Sample Chapter

Azure IN ACTION

Chris Hay
Brian H. Prince

MANNING

Azure in Action

by Chris Hay and Brian H. Prince

Chapter 1

brief contents

Part 1

Welcome to the cloud

Part 1 is all about dipping your toes into the water and getting ready to dive in headfirst.

We cover what Azure is in chapter 1—what the moving parts are, and why people are so excited about cloud computing.

We throw you in the deep end of the pool in chapter 2, building and deploying—step-by-step—your first cloud application. We've all written Hello, World apps; after you've read part 1, you'll begin to see how you can easily scale them to hundreds of servers.

1 Getting to know Windows Azure

This chapter covers

- Overview of Windows Azure
- Building your first Windows Azure web role
- Windows Azure infrastructure
- How Windows Azure implements core cloud concepts
- Flagship Windows Azure platform services

Imagine a world where your applications were no longer constrained by hardware and you could consume whatever computing power you needed, when you needed it. More importantly, imagine a world where you paid only for the computing power that you used.

Now that your imagination is running wild, imagine you don't need to care about managing hardware infrastructure and you can focus on the software that you develop. In this world, you can shift your focus from managing servers to managing applications.

If this is the sort of thing you daydream about, then you should burn your server farm and watch the smoke form into a cloud in the perfect azure sky. Welcome to the cloud, and welcome to Windows Azure. We also suggest that if this is the sort of thing you daydream about, you might want to lie to your non-IT friends.

We'll slowly introduce lots of new concepts to you throughout this book, eventually giving you the complete picture about cloud computing. In this chapter, we'll keep things relatively simple. As you get more comfortable with this new paradigm, and as the book progresses, we'll introduce more of Azure's complexities. To get the ball rolling, we'll start by looking at the big Azure picture: the entire platform.

1.1 What's the Windows Azure platform?

As you might have already gathered, the Windows Azure platform encompasses Microsoft's complete cloud offering. Every service that Microsoft considers to be part of the cloud will be included under this banner. If the whole cloud thing passed you by, there isn't really anything magical about it. The *cloud* refers to a bunch of servers that host and run your applications, or to an offering of services that are consumed (think web service).

The main difference between a cloud offering and a noncloud offering is that the infrastructure is abstracted away—in the cloud, you don't care about the physical hardware that hosts your service. Another difference is that most public cloud solutions are offered as a metered service, meaning you pay for the resources that you use (compute time, disk space, bandwidth, and so on) as and when you use them.

Based on the Azure release announced in November 2009 at the Professional Developers Conference (PDC) held in Los Angeles, the Windows Azure platform splits into the three parts shown in figure 1.1: Windows Azure, SQL Azure, and the Windows Azure platform AppFabric. You can expect the parts included in the platform to

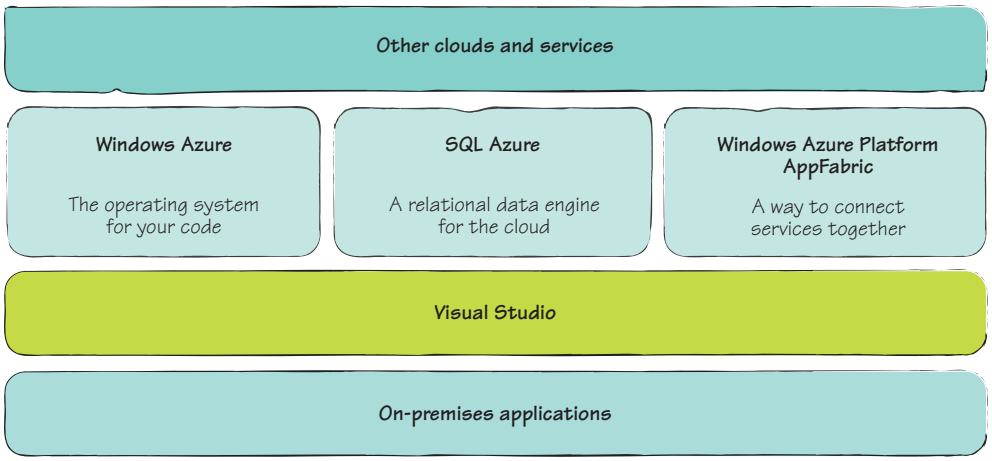


Figure 1.1 The parts that make up the Windows Azure platform include the Windows Azure operating system, SQL Azure, and AppFabric.

increase over time; in fact, we wouldn't be surprised to see Microsoft Flight Simulator in the cloud.

As cool as AppFabric and SQL Azure are, for now we're going to stay focused on the Windows Azure part of the Windows Azure platform and ignore all the other platform-specific stuff until the end of the chapter. Talking about Windows Azure immediately gets a little confusing. Unfortunately, when most folks refer to Windows Azure, it's not clear whether they're referring to the Windows Azure platform, the complete cloud offering, or to Windows Azure, which is a part of the platform.

It's kind of like the ESPN naming convention. The ESPN Network has multiple channels (ESPN, ESPN2, ESPN News, and so on), yet we tend to refer to these channels collectively as ESPN rather than as the ESPN Network. To confuse matters further, we also refer to the individual ESPN channel as ESPN, also. If you ask someone what game is on ESPN tonight, it's not clear if you mean all the channels on ESPN (including ESPN News and ESPN2) or if you mean just the channel named ESPN (not including ESPN2 and the others).To keep things consistent, whenever we talk about the platform as a whole, we'll refer to the *Windows Azure platform* or *the platform*; but if we're talking about the core Windows Azure product, then we'll use the term *Windows Azure*, or just *Azure*.

So, what exactly is Windows Azure? Microsoft calls Azure its core operating system for the cloud. OK, so now you know what Windows Azure is, and we can skip on, right? Not so fast! Let's break it down, strip away all the hype, and find out what Azure is all about.

1.1.1 Windows is in the title, so it must be an operating system

Windows Azure is an operating system that provides the ability to run applications in a highly scalable manner on Microsoft servers, in Microsoft's data centers, in a manageable way. You can host either your web applications, such as a website that sells Hawaiian shirts, or backend processing services, such as an MP3-to-WMA file converter, in Microsoft's data centers.

If you need more computing power (more instances of your website or more instances of your backend service) to run your application, you can allocate more resources to the application, which are then spread across many servers. By increasing the number of resources to your application, you'll ultimately be able to process more data or handle more incoming traffic.

Hmmm…how exactly is that an operating system? To answer that question, we have to define what it means to be a *cloud operating system*.

When Microsoft refers to Windows Azure as an operating system for the cloud, it doesn't literally mean an operating system as you might know it (Windows 7, Windows Vista, Leopard, Snow Leopard, and so on). What Microsoft means is that Windows Azure performs jobs that are similar to those that a traditional operating system might perform. What does an operating system do? Well, it has four tasks in life:

- Host and run applications
- Remove the complexities of hardware from applications

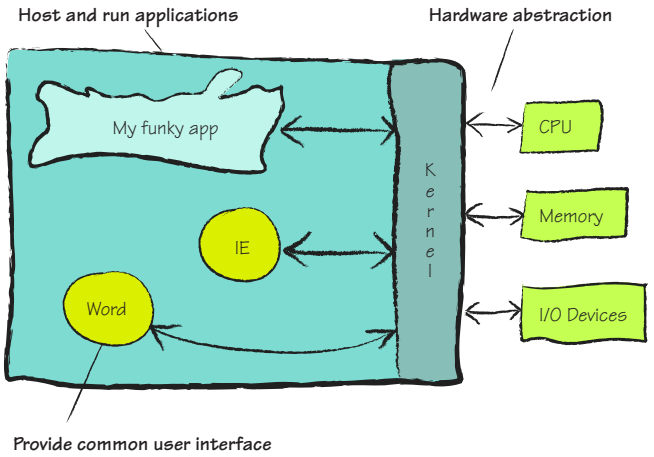


Figure 1.2 A typical representation of an operating system interacting with applications and resources. Notice that applications don't directly interact with CPU, memory, or I/O devices.

