Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

MEAP Edition
Manning Early Access Program

Copyright 2009 Manning Publications

For more information on this and other Manning titles go to
www.manning.com

Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Contents

Preface

Acknowledgments

About this book

About the authors

Part 1 Welcome to the cloud

1. Getting to know Windows Azure
2. Your first steps with a web role

Part 2 Understanding the Azure service model

3. How Windows Azure works
4. It's time to run with the service
5. Configuring your service

Part 3 Running your site with web roles

6. Scaling web roles
7. Running native, full trust, and other code

Part 4 Working with blob storage

8. The basics of blobs
9. Uploading and downloading blobs
10. When the blob stands alone

Part 5 Working with structured data

11. The Table Service, a whole different entity
12. Working with the Table REST API
13. SQL Azure and relational data
14. Working with different types of data

Part 6 Doing work with messages

15. Processing with worker roles
16. Messaging with the queue
17. Connecting in the cloud with AppFabric
18. Running a healthy service in the cloud

1

Getting to know Windows Azure

This chapter covers:

- Overview of Windows Azure and the Windows Azure Platform
- Building your first Windows Azure Web Role
- Core Cloud Concepts

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

Now that your imagination is running wild, imagine you no longer need to care about the management of hardware infrastructure and you can focus just on the software that you develop.

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 blue sky. Welcome to the Cloud and welcome to Windows Azure. I also suggest that if this is the sort of thing you daydream about you may wish to lie to your non IT friends ;)

Cloud Computing is like a big scary monster

With Cloud Computing in general there are lots of new concepts that we will slowly introduce you to throughout this book eventually giving you the complete picture but for this chapter we will keep things relatively light and simple and try not to scare you off. As you get more comfortable with this new paradigm we will introduce more of the complexities as the book progresses.

If Cloud Computing is a big scary monster, then think of this book as monster therapy. Let's say you were walking down the street and you saw a big scary monster, you would probably jump out of your skin. However, if you saw lots of cute little monsters that got progressively scarier you'd probably be okay when you saw that same big scary monster. Throughout this book we will introduce these new concepts as cute little monsters allowing you to eventually see that same monster but keep your skin intact. Obviously we would also like you to avoid being eaten too ☺

Over the course of this chapter we'll look at the following concepts

- What is the Windows Azure Platform?
- What is Windows Azure?
- Developing your first Web Role
- Saving money by running in the Cloud
- Overview of the other Windows Azure Platform services

To get the ball rolling, we'll start by looking at the big picture by looking at the entire platform.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

1.1 What is the Windows Azure Platform?

As you might have gathered from the opening section, the Windows Azure Platform really describes Microsoft's complete Cloud offering. Every service that Microsoft considers as part of the Cloud will come under this banner.

> **The whole Cloud thing passed me by**
>
> There isn't really anything magical about the Cloud. It really just means a bunch of servers that can host and run your applications, or an offering of services that can be consumed (e.g. a Web Service ☺).
>
> The main difference between a Cloud offering and a non-Cloud offering is that the infrastructure is abstracted away (i.e. you no longer care about the physical hardware that hosting your service)
>
> Finally most Public Cloud solutions are offered as a metered service, i.e. you pay for the resources that you use (compute time, disk space, bandwidth etc) as and when you use them.

As of today (PDC09 release) the Windows Azure Platform really splits into 3 parts:

- The Operating System for the Cloud (Windows Azure)
- A Relational Database in the Cloud (SQL Azure)
- Enterprise Services in the Cloud (Windows Azure platform AppFabric)

As I said earlier, you can expect the above list to expand over time, in-fact I wouldn't be surprised if we saw Microsoft Flight Simulator in the Cloud ;)

As cool as AppFabric and SQL Azure are, for just now we are going to stay focused on the Windows Azure part of the Windows Azure Platform and ignore all the other platformy stuff until the end of the chapter.

> **The names confuse me**
>
> So this is where the naming gets a little confusing straight off. Unfortunately when most folks refer to Windows Azure it's not clear if they are referring to Windows Azure or the Windows Azure Platform.
>
> It's not unlike the ESPN naming convention. The ESPN network has multiple channels (ESPN, ESPN2, ESPN News etc) 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. Therefore if you ask someone what game is on ESPN tonight, it's not clear if are you meaning all the channels on ESPN (including ESPN News and ESPN2) or are you meaning just the channel named ESPN (not including ESPN2 etc).
>
> To keep things consistent in this book whenever we talk about the platform as a whole we will refer to "The Windows Azure Platform" or "The Platform" but if we are talking about the core Windows Azure product then we will use the term "Windows Azure".

1.1.1 It must be an operating system it has Windows in the title

Windows Azure is what Microsoft calls their core *operating system* for the Cloud. Ok so now you know what Windows Azure is we can skip on right? I didn't think so! So let's strip away all the hype, what does it actually give us?

