482
 - features of, 401
- REST API
 - management commands, 475–480
 - and subscriptions, 474–475
 - subscription rules, 462
- trace listeners, configuring, 96–97
- Trace.WriteLine() statements, 106
- TransportClientEndpointBehavior class, 390, 411, 417–418
- TreeView control, 171, 182, 185
- Trusted Root Certificate Authorities folder, 360

■ U

- UDP protocol, 19
- Uniform Resource Identifiers. *See* URIs
- Unmount() method, 198
- Update Entity operation, 278–279
- UpdateGatewayLastCommunication stored procedure, 540
- UpdateUrl() method, 293–295
- Upgrade Domain, 53–57, 108, 116–117
- UploadFromStream() method, 179
- uploads, massive data, 200–201
- URI parameter, 143, 154, 164, 169, 212, 219, 254, 274, 474–475
- URIs (Uniform Resource Identifiers)
 - parameters of, 143
 - request, 142–143, 212–254
- Use an Existing STS option, 370
- user claims, 378
- UserProfiles table, 253
- UTC (Coordinated Universal Time), 143, 212, 255

■ V

- VHD Verification Tool, 313
- VHD (virtual hard drive), 21
- Virtual Hard Disk tab, 314
- virtual hard drive (VHD), 21
- Virtual Machine role. *See* VM

- Virtual Network tab, 68
- Visibilitytimeout attribute, 211
- visibilitytimeout parameter, 211, 228, 232–234, 236, 243–244
- Visual Studio command, 360
- Visual Studio, Windows Azure Tools for. *See* Windows Azure Tools for Visual Studio
- VM Role, 18, 21–22
- VM size option, 65
- VM (Virtual Machine) role, 21, 307–317
 - benefits and tradeoffs of, 308
 - hosted services
 - deploying, 316–317
 - and Service Definition file, 313–316
 - images
 - coordination between Fabric Controller service and, 310
 - uploading, 312–313
 - viewing in management portal, 313
 - installing certificates, 310
 - Integration components, 309–311
 - local storage resources, 311
 - packaging and enabling remote access, 316
 - scenarios for, 308–309
 - System Preparation tool, 310–312
- WADDiagnosticInfrastructureLogsTable, 95
- WADLogsTable, 95
- WADPerformanceCountersTable, 96, 300
- WADWindowsEventLogsTable, 96
- WaIntegrationComponents-x64.msi, 311
- WaitOne() method, 240
- WASStorageHelper class, 158, 166, 222, 225, 227, 273, 276–277
- WASStorageHelper.cs file, 178, 183, 186
- WCF method, 106
- WCF (Windows Communications Foundation), 19, 33, 379, 382, 399
- web applications, with multiple identity providers using ACS, 367
- Web Resource Authorization Protocol (WRAP), 379
- Web roles
 - Java server applications, 19
 - overview, 108–109
 - and Worker roles, communications between, 240–241
- web.config file, 371
- WebHttpRelayBinding binding, 446–452
- WebRole class, 62, 86
- WIF (Windows Identity Foundation), 328, 359, 366–367, 377, 379–380
- Windows Azure AppFabric Cache, 32
- Windows Azure Cloud Service template, 60
- Windows Azure Drives
 - operations of, 189–200
 - creating drive locally, 192–195
 - mounting drive, 196
 - snapshotting drive, 197–198
 - uploading drive, 196
 - overview of, 189–190
 - scenarios for, 198–200
- Windows Azure Management portal, 342
- Windows Azure Tools for Visual Studio, 59
 - debugging in, 71–73
 - project actions
 - cloud service roles, 69
 - storage services, 70
 - project types, 59–63
- Windows Azure Trace logs, 94
- Windows Communications Foundation (WCF), 19, 33, 379, 382, 399
- Windows event logs, 94
- Windows Identity Foundation (WIF), 328, 359, 366–367, 377, 379–380
- Windows Live, 327, 332, 334, 341, 345, 350, 355, 364, 367, 377
- Windows performance counters, 94
- WindowsAzureStorage.cs file, 232
- WinHttpCertCfg.exe certificate, 360
- Worker roles, 19–21, 104–108
 - load distribution, 241–242
 - Web roles and, communications between, 240–241
- WRAP (Web Resource Authorization Protocol), 379
- WS-Federation Metadata section, 369
- WS-Federation providers, 363, 371–372, 375
- WS2007HttpRelayBinding binding, 443–446