- Provide an interface between users and applications
- Provide a mechanism that manages what's running where and enforces permissions in the system

Figure 1.2 shows how a traditional operating system achieves these tasks in a typical PC environment.

The applications shown in figure 1.2 are running within an operating system. The applications don't have direct access to the hardware; all interactions must come through the kernel, the low-level operating system component that performs all the tasks we're discussing: processing, memory management, and device management. We'll look at how some components of Windows Azure fill the role of the kernel in the cloud later in this chapter.

The analogy of Windows Azure being an operating system looks like it could work out after all. Over the next few sections, we'll use this analogy to see how Windows Azure fares as an operating system, which will give you a good overview of how Windows Azure works and what services it provides.

1.1.2 Hosting and running applications the Azure way

Hosting and running applications might be the most important task of an operating system. Without applications, we're just moving a mouse around with no purpose. Let's look at the types of applications that can be run in both traditional operating systems and in Windows Azure.

TYPES OF APPLICATIONS: WHAT'S IN A NAME?

In a traditional operating system, such as Windows 7, we can consider most of the following to be applications:

- Microsoft Word (yep, it's an app)
- Internet Explorer or Firefox (still an app)
- Killer Mutant Donkey Zombie Blaster game (even that's an app)

Remember those applications running in the context of a typical PC operating system in figure 1.1? Instead of hosting client applications (games, Word, Excel, and so on), the types of applications that you host in Windows Azure are server applications, such as web applications (for example, a Hawaiian Shirt Shop website) or background computational applications (for example, an MP3 file converter).

Figure 1.3 shows these server applications running in a traditional operating system.

Turns out (see figures 1.2 and 1.3) that there's no real difference between Microsoft Excel and a Hawaiian Shirt Shop website. As far as a traditional operating system is concerned, they're both applications.

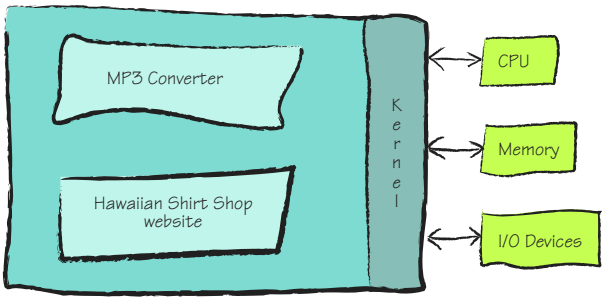


Figure 1.3 Windows Azure–type applications running in a traditional OS. Azure applications function in an OS the same way that traditional applications do.

RUNNING APPLICATIONS ACROSS THOUSANDS OF SERVERS

The traditional operating system is responsible for allocating CPU time and memory space that allows your application to run (as seen in both figures 1.1 and 1.2). Not only is the operating system responsible for allocating these resources, but it's also responsible for managing these resources. For example, if an application fails, then it's the operating system's job to clean up the application's resource usage and restart the application, if necessary. This level of abstraction is perfect for an operating system that manages a single server, but it isn't scalable when it comes to a cloud operating system. With Windows Azure, your application doesn't necessarily run on a single server; it can potentially run in parallel on thousands of servers.

A cloud operating system can't be responsible for allocating CPU time and memory on thousands of physically separate servers. This responsibility has to be abstracted away from the OS. In Windows Azure, that responsibility is given to *virtual machines* (VMs). Figure 1.4 shows how your applications might be distributed among the VMs in a Windows Azure data center.

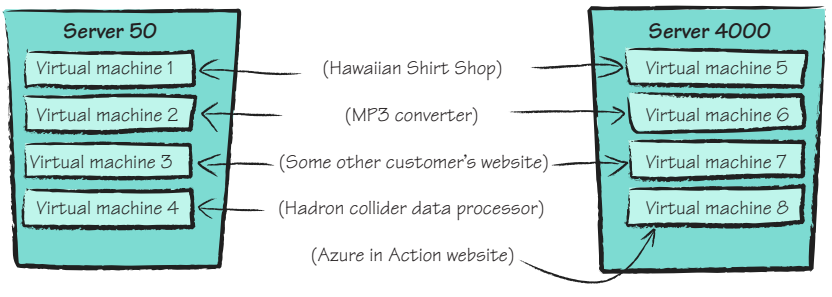


Figure 1.4 Applications split across many VMs in a Windows Azure data center

Your cloud operating system is no longer responsible for assigning your applications' resources by CPU and memory, but is instead responsible for allocating resources using VMs. Windows Azure uses VMs to achieve separation of services across physical servers. Each physical server is divided into multiple VMs. An application from another customer on the same physical hardware as yours won't interfere with your application.

In figure 1.4, the Hawaiian Shirt Shop website is allocated across two VMs (VM1 and VM5), which are hosted on two different physical servers (server 50 and server 4000), whereas the *Azure in Action* website is allocated only a single VM (VM8) on server 4000 (shirt shops make more money, so they get more resources).

Let's drill down and take a closer look at what constitutes a VM.

ANATOMY OF A VIRTUAL MACHINE

Figure 1.5 shows what the VM hosting a web application looks like.

The physical server is split up into one or more VMs. Every instance of your service (web role or worker role) is installed onto its own VM, which is a base installation of Windows Server 2008 (with some extra Azure bits). The VM hosts the web application within Internet Information Services (IIS) 7.0.

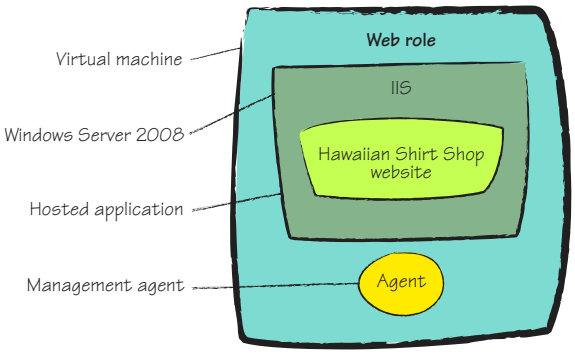


Figure 1.5 A logical representation of the VM that hosts your web application

Although your application runs on a VM, the VM is abstracted away from you, and you only have a view of the role instance, never of the VM. A single instance of your web application is assigned to a single VM, and no other applications will be assigned to that VM. In this way, every instance of your web application is isolated from other applications running on the same physical server. The VM image also runs an agent process. We'll explain what this agent does in chapter 3 when we discuss the Red Dog Agent.

Web role and worker role

A role is another name for your application. The role refers to the base VM image that hosts your application. A web role is a VM that hosts your application within IIS. A worker role is the same as a web role, but without IIS. It's intended for typical backend processing workloads.