Well, Windows Azure simply provides us the ability to run our applications in a highly scalable manner on Microsoft servers in Microsoft's Data Centers in a manageable way. This basically means we can either host our web applications (e.g. Websites that sell Hawaiian shirts) or some backend processing services (e.g. an MP3 to WMA file converter) in Microsoft's Data Centers. If we need more computing power to run our application (i.e. more instances of our website or most instances of our backend service), we can just spin up more instances of our application that will be spread across many servers. By increasing the number of instances of our application, we will ultimately be able to process more data or handle more incoming traffic.

Hmmmm, How exactly is that an operating system? Well, to answer that we have to define what it really means to be a Cloud operating system.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The first thing we have to consider is that when Microsoft refer to Windows Azure as an operating system for the Cloud they don't literally mean an operating system as you may know it (Windows 7, Windows Vista, Leopard, Snow Leopard etc). What they really mean is that Windows Azure performs the same sort of job that a traditional Operating System may perform.

So what does an operating system do? Well it really has 4 tasks in life:

- Host and run our applications
- Abstract the complexities of hardware from our applications
- Provide an interface between users and applications
- Funds the development division of a very large blue company that produces Visual Studio

Figure 1.1 shows how these tasks are achieved in traditional operating system on a normal PC.

Figure 1.1 - A typical representation of an operating system how it interacts with applications and resources. Notice that applications do not directly interact with CPU, Memory and Devices.

Figure 1.1 shows a set of application running with an Operating System. As you can see from the diagram since the applications have no direct access to the hardware meaning that all interactions with the hardware must come through the Kernel.

The Kernel

The Kernel is the low level operating system component that pretty much performs all the tasks we will describe i.e. Process Management, Memory Management and Device Management.

Actually this analogy of Windows Azure being an Operating System looks like it could work out. Over the next few sections we will continue with this analogy and see how Windows Azure fairs as an Operating System, by doing this we should get a pretty good overview of how Windows Azure works and what services it provides.

1.2 Hosting and running applications

So number 1 on the list is hosting and running applications. I kind of think this is the most important task of an operating system. Without applications we are really just moving a mouse around with no purpose ☺

Let's get started by looking at the types of applications that can be run in traditional operating systems as well as Windows Azure.

1.2.1 Types of Applications

In traditional operating system such as Windows 7, we could consider most things as an application e.g.

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

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 1.1 shows some applications running in the context of normal PC operating system. Instead of hosting client applications (games, Word, Excel etc), the types of applications that you host in Windows Azure would be server applications such as:

- Web Applications (e.g. a Hawaiian Shirt Shop Website)
- Background Computational Applications (e.g. an MP3 Converter)

Figure 1.2 shows some server applications running in a traditional operating system

Figure 1.2 Windows Azure type applications running in an normal OS

As you can see from Figure 1.1 and Figure 1.2 there is no real difference between Excel and a Hawaiian Shirt Shop Website they are simply applications as far as a traditional operating system is concerned.

1.2.2 Running Applications across Thousands of Servers

In traditional operating systems, the operating system is responsible for allocating CPU time and memory space so that our application can run (as seen in Figure 1.1 and 1.2).

Not only is the operating system responsible for allocating these resources, it's responsible for managing these resources. For example if an application fails, then it's the Operating Systems job to clean up the applications resource usage and restart the application if necessary.

This level of abstraction is perfect for an Operating System that manages a single server but this just isn't scalable for a Cloud Operating System. With Windows Azure our application doesn't just run on a single server, it can potentially be running 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. We need to abstract to a higher level.

In the case of Windows Azure that higher level of abstraction is Virtual Machines. Figure 1.3 shows how our applications could be distributed amongst the Virtual Machines in a Windows Azure Data Center.

Figure 1.3 – Application split across Virtual Machines in the Windows Azure Data Center

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

As we can see from Figure 1.3 our Cloud Operating System is no longer for responsible for assigning our applications resources by CPU and memory but is instead responsible for allocating resource by virtual machines. In this diagram our Hawaiian Shirt Shop Website is allocated 2 virtual machines hosted in 2 different physical servers, whereas the Azure in Action Website is only allocated a single Virtual Machine (shirt shops make more money so they get more roles).

VIRTUAL MACHINES

Finally Figure 1.4 shows what the virtual machine hosting a web application roughly looks like.

Figure 1.4 – A logical representation of the virtual machine that hosts our web application.

In figure 1.4 we can see the Virtual Machine hosts the Web Application within IIS7.0. As we could see from Figure 1.3 each physical server is split up into multiple virtual machines. Every instance of your service (web or worker role) is installed onto its own virtual machine which is a base installation of Windows 2008 server (with some extra Azure bits) as shown in Figure 1.4. The use of virtual machines is how Windows Azure achieves separation of services across physical servers; this means that another instance of a role hosted on the same physical hardware cannot interfere with your role instance.