■ X

- x-ms-request-id, 214, 220, 224, 231, 233, 257, 272, 275, 280
- X.509 certificates, 360

■ **Y**

Yahoo!, 345, 350, 366–367

■ **Z**

Zimbra, 5

ZipCode property, 253

Windows Azure Platform

Second Edition



Tejaswi Redkar
Tony Guidici

Apress®

Windows Azure Platform

Copyright © 2011 by Tejaswi Redkar and Tony Guidici

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-4302-3563-7

ISBN-13 (electronic): 978-1-4302-3564-4

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

President and Publisher: Paul Manning

Lead Editor: Ewan Buckingham

Technical Reviewer: Todd Meister

Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh

Coordinating Editor: Jessica Belanger

Copy Editor: Tracy Brown

Indexer: BiM Indexing & Proofreading Services

Artist: SPI Global

Compositor: Bytheway Publishing Services

Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at www.apress.com/bulk-sales.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at www.apress.com. You will need to answer questions pertaining to this book in order to successfully download the code.

*I dedicate this book to my late grandfather, Shri. Sharad Atmaram Redkar.
My name is his creation. I thank my wife, Arohi, and my sons Aaryan (Heart-of-Gold) and
Dhruv for giving me the liberty to do what I love. I also thank my sister Aasawari for being with
me when I needed her. I thank my parents for all their teachings.
Finally, I thank God for gifting me with the people I love.*

–Tejaswi Redkar

*This book is dedicated to my father, James Guidici, for being the first major technical influence
in my life – in particular for buying me my first computer (a Texas Instruments TI-99/4A),
teaching me how to write GOTO loops, and how to dial into BBSes on my IBM PCJr.
Your passion for technology spurred a lifetime love of technology in me. More important,
I thank you for being an outstanding role model and for showing me how to be a good man
and good father.*

*I thank my mother, Deborah Guidici, for always believing that I could accomplish anything.
Her confidence and encouragement helped me to tackle challenges I wouldn't have thought
possible, such as writing a book.*

*To my wife of nearly 15 years, Jenifer, thank you for supporting me during the writing of this
book. There aren't enough words to describe my gratitude for the sacrifices you've made to
support my career and our family. You are truly amazing.*

*To my sons, Joseph and Nicholas, you are my daily inspiration. I hope that you grow up with the
same belief that I did, that you can accomplish anything as long as you reach for the stars, and
you should never settle for less than your best.*

–Tony Guidici

Contents

■ About the Authors	xvi
■ About the Technical Reviewer	xvii
■ Acknowledgments	xviii
■ Introduction	xx
■ Chapter 1: Windows Azure Platform Overview	1
Introducing Cloud Services	1
Industry Terminology	2
Cloud Service Providers	5
Shifting to the Cloud Paradigm	6
Understanding the Cloud Services Ecosystem	8
Service Providers	8
Software Vendors	8
Independent Software Vendors	8
Enablers	9
Businesses	9
Microsoft's Cloud Strategy	9
Windows Azure Platform Overview	10
Understanding Windows Azure Compute Architecture	13
Windows Azure	15
Compute	18

Windows Azure Storage	24
Management.....	27
SQL Azure	28
Windows Azure AppFabric	32
Windows Azure Platform Pricing	34
Management Portal – Let's Provision	38
Windows Azure Marketplace DataMarket	41
Windows Azure Platform Common Scenarios	43
Foundational Scenarios	43
Enterprise Scenarios	44
ISV Scenarios.....	45
Summary	46
Bibliography.....	46
■ Chapter 2: Windows Azure Compute	49
Compute Service.....	49
Upgrade Domains and Fault Domains	53
Compute Service Security	57
Developing Windows Azure Services.....	58
Windows Azure API Structure.....	59
Developer Environment	59
Windows Azure SDK Tools.....	77
Service Models	78
Development Fabric.....	90
Development Storage	92
Diagnostics	94
Logging.....	96

Developing Windows Azure Services with Inter-Role Communication	100
Objectives	101
Adding Diagnostics and Inter-role Communication	101
Running the HelloAzureCloud Service	109
Publishing to Windows Azure Cloud	111
Geo-location	120
Enabling Geographic Affinity	121
Windows Azure Service Management	123
Service Management API Structure.....	123
Programming with the Service Management API.....	124
Windows Azure Service Development Life Cycle.....	127
Architectural Advice.....	128
Summary	129
Bibliography.....	130
■ Chapter 3: Windows Azure Storage Part I – Blobs and Drives	131
Storage Service Taxonomy	132
Storage Service Architecture.....	134
The Blob Service.....	137
Blob Limitations and Constraints.....	138
Blob Architecture	138
Windows Azure Storage Account	139
Containers	139
Blobs.....	140
Types of Blobs	140
REST API	142
Request	142
Response	145