To be honest, we're now itching for some code. Let's look at how you can build a simple ASP.NET website that you can run in one of those Windows Azure VMs. Don't worry; we'll continue dissecting Windows Azure after you get your hands dirty with a little code.

1.2 Building your first Windows Azure web application

Although you're going to build an ASP.NET website in this example, the good news is that almost any website that can currently be hosted in IIS on Windows Server 2008 can be hosted in Windows Azure.

The following are examples of the types of web applications Azure supports out of the box:

- ASP.NET 3.5 web applications
- ASP.NET MVC 1.0, 2.0 web applications
- Web services (WCF, ASMX)
- Any FastCGI-based website such as PHP or Python
- Java and Ruby applications

Although Windows Azure supports the ability to host different types of websites, for now you'll create a simple Hello World web application using ASP.NET 3.5 SP1. In chapters 7 and 15, we'll look at how you can create PHP websites, WCF Web Services, and ASP.NET MVC websites.

To get started developing an ASP.NET 3.5 SP1 website, you'll need to download the Windows Azure software development kit (SDK).

1.2.1 Setting up your environment

The SDK contains a whole bunch of things that'll make your life easier when developing for Windows Azure, including the following:

- Windows Azure development fabric (a simulation of the live fabric)
- Visual Studio templates for creating web applications
- Windows Azure storage environment
- Deployment tools
- A glimpse of a bright new world

In chapter 2, we'll take a deeper look at some of the items in the SDK. For now, you'll just install it. If you're an experienced ASP.NET developer, you should be able to install the SDK by clicking the Next button a few times. You can grab the SDK from www.Azure.com.

Before installing the SDK, check your version of Windows and Visual Studio. A local instance of some flavor of SQL Server (either Express, which is installed with Visual Studio, or full-blown SQL Server) is required to use the SDK. We'll explain this in more depth in chapter 9.

SUPPORTED OPERATING SYSTEMS

Before you attempt to install the SDK, make sure that you have a suitable version of Windows. Supported versions of Windows currently include the following:

- Windows 7 (which you should be running because it's lovely)
- Windows Vista
- Windows Server 2008 (and beyond)

NOTE Windows XP isn't supported by Windows Azure. Before you jump up and down about Windows XP, there isn't some conspiracy against it. XP isn't supported because Windows Azure web roles are heavily built on IIS 7.0. Windows XP and Windows 2003 use earlier versions of IIS that won't work with Windows Azure.

SUPPORTED VERSIONS OF VISUAL STUDIO

To develop Windows Azure applications in Visual Studio, you'll need either Visual Studio 2008 or Visual Studio 2010. If you're still running Visual Studio 2005, you now have the excuse you need to upgrade. If for some reason you can't get Visual Studio or your company won't upgrade you, then you can use the Web Express versions of either Visual Studio 2008 or 2010 for free, or you can use Visual Studio 2008. We'll be using Visual Studio 2010 throughout this book. The windows and dialog boxes shown in the figures might differ slightly from those in Visual Studio 2008 or the Express Edition, but, all in all, it works in the same way.

STARTING VISUAL STUDIO

To launch your Windows Azure application in the development fabric from Visual Studio, you need Administrator privileges. Get into the habit (for Azure development) of right-clicking your Visual Studio icon and selecting Run as Administrator.

Now we'll help you create your first Azure web application.

1.2.2 Creating a new project

Your first step is to create a new project. Open Visual Studio and select File > New > Project. Select the Cloud Service project type, which gives you the option to select the Cloud Service template, as shown in figure 1.6.

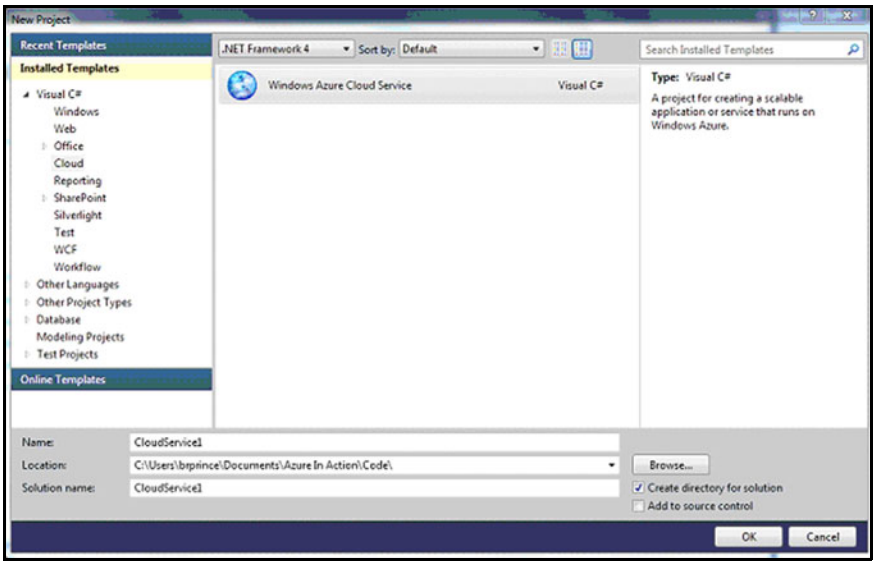


Figure 1.6 The Cloud Service template in the New Project dialog box of Visual Studio 2010

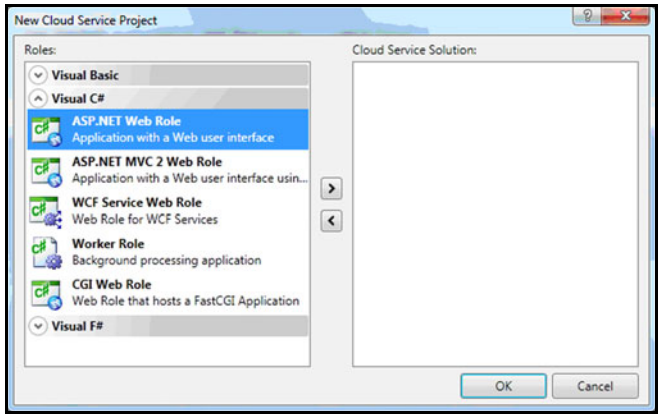


Figure 1.7 New Cloud Service Project dialog box. From here, you can add several Azure projects to your solution.

After you select the Cloud Service template, enter a name for your project and solution, and then click OK. The dialog box shown in figure 1.7 opens, in which you select the type of Windows Azure project that you want to create.

You can create the following types of roles:

- ASP.NET web roles
- ASP.NET MVC 2 web roles
- WCF service web roles
- Worker roles
- CGI-based web roles

You can create your projects in either Visual Basic or C#. In this book, we use C# rather than Visual Basic. This is no disrespect to Visual Basic; we've found over time that although C# developers typically aren't comfortable with Visual Basic, Visual Basic developers are comfortable with both languages (you have to be though, because most samples are in C#).

Select the ASP.NET Web Role project, and then click the right arrow button to add the project to the Cloud Service Solution panel, as shown in figure 1.8.