Although your application runs on a virtual machine, the virtual machine is abstracted away from us and we only ever have a view of the role instance, never of the virtual machine. You should also notice that a single instance of your web application is assigned to a single virtual machine (also no other applications will be assigned to that virtual machine). This is ensures that every instance of your web application is isolated from other applications running on the same physical server. The final thing to note that the Virtual Machine image also runs an agent process; we'll explain what this does a little later on in the book.

Web Role

A role is simply another name for your application. It actually refers to the base VM image that hosts your application. A web role is simply VM that hosts your application within IIS.

To be honest I'm now itching for a bit of code. Let's look at how we can build a simple ASP.NET Website that we can run in one of those Windows Azure Virtual Machines.

1.2.3 Building your first Windows Azure Web Application

Although we are going to build an ASP.NET Website in this example, the good news is that pretty much any website that can be currently hosted in IIS on Windows Server 2008 can be hosted in Windows Azure.

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

- ASP.NET 3.5 Web Applications
- ASP.NET MVC 1.0 Web Applications

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

- Web Services (WCF & ASMX)
- Any FastCGI based Website
    - PHP
    - PythonEtc

Although Windows Azure supports the ability to host different types of website, for now we will just create a simple hello world web application using ASP.NET 3.5SP1. Later on in the book we will look at how you can create PHP websites, WCF Web Services and ASP.NET MVC Websites.

SETTING UP YOUR ENVIRONMENT

To get started developing an ASP.NET 3.5SP1 Website you will need to download the Windows Azure Software Development Kit (SDK). The SDK contains a whole bunch of things that will make your life easier when developing for Windows Azure including:

- 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 new bright new world (with a slight blueish tinge)

In the next chapter we will take a deeper look at some of the items in the SDK. For now we will just install it. If you are an experienced ASP.NET Developer you should be able to just install the SDK by clicking the next button a few times.

Before installing the SDK, check your version of Windows and Visual Studio.

SUPPORTED OPERATING SYSTEMS

Before you attempt to install the SDK, it's worth making sure that you have a suitable version of Windows. At present the only versions of Windows supported are:

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

Note: You cannot install the SDK on Windows XP or Windows 2003

Please note that Windows XP is not supported for Windows Azure. Before you jump up and down about Windows XP, there isn't some conspiracy against it (although there should be). The reason that XP isn't supported is that Windows Azure Web Roles are heavily built upon IIS 7. Windows XP (and Windows 2003) use earlier versions of IIS that won't work with Windows Azure. Later on in the book, we will look exactly what features of IIS7.0 is dependent upon and why it cannot work in earlier versions of IIS.

SUPPORTED VERSIONS OF VISUAL STUDIO

One final point before you go an install the Windows Azure SDK. To develop Windows Azure applications in Visual Studio, you will need either Visual Studio 2008 or Visual Studio 2010. If you are still running Visual 2005, you now have the excuse you needed to upgrade. If you can't afford Visual Studio or your company won't upgrade you (stoopid company), then you can either use the Visual Studio 2010 Beta (until it time bombs), or you can use Visual Studio Web Developer 2008 Express Edition (which is of course, free).

Now that you have installed the Windows Azure SDK, let's fire Visual Studio up and create our first Azure web application.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

FIRING UP VISUAL STUDIO

In order to launch your Windows Azure Application in the Development Fabric from Visual Studio you will need Administrator privileges. It's worth getting into the habit (for Azure development) of right clicking on your Visual Studio icon, and selecting (run as administrator).

NEWING UP OUR PROJECT

Our first step is to 'new up the project'. To do this, we need to open Visual Studio, and select File > New > Project. You should select the "Cloud Service" project type, which will then give you the option to select the "Cloud Service" template as shown in Figure 1.5.

Figure 1.5 - The Cloud Service template in the New Project dialog of Visual Studio

Once you have selected the "Cloud Service" template, you should set the name for your project and solution then click OK. You will then be presented with the dialog as shown in Figure 1.5 that will allow you to select the type of Windows Azure project that you wish to create.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 1.6 - New Cloud Service Project dialog.

As you can see from figure 1.6, you are able to create:

- ASP.NET Web Roles
- Worker Roles
- CGI Based Web Roles

It is also worth noting that you can create your projects in either Visual Basic or C#. In this book we will be using C# rather Visual Basic. This is no disrespect to Visual Basic but we have found over time that C# developers are typically not comfortable with Visual Basic but Visual Basic developers are comfortable with both languages (I guess you have to be though since most samples are in C#).

You should now select the "ASP.NET Web Role" project and click on the right arrow button so the project is displayed in the right hand listbox as shown in figure 1.7.

Figure 1.7 - Selected a web role project from the new cloud service project dialog

Now that you have selected your web project, you can click OK and wait for Visual Studio to generate your solution.

Once Visual Studio has came out of project creation meltdown, it will have create you a new solution with 2 new projects as shown in figure 1.8

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 1.8 - Solution Explorer for our newly created web role project

The 2 projects as shown in Figure 1.8 are:

- Our Windows Azure Cloud Service Project
- A plain old ASP.NET ASP.NET 3.5 Web Application

The first project (MyCloudService) is our Cloud Service Project. This project contains a bunch of configuration that is specific to our Windows Azure Web Role. For now we won't look at the contents of this project and instead save that for later a chapter.