Storage Client API.....	146
Account Operations	154
Programming Example	157
Container Operations	160
Create Container.....	162
Set Container Metadata.....	164
List Blobs.....	166
Blob Operations	172
Put Blob	175
Get Blob	180
Copy Blob.....	184
Content Delivery Network (CDN).....	187
Windows Azure Drives	189
Overview.....	189
Drive Operations.....	190
Windows Azure Drives Scenarios.....	198
Blob Storage Scenarios	200
Massive Data Uploads	200
Storage as a Service in the Cloud.....	201
Enterprise File Sync.....	203
Summary	205
Bibliography.....	205
■ Chapter 4: Windows Azure Storage Part II – Queues.....	207
Queue Limitations and Constraints.....	208
Queue Service Architecture	208
Windows Azure Storage Account	209
Queues.....	210

Messages	210
REST API	211
Request	211
Response	214
Storage Client API	215
Account Operations	218
Queue Operations	223
Create Queue	224
Set Queue Metadata	226
Message Operations	228
Put Message	230
Get Messages	232
Asynchronous API	238
Queue Scenarios	240
Scenario 1: Windows Azure Web and Worker Role Communications	240
Scenario 2: Worker Role Load Distribution	241
Scenario 3: Interoperable Messaging	242
Scenario 4: Guaranteed Processing	243
Summary	245
Bibliography	245
■ Chapter 5: Windows Azure Storage Part III – Tables	247
Table Service Architecture	247
REST API	253
Example Table Model	263
Account Operations	269
Table Operations	269
Entity Operations	278

Storage Analytics.....	296
Logging.....	296
Metrics.....	297
Enabling Storage Analytics.....	298
Table Storage versus SQLAzure.....	299
Table Service Scenarios	300
Scenario 1: Reading Performance Counters from Table Storage	300
Scenario 2: Paging in Table storage.....	304
Summary	306
Bibliography.....	306
■ Chapter 6: VM Role and Windows Azure Connect.....	307
VM Role.....	307
VM Role Benefits/Tradeoffs	308
Scenarios.....	308
Creating the Virtual Machine	309
Windows Azure VM Role Integration Components.....	309
Upload Image to Windows Azure.....	312
Windows Azure Connect.....	317
Windows Azure Connect vs. Service Bus	318
Provisioning Windows Azure Connect	318
Activated Endpoints, Groups, and Roles.....	319
Installing and Activating an Azure Endpoint on a Local Machine	320
Enabling Windows Azure Connect for a Role.....	322
Creating Connect Groups.....	324
Summary	325
Bibliography.....	325

■ Chapter 7: AppFabric: Access Control Service	327
What Is Your Digital Identity?.....	327
What Are Claims?	328
Claims-Based Identity Model.....	330
Access Control Service Usage Scenarios	333
Scenario 1: Enterprise Cloud Application	334
Scenario 2: Cross-Enterprise Application	337
Scenario 3: ISV Cloud Service	339
Retrieving Tokens from ACS.....	341
Access Control Service Management Portal.....	342
Provisioning Your ACS Service Namespace	342
Identity Providers.....	345
Relying Party	355
Rule Groups	357
Certificates and Keys.....	359
Service Identities	361
Portal Administrators.....	361
Management Service.....	362
Application Integration	362
Programming Access Control Service Applications.....	366
Passive Federation with ACS.....	367
Web Application: Multiple Identity Providers using ACS.....	367
Adding a WS-Federation Provider	371
Summary	377
Concepts and Terminology	377
Identity Provider	377
Relying Party	377

Security Token (SAML Token).....	378
Secure Token Service (STS)	378
Request for Security Token (RST).....	378
Request Security Token Response (RSTR)	378
Claim	378
Identity Federation.....	379
Windows Identity Foundation (WIF)	379
Active Directory Federation Server (ADFS 2.0).....	379
Web Resource Authorization Protocol (WRAP) and Simple Web Token (SWT).....	379
Bibliography.....	380
■ Chapter 8: AppFabric Service Bus	381
First, a Little Background.	381
Enterprise Service Bus (ESB).....	382
Security and Access Control.....	383
Connectivity Infrastructure	383
Enterprise Naming Scheme.....	383
Interface Contracts	383
Evolution of the Internet Service Bus (ISB).....	384
Relay Service.....	386
Introduction to the AppFabric Service Bus	387
Security	388
Naming Service	393
Service Registry.....	395
Messaging Fabric	397
Programming with the AppFabric Service Bus.....	401
ProAzure Energy Service Example.....	403
NetOnewayRelayBinding	406
netEventRelayBinding.....	423