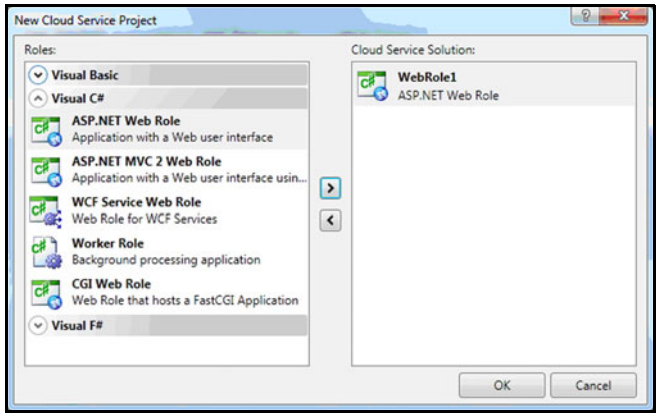
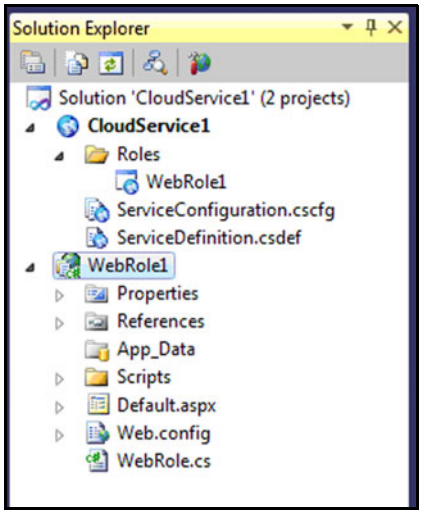


Figure 1.8 Selecting a web role project from the New Cloud Service Project dialog box. Click the default name WebRole1 to change it to something more to your liking.



Now that you've selected your web project, click OK and wait for Visual Studio to generate your solution. After Visual Studio has taken some time to set up your solution and project, it'll have created a new solution for you with two new projects, as shown in figure 1.9.

The first project (CloudService1) contains configuration that's specific to your Windows Azure web role. For now, we won't look at the contents of this project and instead save that for chapter 2. Next, you'll create a simple web page.

Figure 1.9 Solution Explorer for your newly created web role project. The top project (CloudService1) defines your application to Azure. The bottom one (WebRole1) is a regular ASP.NET project with a starter template.

1.2.3 Modifying the web page

The second project (WebRole1) in figure 1.9 is a regular old ASP.NET web application. You can modify the default.aspx file as you would normally. In this case, modify the file to display Hello World, as shown in the following listing.