Modifying our Web Page

As you may have already gathered the second project (WebRole1) is just a regular old ASP.NET Web Application. Therefore you can modify the Default.aspx file as you would normally. In this case you should modify the file to display Hello World as shown below:

```
<%@ 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>
```

Running the Web Page

Now that we have created our web page we just want to fire it up in our development fabric. Before you run your new Web Role, you must ensure that the Cloud Service is your startup project rather than the ASP.NET project. By default Visual Studio will do this for you on creation of your new project.

Now for the exciting bit, you are now about to run your first web application in the Windows Azure Development Fabric. Drumroll……. Just hit F5 like any other Visual Studio application.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Once you have hit F5, Visual Studio will fire up the development fabric and launch your web page in your browser just like any other web page. Unlike regular ASP.NET Web Applications, the Development Fabric will host your web page rather than the Visual Studio Web Development Server (Cassini). Figure 1.9 shows our web page running in the development fabric.

Figure 1.9 - ASP.NET 3.5 Hello World running in the development fabric

Congratulations, you have now developed your first cloud application. In the next chapter we will look in more detail at the development SDK, the development fabric and how to deploy your service to the live production servers.

1.2.4 How does it all fit together?

Later on in this chapter, we will take a look at how we can take our web role and scale it (Hello World web applications often need multiple instances due to the levels of traffic they receive) but before we do that we need to have an understanding of the overall Windows Azure logical infrastructure as shown in Figure 1.10

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 1.10 - Windows Azure

As you can see from Figure 1.10 the Web Role is just one part of the overall infrastructure. Over the next couple of sections we will look at how some of the other components fit into the overall service namely:

- Worker Roles
- Fabric Controller
- Storage Services

Before we do that it's worth pointing out the Load Balancer.

LOAD BALANCER

In figure 1.10 you will notice that neither our web roles (web applications) nor our worker roles (background services) have direct incoming traffic from the internet. In the case of both worker and web roles all incoming traffic must be forwarded on via the load balancers. The load balancers provide 4 very important functions:

- Reduces Surface Attack Area (improves security)
- Load Distribution (incoming requests can be forwarded to multiple instances)
- Fault Tolerance (traffic is routed to another instance during a fault)
- Maintenance (during an upgraded load balancer can route traffic to another instance)

Not only can a role receive incoming traffic, it should be noted that the web role can initiate communication with services hosted outside of the Windows Azure data centers, roles inside of the data center as well as storage services.

Now that you understand how Windows Azure can distribute requests across multiple instances of web roles using the load balancer, we can expand upon that knowledge and looking at some of the demands of scaling web applications slightly later on.

Before we look at scaling however, we will take a brief look at the other type of supported role, i.e. the Worker Role.

1.2.5 Worker Roles

TODO

So we have explored the first task of an operating system (hosting and running applications). I think we agree that Windows Azure ticks that box.

What we haven't explained is where the Kernel fits into this picture of the Operating System? We need something that will manage our applications and all our Virtual Machines running in the Windows Azure Data Center. In Windows Azure the Kernel is really the Fabric Controller.

1.3 The Fabric Controller

The Fabric

Windows Azure follows a model of Cloud Computing known as "The Fabric". The Fabric is really another way of describing the data center. Like "The Matrix", "The Fabric" is everywhere. Every single piece of hardware (server, router, switch, network cable etc) and every virtual machine is connected together to form the Fabric. Each resource is the Fabric is designed and monitored for fault tolerance. The Fabric forms an abstract representation of the physical data center allowing applications to be run within the Fabric without knowledge of the underlying infrastructure.

The Fabric is managed by a software overlord known as the Fabric Controller. The Fabric Controller is aware of every hardware and software asset within the Fabric. It is responsible for the installation of your web and worker roles onto the physical or virtual servers living within the Fabric (this is not unlike the Kernel assigning

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

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 is responsible for taking steps to resolve the fault. This may include:

- 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.

Figure 1.11 shows how the fabric controller monitors and interacts with the servers.

Figure 1.11 – The Fabric Controller: TODO new diagram and caption

As you can see from Figure 1.11, the Fabric Controller is performing the job of the kernel (except across multiple servers at a server level rather than CPU + Memory Level) in respect to the following items:

- Allocation of Resources
- Monitoring of Resources

One of the jobs that the Fabric Controller doesn't do that a Kernel will do, is the abstraction of the IO Devices. This job is performed by Storage Services.

1.4 Storing Data in the Cloud

Before we look at how Windows Azure abstracts storage in the cloud from the physical hardware. It's worth going back to our analogy of Windows Azure being an Operating System for the Cloud.