NetTcpRelayBinding	431
HTTP Relay Bindings.....	442
Message Buffer.....	452
Programming Message Buffer Applications	454
AppFabric Messaging: Queues and Topics	458
AppFabric Service Bus Queues.....	459
AppFabric Service Bus Queues vs. Azure Storage Queues.....	460
AppFabric Service Bus Topics	461
Subscription Rules.....	462
Programming Service Bus Queues and Topics.....	462
.NET Client API.....	462
REST API.....	468
Summary	482
Bibliography.....	483
■ Chapter 9: AppFabric: Caching	485
AppFabric Caching vs. Other Cache Providers	486
Provisioning an AppFabric Cache	487
AppFabric Cache Clients.....	487
Assembly References	487
Configuring the Cache Client	488
Programming AppFabric Cache.....	492
ASP.NET Session State Provider	493
Enabling Session State in AppFabric Cache	493
Enabling ASP.NET Output Cache in AppFabric Cache.....	494
Summary	495
Bibliography.....	495

■ Chapter 10: SQL Azure	497
SQL Azure Overview	497
SQL Azure Architecture.....	498
Infrastructure Layer	499
Platform Layer	499
Services Layer	500
Client Layer.....	501
SQL Azure Limitations and Supported Features	501
Database Features.....	502
Application Features.....	502
Administration Features	503
SQL Azure Data Access.....	503
Code-Near Connectivity	503
Code-Far Connectivity	505
Getting Started with SQL Azure	506
Creating a SQL Azure Server	507
Creating a SQL Azure Database.....	510
Connecting to a SQL Azure Database	512
Developing Windows Azure Services That Use SQL Azure	530
Service Description	531
Processes for Curtailment	531
Technical Architecture.....	532
Pricing and Gateway Database Design.....	534
Database-Migration Strategies.....	549
Data Definition Migration.....	549
Data Migration	551
Business Logic Migration	552
Application Migration.....	552

Database Growth-Management Strategies	553
SQL Azure Reporting.....	555
Sample Report.	555
Creating Reports.	555
SSRS Feature Fidelity	558
Data Sync.....	558
Data Sync Design.....	559
Synchronization Options.	559
Summary	559
Bibliography.....	560
■ Index	561

About the Authors



■ **Tejaswi Redkar** is a software architect with a passion for writing. He works for Microsoft and has been working on the Windows Azure platform since 2008. He is also the Worldwide Community Lead for the Windows Azure platform in Microsoft Services. He has architected several small- and large-scale systems on Windows Azure for Enterprises and ISVs. Tejaswi has not only written about conceptual topics like Threading and MSMQ, but also on broader topics, such as software ecosystems, businesses, and platforms. Tejaswi has a Master's degree in Computer Engineering from San Jose State University and an MBA from University of Wisconsin, Whitewater.

Tejaswi lives in the beautiful San Francisco Bay Area with his wife, Arohi, and two sons, Aaryan and Dhruv. When not working on what's next, he is either having fun with the family or bicycling on San Ramon trails. Professionally, he idolizes three people: Bill Gates, Kishore Kumar, and Sachin Tendulkar. You can find more details about him on his LinkedIn profile at www.linkedin.com/in/tejaswiredkar. Follow him at: Twitter: @tejaswi_redkar.



■ **Tony Guidici** has fifteen years' experience as a software developer and architect. He is currently a Senior Architect Evangelist at Microsoft in the Developer & Platform Evangelism (DPE) group. He is on the Azure Incubation team, and is focused on providing technical guidance and assistance to Cloud Service Vendors creating SaaS applications on Windows Azure. Additionally, he works with the product team to provide product feedback from customers.

He has been working with Windows Azure since it was first announced at PDC 2008, and with early adopters since 2009 as part of Microsoft Consulting Services. He believes that the cloud is a generational paradigm shift, and that Windows Azure is the premier cloud platform. He continues to stay sharp by seeking out interesting problems his customers face and helping solve them.