Listing 1.1 Modifying the default.aspx file to display Hello World

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="WebRole1._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Hello World</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
       Hello World
    </div>
    </form>
</body>
</html>
```

Now that you've created your web page, you can run it in your development fabric.

1.2.4 Running the web page

Before you run your new web role, you must ensure that the cloud service, rather than the ASP.NET project, is your startup project. By default, Visual Studio does this for you

when you create your new project. If the ASP.NET project is the startup project, Visual Studio will run it with the built-in development web server and not the Azure SDK.

Now for the exciting part: you're about to run your first web application in the Windows Azure development fabric. Press F5 as you would for any other Visual Studio application. Visual Studio fires up the development fabric and launches your web page in your browser just like any other web page. Unlike regular ASP.NET web applications, the development fabric hosts your web page rather than the Visual Studio Web Development Server (Cassini). Figure 1.10 shows your web page running in the development fabric.

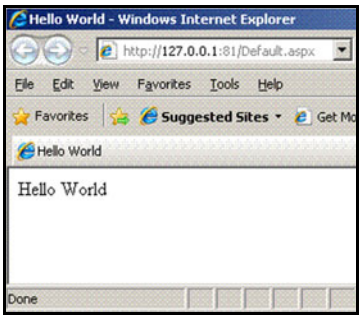


Figure 1.10 ASP.NET 3.5 Hello World running in the development fabric

Congratulations! You've developed your first cloud application. In chapter 2, we'll look in more detail at the development SDK, the development fabric, and how to deploy your service to the live production servers.

Let's return now to our big-picture discussion of Azure.

1.3 Putting all the Azure pieces together

Even Hello World web applications often require multiple instances as the result of the levels of traffic they receive. To understand all that's involved in multiple instances, you first need to understand the Windows Azure logical infrastructure and how it makes it so easy to deploy and run applications in the cloud. As you can see in figure 1.11, the web role is just one piece of the overall infrastructure.

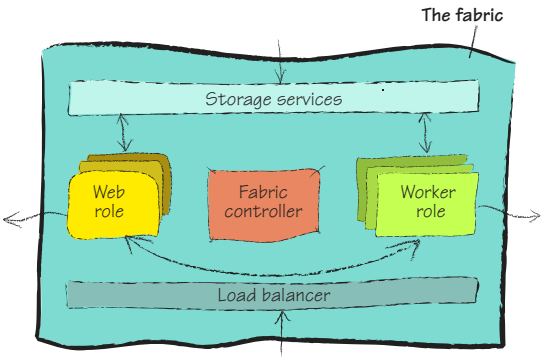


Figure 1.11 The Windows Azure compute infrastructure involves several components. They all work together to run your application.

Over the next couple of sections, we'll look at how the other components—worker roles, the fabric and the Fabric Controller, and the storage services—fit together. First, let's take a closer look at the *load balancer* (the component at the bottom of figure 1.11).

1.3.1 How the load balancer works

Note in figure 1.11 that neither your web roles (web applications) nor your worker roles (background services) have direct incoming traffic from the internet. For both

worker and web roles, all incoming traffic must be forwarded via one or more load balancers. The load balancer provides four important functions, as listed in table 1.1.

Table 1.1 Primary load balancer functions

Function	Purpose
Minimize attack surface area	Improves security
Load distribution	Enables incoming requests to be forwarded to multiple instances
Fault tolerance	Routes traffic to another instance during a fault
Maintenance	Routes traffic to another instance during an upgrade

Not only can a role receive incoming traffic, but roles can also initiate communication with services hosted outside the Windows Azure data centers, with roles inside the data center, and with storage services.

Now that you understand the load balancer's job of distributing requests across multiple instances of web roles, we'll take a brief look at Azure's other type of supported role, the worker role.

1.3.2 Creating worker roles

Worker roles are a lot like web roles and will be covered in depth in chapter 15. The biggest difference is that they lack IIS, which means they can't host a web application, at least not in the traditional sense. Worker roles are best suited for hosting backend processing and a wide variety of web services. These types of servers are often referred to as *application servers* in many IT departments.

At this point, we've explored a few tasks an operating system performs (hosting and running applications). What we haven't explained is how the kernel fits into this analogy of Azure as a cloud operating system. You need something that will manage your applications and all your VMs running in the Windows Azure data center. It's one thing to host an application; it's another to manage what's running and enforce permissions and resource allocation. In a normal operating system, the kernel performs these tasks. In Windows Azure, the kernel is the Fabric Controller (it sits right in the center of figure 1.11).

1.3.3 How the fabric and the Fabric Controller work

Azure contains a massive number of servers, and there isn't any way they can possibly be managed on an individual basis. This is where the Azure operating system concept comes into play. By abstracting away all of those individual servers into a swarm or cloud, you only have to manage the cloud as a whole. This swarm of servers is called the *fabric*, and your applications run in the fabric when you deploy them to the cloud.

The fabric is managed by a software overlord known as the *Fabric Controller*. The Fabric Controller plays the role of the kernel and is aware of every hardware and software

asset in the fabric. It's responsible for installing your web and worker roles onto the physical or virtual servers living in the fabric (this process is similar to how the kernel assigns memory or CPU to an application in a traditional operating system). The Fabric Controller is responsible for maintaining its inventory by monitoring the health of all its assets. If any of the assets are unhealthy, it's responsible for taking steps to resolve the fault, which might include the following:

- Restarting your role
- Restarting a server
- Reprogramming a load balancer to remove the server from the active pool
- Managing upgrades
- Moving instances of your role in fault situations

Windows Azure follows a cloud computing paradigm known as the fabric, which is another way of describing the data center. Like in the movie *The Matrix*, the fabric is everywhere. Every single piece of hardware (server, router, switch, network cable, and so on) and every VM is connected together to form the fabric. Each resource in the fabric is designed and monitored for fault tolerance. The fabric forms an abstract representation of the physical data center, allowing your applications to run in the fabric without knowledge of the underlying infrastructure.

Figure 1.11 shows how the Fabric Controller monitors and interacts with the servers. It's the central traffic cop, managing the servers and the code that's running on those servers. The Fabric Controller performs the job of the kernel (except across multiple servers at a server level rather than at CPU and memory level) in terms of allocating resources and monitoring resources.

One of the jobs that the Fabric Controller doesn't do (but that a kernel does) is the abstraction of the I/O devices. In Azure, this job is performed by storage services, which we'll discuss next (the storage services component sits near the top of figure 1.11).

1.4 Storing data in the cloud with Azure

Suppose you're developing a new podcasting application for Windows 7. For this application, you want to convert MP3 files to WMA. To convert an MP3 file, you first need to read the file from a hard disk (and eventually write the result). Even though there are thousands of different disk drives, you don't need to concern yourself with the implementation of these drives because the operating system provides you with an abstracted view of the disk drive. To save the converted file to the disk, you can write the file to the filesystem; the operating system manages how it's written to the physical device. The same piece of code that you would use to save your podcast will work, regardless of the physical disk drive.

In the same way that Windows 7 abstracts the complexities of the physical hardware of a desktop PC away from your application, Windows Azure abstracts the physical cloud infrastructure away from your applications using configuration and managed APIs.

Applications can't subsist on code alone; they usually need to store and retrieve data to provide any real value. In the next section, we'll discuss how Azure provides you with shared storage, and then we'll take a quick tour of the BLOB storage service, messaging, and the Table storage service. Each of these is covered in detail in their related sections later in this book.

1.4.1 Understanding Azure's shared storage mechanism

If we consider the MP3 example in the context of Windows Azure, rather than abstracting your application away from a single disk, Windows Azure needs to abstract your application away from the physical server (not just the disk). Your application doesn't have to be directly tied to the storage infrastructure of Azure. You're abstracted away from it so that changes in the infrastructure don't impact your code or application. Also, the data needs to be stored in shared space, which isn't tied to a physical server and can be accessed by multiple physical servers. Figure 1.12 shows this logical abstraction.

You can see how storage is logically represented in figure 1.12, but how does this translate into the world of Windows Azure? Your services won't always be continually running on the same physical machine. Your roles (web or worker) could be shut down and moved to another machine at any time to handle faults or upgrades. In the case of web roles, the load balancer could be distributing requests to a pool of web servers, meaning that an incoming request could be performed on any machine.