1.4.1 Abstracting the physicality's in a Regular OS?

Let's say you are developing a new timesheet application for Windows 7, and for this application you may wish to convert MP3 files to WMA. In order to be able to convert our MP3 file, we 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 do not need to concern yourself about the implementation of these drives as the operating system provides you an abstracted view of the disk drive. If you need to save your converted file to the disk, you can just write the file to the file system and the operating system will manage how this is written to the physical device. The same piece of code that you would use to save your timesheet 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 from your application, Windows Azure abstracts the physical cloud infrastructure from your applications via

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

configuration and managed API's. So for the MP3 example in Windows Azure rather than abstracting a single disk from our application, Windows Azure needs to abstract our application from the physical server (not just the disk). Also the data would need to be stored in abstracted storage array which is no longer tied to a physical server and can be accessed by multiple physical servers. Figure 1.11 shows this logical abstraction.

Figure 1.11 – Multiple instances of our service (that don't care what physical server they live on) talking to an abstracted logical file system, rather a physical drive

So we can see how storage is logically represented in Figure 1.1, but how does this translate into the world of Windows Azure?

As we have discovered our services will not always be continually running on the same physical machine. Our roles (web or worker) could be shutdown and moved to another machine at anytime in order 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.

In order to run services in such an environment, all instances of our roles (web and worker) need access to a consistent durable scalable storage service. What this means is that all instances of our roles must have access to a shared storage mechanism.

Windows Azure provides its scalable storage service that can be accessed both inside and outside of the Microsoft data centers. On sign-up for Windows Azure you will be assigned your own storage account with a set of endpoint URI's that you can use to access the storage services for your account. The storage services are accessed via a set of REST API's secured by an authentication token. We will take a more detailed look at these APIs later on in the book.

Windows Azure Storage Services are hosted in The Fabric in the same way as our own roles are hosted. This means that it is a very scalable solution and we never need to worry about running out of capacity. We will now take a brief look at the services offered by Windows Azure Storage Services.

1.4.2 Binary File Store (Blobs)

Windows Azure provides the ability to storage binary files (blobs) within a storage area known as Blob Storage.

Within your account you create a set of containers (similar to folders) that you can store your binary files within. In Version 1 of the Blob Storage Service, containers can either be restricted to private access (i.e. you must use an authentication key to access the files held in this container), or to public access (i.e. anyone on the internet can access the file, without using an authentication key).

In figure 1.12 we return to our audio file conversion scenario. In this example we wish to convert a source recording of our podcast (Podcast01.mp3) to Windows Media Audio (Podcast01.wma). The source files will be held in Blob Storage in a private container called "Source Files", and the destination files will be held in Blob Storage in a public container called "Converted Files". Anyone in the world can access the converted files as it is held in a public container but only you can access the files in the private container as it secured by our

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

authentication token. Both the private and public containers are held within the storage account known as "MyStorage" account.

Figure 1.12 - Audio Files held in blob storage.

Blobs can be split up into more manageable chunks known as Blocks for more efficient uploading of files.

In chapter 8 we will explore in much more detail the usage of Blob Storage in Windows Azure.

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 will place a message in the queue for processing. Instances of the worker role would poll the queue for any new messages and then process the retrieved message. Once a message is read from the queue, it is no longer available to any other instances of the worker role.

In the case of our audio file conversion example, once the source podcast blob (Podcast01.mp3) is placed in the source container, a message could be placed in the queue (containing the location of the blob) ready for a worker role to perform the conversion. Once the worker role has converted the file from MP3 to WMA it can place the converted file (Podcast01.wma) in the Converted Files container.

In Chapter 16 we will look at message queues in much greater detail and expand upon these scenarios.

1.4.3 Storing data in tables

Windows Azure has the ability to store data in a highly scalable simple table storage service. This service provides the ability to store serialized entities in a big table; entities can then be partitioned across multiple servers.

This is a very simple storage mechanism that is particularly suitable for scenarios such as Session Management, User Authentication or Shopping Carts. This is not a relational database in the cloud, and if you need the power of a database (such as using server side joins) then SQL Azure is a more appropriate technology.

In Chapter 11 we will look at how to use Table Storage and in what scenarios it can be useful.

1.5 Saving Money by running in the Cloud

Ok, so far in this chapter, we've basically 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 might I want to host my applications with Microsoft rather than host them myself? What advantages do I get using this new platform? I guess some of the answers to these questions could be:

- Save lots of money

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

- I don't need to buy any infrastructure to run my application
- I don't need to manage the infrastructure to run my application
- My application runs on the same infrastructure that Microsoft uses to host their services, not some box under a desk
- I can scale my application out on-demand to use whatever resources it needs to meet the demands of my application
- I only pay for the resources that I use when I use it
- I am provided a framework that allows me to develop scalable software that runs within the Windows Azure Platform so my applications can run at Internet Scale
- I can focus on what I am good at, i.e. developing software
- I can watch Football and drink milkshakes without being disturbed because some idiot pulled out the server power cable so they could do the vacuuming
- Save lots of money

Just in case you think I'm repeating myself by saying **Save lots of money** twice, well it's the key point, **you save bucket loads**. So without giving away any commercials, I'm often involved in large scale systems and the infrastructure of these systems costs millions (and most of the time the servers sit idle), the cost of equivalent systems in Azure are about 10% of the cost. I'm not including the cost of running these system, you do the math (and this book seems pretty cheap now in comparison too) ☺.