He holds an MBA from the University of Wisconsin-Madison in Information Technology Analysis and Design, and is an avid fan of Badger football. He also holds a BA in International Business from Bradley University, and is looking forward to the next time we see the Braves in the Final Four.

Tony lives in the Chicago suburbs with his wife, Jenifer, and their two sons, Joseph and Nicholas. He spends much of his free time with his family, coaching his kids' baseball teams and patiently waiting for the Cubs to win the World Series.

You can follow him at: Twitter: @tonyguid and via his blog at <http://blogs.msdn.com/tonyguid>.

About the Technical Reviewer

■ **Todd Meister** has been working in the IT industry for over fifteen years. He's been a technical editor on over 75 titles on topics ranging from SQL Server to the .NET Framework. He is also the Senior IT Architect at Ball State University in Muncie, Indiana. He lives in central Indiana with his wife, Kimberly, and their four incredible children.

Acknowledgments

I would like to thank the following individuals for their contributions to my professional and personal life:

- Smt. Laxmi Natarajan, a teacher who believed in the author in me.
- Prof. M.B. Unde at the National Chemical Laboratory, Pune, for teaching me valuable engineering lessons.
- Randy Bainbridge at Microsoft, who is one of the best managers I've had.
- Jamal Haider at Microsoft for encouraging the author in me.
- Ewan Buckingham at Apress for driving the first and second edition of this book.
- Penny Tong, for teaching me valuable work-life lessons.
- Prof. Dan Harkey at San Jose State University for giving me the opportunity to teach.
- The Microsoft leadership team for fostering an atmosphere of innovation.
- Kui Jia, for selling me the Microsoft employment value proposition.
- Tony Guidici for taking up the challenge of creating the second edition of this book.
- Larry Fenster, Jenn Goth, Danny Garber, Anu Chawla, Ken Archer, Kevin Fleck, and Scott Lengel for believing in my work and providing me with necessary feedback and opportunities.

My personal life would be incomplete without a network of amazing friends, co-workers, educators, and students who have played an important role in shaping it.

– Tejaswi Redkar

I would like to thank the following professionals for their contributions:

- Tejaswi Redkar, for having the faith to let a first-time author contribute to your book, and your enduring support.
- Eric Golpe, Windows Azure OneTAP guru, for helping me get started with Azure way back in 2008.

- Ewan Buckingham, Jessica Belanger, and James Markham from Apress for guiding me through the process of authoring this book.
- Danny Garber, for helping me move from “knowledgable” to “expert” in Azure.
- David Makogon (“World’s First Former Azure MVP”), Bhushan Nene, and Kashif Alam of Microsoft, for providing assistance to many technical questions.
- My colleagues on the Worldwide Azure CSV team. Your brilliance drives me to be better every day.
- Clark Sell, my first mentor at Microsoft, for getting me off to a good start.
- Kevin Fleck and Ken Archer of Microsoft Consulting Services, for having the foresight to start a team called Cloud 123, focused on early adopters of Azure and BPOS. The work we did in that group with Danny and Tejaswi laid a great foundation for other teams to follow.

– Tony Guidici

Introduction

Cloud Computing is not just hype anymore. It has graduated to the early stages of maturity where companies have started betting their existing businesses and embarking on new business ventures. The cloud provides opportunity to quickly turn vision into reality, because of the very low acquisition costs. You can open an account with a credit card and start using any public cloud platform in minutes. Architecting Windows Azure solutions is my full-time job, and I have tried to share my experiences in this book. The book covers most of the fundamental concepts in cloud computing through the Windows Azure platform. The Windows Azure platform is a fully functional cloud platform with Compute, Storage, Management, and Middleware services for you to develop distributed applications.

Due to the agility of product releases, many more services will be released after this book has been written. But, I promise to bring these services to light via other channels like public articles, my twitter feed (@tejaswi_redkar), Field Notes (<http://www.microsoft.com/windowsazure/learn/real-world-guidance/>), and other services, so please stay tuned.

My hope is that, after reading this book, you will be able to build Windows Azure applications of your own and start reaping the benefits of the platform by embarking on new opportunities. In each chapter, I have covered scenarios from my real-world experiences working on this platform. There is also a wealth of source code in all the chapters (except Chapter 1) to get you started. You can download the source code either from the publisher's web site at www.apress.com or from the book's CodePlex site at <http://azureplatformbook.codeplex.com>.

Sincerely,
Tejaswi Redkar