To run services in such an environment, all instances of your roles (web and worker) need access to a consistent, durable, and scalable storage service. Windows Azure provides scalable storage service, which can be accessed both inside and outside the Microsoft data centers. When you register for Windows Azure, you'll be able to create your own storage accounts with a set of endpoint URIs that you can use to access access the storage services for your account. The storage services are accessed via a set of REST APIs that's secured by an authentication token. We'll take a more detailed look at these APIs in parts 4 and 5 of this book.

Windows Azure storage services are hosted in the fabric in the same way as your own roles are hosted. Windows Azure is a scalable solution; you never need to worry about running out of capacity.

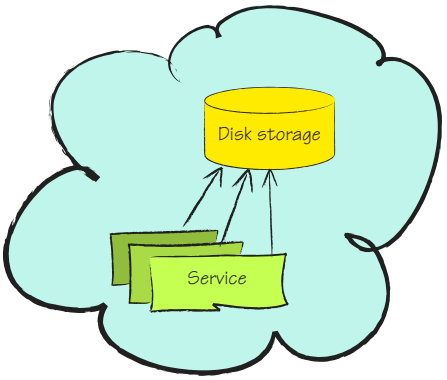


Figure 1.12 Multiple instances of your service (that don't care what physical server they live on) talking to an abstracted logical filesystem, rather than to a physical drive

1.4.2 Storing and accessing BLOB data

Windows Azure provides the ability to store binary files (BLOBs) in a storage area known as *BLOB storage*.

In your storage account, you create a set of containers (similar to folders) that you can store your binary files in. In the initial version of the BLOB storage service, containers can either be restricted to private access (you must use an authentication key to access the files held in this container) or to public access (anyone on the internet can access the file, without using an authentication key).

In figure 1.13, we return to the audio file conversion (MP3 to WMA) scenario. In this example, you're converting a source recording of your podcast (Podcast01.mp3) to Windows Media Audio (Podcast01.wma). The source files are held in BLOB storage in a private container called *Source Files*, and the destination files are held in BLOB storage in a public container called *Converted Files*. Anyone in the world can access the converted files because they're held in a public container, but only you can access the files in the private container because it's secured by your authentication token. Both the private and public containers are held in the storage account called *MyStorage*.

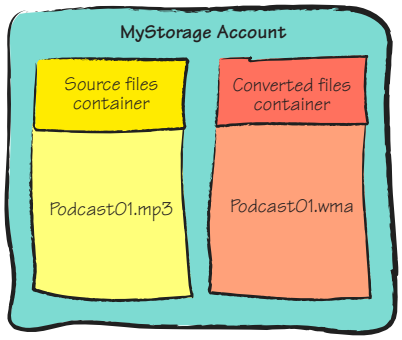


Figure 1.13 Audio files held in BLOB storage

BLOBs can be split up into more manageable chunks known as *blocks* for more efficient uploading of files. This is only the tip of the iceberg in terms of what you can do with BLOB storage in Azure. In part 4, we'll explore BLOB storage and usage in much more detail.

BLOBs play the role of a filesystem in the cloud, but there are other important aspects of the storage subsystem. One of those is the ability to store and forward messages to other services through a message queue.

1.4.3 Messaging via queues

Message queues are the primary mechanism for communicating with worker roles. Typically, a web role or an external service places a message in the queue for processing. Instances of the worker role poll the queue for any new messages and then process the retrieved message. After a message is read from the queue, it's not available to any other instances of the worker role. Queues are considered part of the Azure storage system because the messages are stored in a durable manner while they wait to be picked up in the queue.

In the audio file conversion example, after the source podcast BLOB (Podcast01.mp3) is placed in the Source Files container, a web role or external service places a message (containing the location of the BLOB) in the queue. A worker role retrieves the message and performs the conversion. After the worker role converts the file from MP3 to WMA, it places the converted file (Podcast01.wma) in the Converted Files container.

If you're experiencing information overload at this point, don't worry! In part 6, we'll look at message queues in much greater detail and give you some concrete examples to chew on. Windows Azure also provides you with the ability to store data in a highly scalable, simple Table storage service.

1.4.4 Storing data in tables

The Table storage service provides the ability to store serialized entities in a big table; entities can then be partitioned across multiple servers.

Using tables is a simple storage mechanism that's particularly suitable for session management or user authentication. Tables don't provide a relational database in the cloud, and if you need the power of a database (such as when using server-side joins), then SQL Azure, discussed in chapter 13, is a more appropriate technology.

In chapters 11 and 12, you'll learn how to use Table storage and in what scenarios it can be useful. Let's turn now to the question of why you might want to run your applications in the cloud. You'll want to read the next section, if for no other reason than to convince your boss to let you use it. But you should probably have a better argument prepared than "it's real cool, man" or "this book told me to."

1.5 Why run in the cloud?

So far in this chapter, we've said, "Isn't Azure shiny and cool?" We've also said, "Wow, it's so great I can take my existing IT app and put it in the cloud." But what we haven't asked is, "Why would I want to stick it in the cloud? Why would I want to host my applications with Microsoft rather than host them myself? What advantages do I get using this new platform?" The answers to these questions include the following:

- You can save lots of money.
- You won't need to buy any infrastructure to run your application.
- You don't need to manage the infrastructure to run your application.
- Your application runs on the same infrastructure that Microsoft uses to host its services, not some box under a desk.
- You can scale out your application on demand to use whatever resources it needs to meet its demands.
- You pay only for the resources that you use, when you use them.
- You're provided with a framework that allows you to develop scalable software that runs in the Windows Azure platform so your applications can run at internet scale.
- You can focus on what you're good at: developing software.
- You can watch football and drink milkshakes without being disturbed because someone pulled out the server power cable so they could do the vacuuming.
- You can save lots of money.

In case you think we're repeating ourselves by saying "You can save lots of money" twice, well, it's the key point: you can save a lot. We're often involved in large-scale systems for which the infrastructure costs millions (and most of the time, the servers sit idle). That's not including the cost of running these systems. The equivalent systems in Azure are about 10 percent of the cost.

With that in mind, this section will show you a few of the ways the Windows Azure platform can help you out and save lots of money.

1.5.1 Treating computing power as a utility service

In traditional on-premises or managed-hosting solutions, you either rent or own the infrastructure that your service is hosted on. You're paying for future capacity that you're currently not using. The Windows Azure platform, like other cloud platforms, follows a model of utility computing.

Utility computing treats computing power or storage in the same way you treat a utility service (such as gas or electricity). Your usage of the Windows Azure platform is metered, and you pay only for what you consume.

PAY AS YOU GROW

If you have to pay only for the resources you use, you can launch a scalable service without making a huge investment up front in hardware. In the early days of a new venture, a start-up company survives from investment funding and generates very little income. The less money the company spends, the more chance it has of surviving long enough to generate sufficient income to sustain itself. If the service is successful, then the generated income will pay for the use of the resources.

It's not unusual for technology start-ups to purchase large and expensive hardware solutions for new ventures to cope with predicted future demand. If the service is successful, then it'll require the extra capacity; in the meantime, the start-up is paying for resources that it's not using. Utility computing offers the best of both worlds, giving you the ability to use extra capacity as the service grows without making up-front investments in hardware, and to pay only for the resources that that you use.

SCALE ON DEMAND

Some situations involve large, unpredictable growth; you want to handle the load, but not pay for the unused capacity. This situation might appear in the following scenarios:

- Viral marketing campaigns
- Referrals by a popular website
- Concert ticket sales

Let's say you run a Hawaiian Shirt Shop, and you typically have a predictable pattern of usage. If, for example, Ashton Kutcher (who has 2,000,000 Twitter followers) tweets that he buys his shirts from your website, and he posts a link to your site to all his followers, it's likely that your website will experience a surge in traffic.

Look at the graph in figure 1.14. It shows that your website normally receives around 1,000 hits per day. After Ashton Kutcher tweeted about your website, that increased to

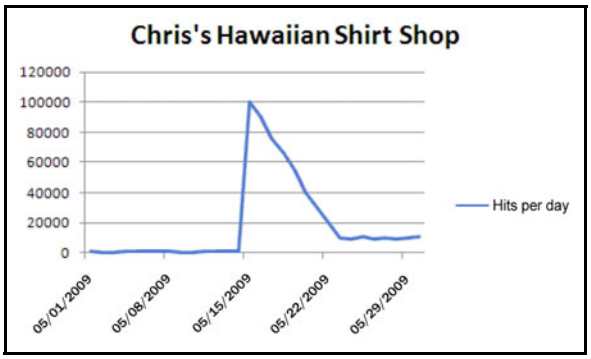


Figure 1.14 Traffic before, during, and after Ashton Kutcher plugged your site

100,000 hits per day. The traffic dropped off after about a week, and then the website had a new baseline of around 10,000 hits per day.

With Windows Azure, you can dynamically scale up to handle the increased traffic load for that week (and get all the extra shirt sales); then, as the traffic decreases, you can scale back down again, paying only for the resources you use.

Scaling up and down in Azure is quite simple, and we'll discuss how to do it in detail in chapter 6. You have several options at your disposal. It's important to remember that Azure doesn't scale your service for you. Because it costs money, you have to tell Azure how many servers you want. Azure gives you tools to do this. You can simply log into the portal and make a small change to a configuration file, which is the easy, manual way. You can also use the Service Management API (covered in chapter 18). This API lets you change the resources you have allocated to your application in an automated way.