With that in mind let's take a look at some of these scenarios in a little more detail and look at how the Windows Azure Platform can help us out.

1.5.1 Treating computing power as a utility service

In traditional on-premise or managed hosting solutions you either rent or own the infrastructure that your service is hosted on. This means that you are paying for future capacity that you are currently not using.

The Windows Azure Platform like other Cloud Platforms follows a model of Utility Computing. The premise of Utility Computing is the ability to offer computing power or storage in the same way as you would treat a utility service (such as gas or electricity). Your usage of the Windows Azure Platform is metered and you only pay for what you consume.

PAY AS YOU GROW

If you only have to pay for the resources that you use, it means that you can launch a scalable service without making a huge investment upfront in hardware. This is a very compelling model for start-ups. In the early days of a new venture a start-up company is surviving from investment funding and is generating 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 as successful then the generated income will pay for the usage of the resources.

It is not unusual for technology start-ups to purchase very large and expensive hardware solutions for new ventures to cope with predicted future demand. If the service is successful then it will require the extra capacity but in the meantime the start-up is paying for resource that it is not utilizing. Utility computing offers the best of both worlds, allowing you the ability to utilize extra capacity as the service grows without making upfront investments in hardware and only paying for the resources that are actually used.

SCALE ON DEMAND

There are some situations where we have large un-sustained growth and we would like 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

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

- etc

Let's say for example we run a Hawaiian Shirt Shop and we normally 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 my website and he posts a link to my site to all his followers, it is likely that we can expect a surge in traffic.

If we look at the graph in Figure 1.13 it shows our website would normally receive around 1000 hits per day. After Ashton Kutcher tweeted about our website, it increased to 100,000 hits per day, the traffic dropped off after around a week and then the website had a new baseline of around 10,000 hits per day.

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

Figure 1.13 - Traffic before, during and after Ashton Kutcher plugged your site

TODO: Show how to scale

VARIED USAGE PATTERNS

It is not only possible to be able to scale up and down for predictable or unpredictable bursts of growth, it is possible to dynamically scale your service based on normal varied usage patterns.

If we return to the Hawaiian Shirt Shop example, after the Ashton Kutcher hype has died down a little, our website leveled off at around 10,000 hits per day. In Figure 1.14, we look at how this traffic is broken down over the course of a day. We can see that most of the time we have very little traffic on the site apart from at lunch time, and in the evening. It seems most people don't buy Hawaiian shirts when they are at work.

Since it only takes a few minutes to provision a new web server in Windows Azure, it is possible for us to dynamically scale our website as our usage patterns dictate. So in the case of our Hawaiian Shirt Shop, we may decide to run one instance of our Website during the day, but in the evening to run 3 instances to deal with the increased traffic.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 1.14 - Distribution of website traffic over a single day

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

Enough Capacity

This analogy of Utility Computing can be extended further in regards to available capacity It is fair to say that most people do not have an idea of the available spare electricity capacity of the network supplying our home but most of us would be happy that if we plug in extra television to our home that the required electricity will be supplied. This analogy holds true in Windows Azure, we have no idea how much spare capacity the Microsoft data centers have but we do know there is enough. If we require an extra instance of our Website / Web Service / Backend Service to be hosted then this will be provisioned for us (within minutes). If the data that our service stores is much larger than we originally anticipated due to the level of growth, then more disk space will be allocated to us. We never have to worry about running out of disk space, or running out of computing power, we only have to worry about running out of money to pay for the services.

As you may have already gathered growth is very difficult to model effectively, so knowing there is always enough capacity to grow allows us to concentrate on providing our service rather than worrying about capacity planning, provisioning of new servers and all their associated tasks.

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

1.5.2 Simplified Data Center Management

In this section we will look at the effort involved in operationally maintaining your application is reduced within a Windows Azure environment.

BUILT-IN FAULT TOLERANCE

In Windows Azure, if the physical hardware that our service instance resides on fails, then our failed instance will be spun-up on another machine. The maintenance of hardware purely resides in Microsoft's domain and we do not have to worry about it.

Although fault tolerance is Microsoft's responsibility, it is worth considering how the distribution of the instances throughout the data center is performed with fault tolerance in mind.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Where we have more than two instances of our application spun up in Windows Azure, each instance of our web role will not live on the same physical server as another instance. This ensures that if the hardware for the instance dies then the other instance would be able to continue to perform incoming requests. Not only will it live on a different physical server, it will live on a different rack (in case the server rack fails). The server rack where the second instance resides will even be connected to a different network and power grid from the first rack. This level of fault tolerance ensures that if there is a failure on the physical server, the server rack, the network or in the electricity supply, our service will continue to run and be able to service requests.

On the installation of our application, Windows Azure decides what servers to place our instances on with the intention of providing the maximum levels of fault tolerance. Since all data center assets are tracked and mapped, the placement of applications on each physical server are algorithmically chosen to match the fault tolerance criteria. These considerations are pretty complex even with 2 instances of an application but Windows Azure will maximize fault tolerance even if there are hundreds of instances.

Todo: DIAGRAM

Big Scary Monsters Squishing Data Centers

Although fault tolerance is maintained within a physical data center, if a big scary monster came along and squished the data center then it's not quite clear what would happen (hopefully it will be by the time this prints). Your data is safe, it will be replicated to another data center, just not sure when the other data center comes up.

The good news the last time I was in San Antonio, I didn't notice any big scary monsters however I can't be sure about Chicago. Although it's not clear how cross data center fault tolerance is currently provided and how the switchover occurs. IF this is something you really need you could manually configure it (if monsters worry you that much). **[TODO: Diagram of a rack with 2 instances on different network etc]**

SERVER SOFTWARE MAINTENANCE BEGONE!

One of the key differences between Windows Azure hosted applications and regular on premise solutions or other cloud solutions, is that Windows Azure abstracts away everything about the infrastructure including the underlying operating system leaving us to focus on our application.

Whether you are running an entire data center, or hosting a website on a dedicated server at a hosting company, maintenance of the operating system is usually your responsibility. Maintenance tasks can include Anti-Virus, Windows Updates, applying service packs and locking down of security. If you are running your own dedicated machine on your own premises rather than it being hosted for you, then you would even be responsible for performing backups.

In Windows Azure, since you are purely focused on your application, the tasks associated with maintaining the server are the abstracted away from you and are purely the responsibility of Microsoft. This level of abstraction 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 do not have to worrying 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. Although the underlying software is abstracted away from your service, the base underlying operating system of your service is Windows Server 2008. If you are running multiple servers the costs of licensing usually runs into thousands of dollars.

DESIGNING FOR DISTRIBUTION

Although we do not have to worry about hardware or software maintenance from an operational or cost perspective, we do have to worry about it from a software design perspective. Our services will not always be running on the same machine and they could be failed over to another machine at any time, this could be due to hardware failures, software maintenance cycles or load distribution. This means that we must design our software to be able to handle these failures. This may mean automatically retrying a failed operation on exception, or reloading any local caches on restart of a service. We will delve further into these issues later on in the book.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

1.6 Windows Azure Platform (previously Azure Services Platform)

As we said earlier, the major difference between what is Windows Azure and what is Windows Azure platform is necessity. In this section, we'll give a brief overview of those services offered in the Windows Azure Platform (beyond Windows Azure itself), which are:

- Relational Database in the Cloud (SQL Azure)
- Enterprise Services in the Cloud (Windows Azure platform AppFabric)

In this book we do not cover every last aspect of every service that is offered across the Windows Azure Platform as each component could probably justify its own dedicated book (I'm really not pitching my next book proposal here ☺). We will however try and give you and understanding of what is offered, when it is useful, and how the more common scenarios can be utilized.

Let's get started by giving a brief overview of the service that you are most likely to use, SQL Azure.

1.6.1 Relational Databases in the Cloud (SQL Azure)

Although Windows Azure does offer support for storing data in tables, this is a very basic storage capability that is only suited for certain core scenarios.

If you need to create more advanced databases, migrate existing SQL databases to Azure or really just can't cope with learning another data storage technology then SQL Azure is really the best solution for you. Quite simply SQL Azure is a relational database (very similar to SQL Server Express Edition) that is 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 very differently. The initial previews of what was known as SQL Server Data Services (SSDS), was a non-relational model that was very similar to Windows Azure Storage Services. The feedback given to the product teams made it very clear that customers wanted a relational database in the Cloud and SSDS was later retired (actually obliterated is probably a closer analogy). Prior to being renamed to SQL Azure, SQL Data Services was renamed to 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 PDC09 provides the core capabilities of SQL Server in the cloud. The first release could be likened to running an instance of SQL Server Express edition on a shared host, with some changes to security so you can't mess with other databases on the same server.

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

WINDOWS INTEGRATED SECURITY IS NOT SUPPORTED AT PRESENT

In the first release of SQL Azure, security is restricted to SQL Server User Accounts and Windows Integrated Security is not supported. I suspect that some sort of support beyond SQL Security will come at a later date

Since you can connect to SQL Azure with a regular connection string, this means any existing data access layers will continue to work as normal. Figure 1.15 shows communication between SQL Azure and applications that are hosted both inside Windows Azure and outside of Windows Azure.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 1.15 – *On premise, Azure hosted applications and SQL Management Studio all communicating with SQL Azure via TDS*

TODO: Need to a new version of this diagram

If your application works today against SQL Express edition and does not make use of some of the more advanced features of SQL Server (see the SQL Azure chapter) then your application should work in the Cloud with little or no modification.