It's not only possible to scale up and down for predictable (or unpredictable) bursts of growth; you can also dynamically scale your service based on normal, varied usage patterns.

VARIED USAGE PATTERNS

Returning to the Hawaiian Shirt Shop example: after the Ashton Kutcher hype died down a little, your website leveled off at around 10,000 hits per day. Figure 1.15 shows how this traffic varies over the course of a day. Most of the time there's little traffic on the site, apart from during lunch and in the evening. Evidently, most people don't buy Hawaiian shirts when they're at work.

Because it takes only a few minutes to provision a new web server in Windows Azure, you can dynamically scale your website as your usage patterns dictate. For the Hawaiian Shirt Shop, you might decide to run one instance of the website during the day, but in the evening to run three instances to deal with the increased traffic.

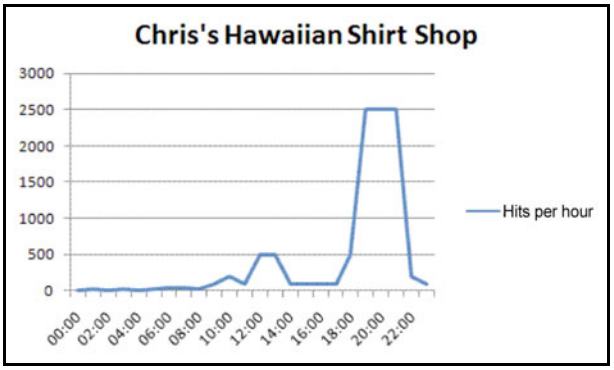


Figure 1.15 Distribution of website traffic over a single day

This sort of scenario is a perfect example of when cloud computing, specifically using Windows Azure, is a perfect fit for your business. If you need to scale beyond a single web server for certain periods of the day when traffic is high, Windows Azure is a cost-effective choice because it allows you to scale back when traffic dies down. Cloud computing solutions are the only offerings that give you this elastic scalability. Other solutions typically require you to over-provision your hardware to cope with the peak periods, but that hardware is underused at off-peak times.

Enough capacity

This example of utility computing can be extended further in regard to available capacity. It's fair to say that most people don't have any idea of the available spare electricity capacity of the network supplying their home, but most of us are confident that if we plug in an extra television, the required electricity will be supplied. The same holds true in Windows Azure. We have no idea how much spare capacity the Microsoft data centers have, but we do know that there's enough. If you require an extra instance of your website, web service, or backend service to be hosted, this will be provisioned for you, within minutes. If the data that your service stores is much larger than you originally anticipated due to the level of growth, more disk space will be allocated to you. You never have to worry about running out of disk space or running out of computing power. You only have to worry about running out of money to pay for the services.

Growth is difficult to model effectively, so knowing there's always enough capacity to grow allows you to concentrate on providing your service rather than worrying about capacity planning, provisioning of new servers, and all their associated tasks.

So far, we've discussed the cost savings you can achieve by scaling your application up and down. Let's now look at how you can save money in maintenance costs.

1.5.2 Simplified data-center management

In this section, we'll look at how the effort involved in operationally maintaining your application is reduced in the Windows Azure environment.

BUILT-IN FAULT TOLERANCE

In Windows Azure, if the physical hardware that your service instance resides on fails, that failed instance is redeployed to another machine. Hardware maintenance resides solely in Microsoft's domain, and you don't have to worry about it.

When you have more than two instances of your application running in Windows Azure, each instance of your web role doesn't live on the same physical server as another instance. This arrangement ensures that if the hardware for one instance dies, the other instance can continue to perform incoming requests. Not only does the second instance live on a different physical server, but it also lives on a different rack (in case the server rack fails). The server rack where the second instance resides is connected to a different network and power grid from the first rack. This level of

fault tolerance ensures that if there's a failure on the physical server, the server rack, the network, or in the electricity supply, your service continues to run and is able to service requests.

When you install your application, Windows Azure decides what servers to place your instances on, with the intention of providing the maximum levels of fault tolerance. Because all data-center assets are tracked and mapped, the placement of applications on each physical server is determined by an algorithm designed to match the fault-tolerance criteria. Even with only two instances of an application, these considerations are pretty complex, but Windows Azure maximizes fault tolerance even when there are hundreds of instances.

Although fault tolerance is maintained within a physical data center, Azure doesn't currently run across the data centers, but runs only within a single data center. You still need to perform offsite backups (if you need them). You can replicate your data to a second data center and run your applications in more than one data center if geo-redundancy is required.

One of the key differences between Windows Azure–hosted applications and regular on-premises solutions or other cloud solutions is that Windows Azure abstracts away everything about the infrastructure, including the underlying operating system, leaving you to focus on your application. Let's see how Azure's ability to maintain the servers your applications run on reduces cost.

SERVER SOFTWARE MAINTENANCE BEGONE!

Whether you're running an entire data center or hosting a website on a dedicated server at a hosting company, maintaining the operating system is usually your responsibility. Maintenance tasks can include managing antivirus protection, installing Windows updates, applying service packs, and providing security. If you're running your own dedicated machine on your own premises rather than it being hosted for you, then you're even responsible for performing backups.

In Windows Azure, because the tasks associated with maintaining the server are the responsibility of Microsoft, you can focus completely on your application. This situation greatly simplifies and reduces the cost of running a service.

A final cost consideration is that if you have a service hosted in Windows Azure, you don't have to worry about the licensing costs for the underlying operating system. You gain all the benefits of running under the latest version of Windows without paying for the costs of maintaining that software. The underlying software is abstracted away from your service, but the base underlying operating system of your service is Windows Server 2008. If you're running multiple servers, the cost of licensing usually runs into thousands of dollars.

Although you don't have to worry about hardware or software maintenance from an operational or cost perspective, you do have to worry about it from a software design perspective.

DESIGNING FOR DISTRIBUTION

Your services won't always be running on the same machine, and they might be failed over to another machine at any time. Failover might be caused by hardware failures, software maintenance cycles, or load distribution. You must design your software so that it can handle these failures. This might mean automatically retrying a failed operation when an exception occurs or reloading any local caches when a service restarts. We'll delve further into these issues in chapter 18.

Let's switch gears now and look at what the Windows Azure *platform* is all about.

1.6 Inside the Windows Azure platform

To reiterate, the major difference between Windows Azure and the Windows Azure platform is the first one is the operating system, and the latter is the broader ecosystem of related services and components. In this section, we'll briefly overview the flagship cloud services provided by the Windows Azure platform (beyond Windows Azure itself): namely, a relational database using SQL Azure, and a set of enterprise services that use the Windows Azure platform AppFabric.

The Windows Azure platform provides many services. In this book, we don't cover every last aspect of every service that's offered across the platform because each component could probably justify its own dedicated book. We'll give you an overview of what's offered, when it's useful, and how you can use the more common scenarios.

Let's get started with the service that you're most likely to use: SQL Azure.

1.6.1 SQL Server capability in the cloud

Although Windows Azure does offer support for storing data in tables, this is a basic storage capability that's suited only for certain core scenarios, which we discuss in chapters 11 and 12.

If you need to create more advanced databases, need to migrate existing SQL databases to Azure, or can't cope with learning another data storage technology, then SQL Azure is the best solution for you. SQL Azure is a relational database (very similar to SQL Server Express Edition) that's hosted within the Windows Azure platform.

A history lesson

When SQL Azure was first announced and made available as a Community Technology Preview (CTP), it was architected differently from the way it currently is. The initial previews of what was known as SQL Server Data Services (SSDS) were of a nonrelational model that was similar to Windows Azure storage services. The feedback given to the product teams made it clear that customers wanted a relational database in the cloud, and SSDS was later retired. Prior to being renamed as SQL Azure, SSDS was renamed SQL Data Services, but there was never a public CTP under this name.

WHAT IS SQL AZURE?

Version 1.0 of SQL Azure, which was released at PDC 2009, provides the core capabilities of SQL Server in the cloud. The first release can be likened to running an instance of SQL Server Express Edition on a shared host, with some changes to security so that you can't mess with other databases on the same server.

Communication with SQL Azure is via the Tabular Data Stream (TDS) protocol, which is the same protocol that's used for the on-premises editions of SQL Server. You can connect SQL Management Studio directly to your database hosted in the cloud, as if it were hosted locally.

> **NOTE** In the first release of SQL Azure, security is limited to SQL Server user accounts. Windows Integrated Security isn't yet supported. Expect some sort of support beyond SQL Security at a later date.

Because you can connect to SQL Azure with a regular connection string, any existing data access layers continue to work normally. Figure 1.16 shows communication between SQL Azure and applications that are hosted both inside and outside Windows Azure.

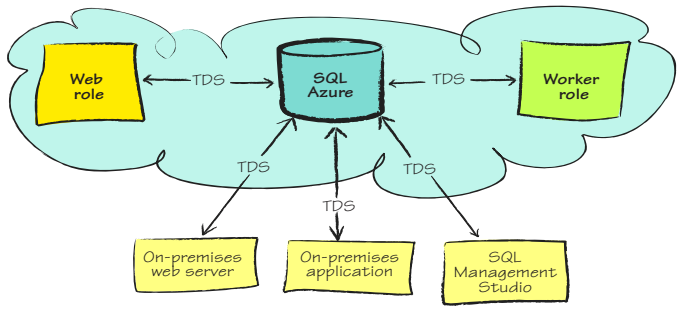


Figure 1.16 On-premises, Azure-hosted applications and SQL Management Studio communicating with SQL Azure via TDS

If your application works today using SQL Server Express Edition and doesn't use some of the more advanced features of SQL Server (see chapter 13), then your application should work in the cloud with little or no modification.

Although on-premises applications can talk to SQL Azure, latency might make this a less attractive option. The closer your application is to the database, the faster it'll go. You can reduce the impact of latency by making your application less chatty.

HOW SCALABLE IS SQL AZURE?

In version 1.0 of SQL Azure, there's no built-in support for data partitioning (the ability to split your data across multiple servers). The initial release is targeted for databases that are sized up to 10 GB; larger databases aren't suitable for SQL Azure in this initial release, but support for larger databases will be available in future service updates. If you need to perform partitioning, you need to implement it in the application layer.

Let's turn now to Azure platform's enterprise services. Known as AppFabric, these services include the Access Control Service (ACS) and the Service Bus.

1.6.2 Enterprise services in the cloud

The Windows Azure platform AppFabric (formerly .NET Services) is a set of services that's more oriented toward enterprise applications and is comprised of the following components:

- AppFabric ACS
- AppFabric Service Bus

AppFabric is a large set of technologies that would require its own book to cover in any depth. In this section, we'll give you an overview of the technologies. In chapter 17, we'll show you how to get started using them and discuss a couple of key scenarios in which to use the technology.

Velocity and Dublin: where are they now?

In addition to the Windows Azure platform AppFabric product, there's also an on-premises product known as Windows Server AppFabric. This is a completely different product set, which currently includes AppFabric Caching (formerly Velocity) and AppFabric Service Workflow and Management (formerly Dublin).

Although these services aren't currently part of the Windows Azure platform (PDC 2009), you can expect them to appear at some point. Depending on when you're reading this book, they could be available right now. Alternatively, you could be reading this in the far future (relative to PDC 2009) and there's no need for this technology because we're all plugged in to the Matrix.

ACCESS CONTROL SERVICE

User authentication security and management is a fairly standard requirement for any service used in an enterprise organization. Enterprises require that their employees, customers, and vendors be able to access all services in the organization with a single login; typically, the authentication process occurs using the Windows login. After you're authenticated, you're typically issued a token, which is automatically passed to other services in the enterprise as you access them. The automatic passing and authentication of this token means you don't have to continually log in each time you access a service.

Services in the enterprise don't implement their own individual authentication and user-management systems but hook directly into the organization's identity-management service (such as Active Directory). This single sign-on process provides many benefits to the company (centralized and simplified user management and security) and is a much more integrated and less frustrating user experience.

Traditionally, identity management and access control have been restricted to the enterprise space. With the advent of Web 2.0 social platforms, such as Live Services and Facebook, this level of integration is now creeping into everyday websites. Web users are now more concerned about data privacy and are reluctant to cheaply give away personal information to third-party websites. They don't want a long list of user names and passwords for various sites and want a much richer social experience on

the web. For example, it's increasingly common for people to want to be able to tell their friends on Facebook about their latest purchase. Between Facebook, Live Services, OpenID (and its differing implementations), and all the various enterprise identity-management systems, access control has now became a complex task.

AppFabric ACS abstracts away the nuances of the various third-party providers by using a simple rules-based authentication service that manages authentication across multiple providers for users with multiple credentials. We'll look more closely at ACS in chapter 17 and show you how to use it in a couple of key scenarios.

APPFABRIC SERVICE BUS

The Service Bus is a cool piece of technology that allows you to message with applications that aren't necessarily running in the Azure data centers. If you have a custom, proprietary service that you need to continue to host, but you want to use Azure for all other aspects of your service offering, then the Service Bus is a good way to integrate with those services.

The Service Bus is effectively an Enterprise Message Bus that's hosted in the cloud. We'll explore in more detail what this means and look at a couple of key scenarios where you could use this technology in chapter 17.

1.7 Summary

In one chapter, you've learned about cloud computing, Windows Azure, and the Windows Azure platform. Although both Windows and Windows Azure are operating systems, providing all the needed functions, they differ greatly in terms of scale. Windows is an operating system for a single machine, whereas Windows Azure is an operating system for a whole fabric of machines, devices, networks, and other related items.

You learned that you can easily scale applications that run in Windows Azure to support the future needs of your application, but you pay only for your current needs.

We also briefly discussed why you might want to use the cloud. The cloud can give you new capabilities, such as dynamic scaling, disposable resources, and the freedom from manning any of it. But the real reason anyone uses the cloud always boils down to money. The rationale is simple: functionality you can't afford to provide in a normal data center is affordable in the cloud.

You developed your first Windows Azure web application and saw that developing in Windows Azure builds on your existing skills. You can now easily write code that runs locally or runs in the cloud—depending on your needs, not your skills or tools.

Like a desktop operating system, the Windows Azure platform consists of a lot of different parts. The platform includes SQL Azure, which provides a traditional relational database that gives you a familiar setting and makes it easier to migrate applications. Azure can also run your applications and manage storage.

We looked at how the Windows Azure platform AppFabric fits into the equation. AppFabric provides both a Service Bus you use to connect your applications together and a simple, standards-based way to secure your services called Access Control Service (ACS).

In chapter 2, we'll continue our discussion of Azure by showing you how to take your first steps with a web role and how to work with code that goes beyond Hello World.

MICROSOFT .NET/CLOUD

Azure IN ACTION

Chris Hay • Brian H. Prince

Microsoft Azure is a cloud service with good scalability, pay-as-you-go service, and a low start-up cost. Based on Windows, it includes an operating system, developer services, and a familiar data model.

Azure in Action is a fast-paced tutorial that introduces cloud development and the Azure platform. The book starts with the logical and physical architecture of an Azure app, and quickly moves to the core storage services—BLOB storage, tables, and queues. Then, it explores designing and scaling frontend and backend services that run in the cloud. Through clear, crisp examples, you'll discover all facets of Azure, including the development fabric, web roles, worker roles, BLOBs, table storage, queues, and more.

This book requires basic C# skills. No prior exposure to cloud development or Azure is needed.

What's Inside

- Data storage and manipulation
- Using message queues
- Deployment and management
- Azure's data model

A Microsoft MVP specializing in high-transaction databases, **Chris Hay** is a popular speaker and founder of the Cambridge, UK, .NET usergroup. **Brian H. Prince** is a Microsoft Architect Evangelist who helps customers adopt the cloud.

For online access to the authors and a free ebook for owners of this book, go to manning.com/AzureinAction

"Easy to read, easy to recommend."
—Eric Nelson, Microsoft UK

"I doubt even the Azure team knows all of this."
—Mark Monster, Rubicon

"An educational ride at an amusement park—great information and lots of humor."
—Michael Wood
Strategic Data Systems

"Highly recommended, like all Manning books."
—James Hatheway
i365, A Seagate Company

"This book will get you in the cloud... and beyond."
—Christian Siegers, Cap Gemini

ISBN 13: 978-1-935182-48-1
ISBN 10: 1-935182-48-X
54499
9 781935 182481

MANNING $44.99 / Can $51.99 [INCLUDING eBOOK]