ON PREMISE LATENCY

Although you can have on-premise applications talking to SQL Azure, latency may make this less attractive. The closer your application is to the database, the faster it will go. You can reduce the effects by making your application less chatty.

HOW SCALABLE IS SQL AZURE?

In version 1.0 of SQL Azure there is no built in support for data partitioning (i.e. the ability to split your data across multiple servers). The initial release is targeted for databases that are sized up to around 10Gb, larger database are not suitable for SQL Azure in this initial release but support will available in future releases. If you need to perform partitioning this would need to be implemented in the application layer.

WHAT WE WILL COVER LATER ON?

Since SQL Azure is a cut down version of SQL Server, we will not focus on standard SQL Server functionality but instead concentrate on the Azure specifics. We will look at the current architecture and how this is likely to change in the future, and dive deeper into what design patterns and scenarios are suitable today for building Cloud Applications.

1.6.2 Enterprise Services in the Cloud (Windows Azure platform AppFabric)

Windows Azure platform AppFabric (formerly .NET Services) is a set of services that are more orientated towards enterprise applications and is comprised of the following components:

- AppFabric Access Control

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

- AppFabric Service Bus

AppFabric is a very large set of technologies that really requires its own book to cover it in depth. In this book we will give an overview of the technologies, how to get started using it, and give a couple of key useful scenarios where you can use the technology.

> **Windows AppFabric Server**
>
> As well as the Windows Azure platform AppFabric product, there is also an on-premise product known as Windows Server AppFabric. This is a completely different product set which currently includes AppFabric Caching (formerly Velocity) & AppFabric Service Workflow and Management (formerly Dublin).
>
> Although these services are not currently present in the Windows Azure Platform (PDC09), you can expect them to appear at some point. Who knows depending on when you are reading this book, they could be here right now. Alternatively, you could be reading this in the far future (relative to PDC09) and there is no need for this technology as we are all plugged into the Matrix.

ACCESS CONTROL

User Authentication security and management is a fairly standard requirement for any service used within an enterprise organization. Enterprises require the ability for their employees, customers and vendors to be able to access all services in the organization with a single login; typically the authentication process occurs via the Windows login. Once you are authenticated, you will typically be issued a token which will be automatically passed to other services in the Enterprise as they are accessed. The automatic passing and authentication of this token means you do not have to continually login, each time you access a service.

Services in the Enterprise therefore do not implement their own individual authentication / user management systems but hook directly into the organizations 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. However, 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 usernames and passwords for various sites and want a much richer social experience on the web. For example it is 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 Access Control Service abstracts away the nuances of the various third party providers utilizing a simple rules based authentication service that can manage authentication across multiple providers for users with multiple credentials. Later on in this book we will look into the Access Control service in a little more detail, and use it in a couple of key scenarios.

SERVICE BUS

The service bus is a pretty cool piece of technology allows you to message with applications that are not necessarily running within the Azure data centers. Therefore if you have a custom proprietary service that you need to continue to host but want to use Azure for all other aspects of your service offering, then the Service Bus is a good way of integrating with those services.

The Service Bus is effectively an Enterprise Message Bus that is hosted in the Cloud. We will explore in more detail what this means and a couple of key scenarios where you could use this technology, in chapter 17.

1.7 Summary

Hopefully we have started to tame that big scary monster and started to make it look like a cute little teddy bear. In one chapter we have learned about Cloud Computing, Windows Azure and the Windows Azure Platform.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In this chapter we have seen how applications that run within Windows Azure can be easily scaled to support the future needs of your application whilst only paying for your current needs. We've also seen how developing Windows Azure builds upon your existing skills by developing your first Windows Azure Web Application.

In future chapters we will start to peel away the layers of this new platform and look at how we can use the platform effectively.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

2

Your first steps with a Web role

In this chapter:

- Build a basic web site
- Sign up for Azure
- Deploy your first cloud application

With the first chapter out of the way, you should have a feeling for the lay of the land, installed the SDK, and ran a hello world application locally. Let's dive right into building a web site to run on Azure. In this chapter, we'll cover all the steps involved:

- Start a new Visual Studio project
- Build the web site's XHTML and code
- Run and debug the site locally
- Deploy the site to the Azure staging environment
- Move the site to the Azure production environment

Don't let these steps daunt you. If you have ever developed a web site with ASP.NET, then you are already ahead of the game. You will have far fewer steps to deploy an application with Azure than you would with a traditional server.

In a large enterprise project I was working on, we spent 15% of the work hours just on planning out the development, quality assurance, and production environments. This was mostly due to defining hardware requirements, capital expenditure approval, and vendor management. We could have shipped much sooner had we been able to focus on the application, and not the underlying infrastructure and platform. Many organizations take three to six months just to deploy a server!

We will be focusing on the basic process of deploying a simple web site using the web role in Azure, which is highlighted in figure 2.1.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 2.1 - We will be focusing on the basic use of the Web Role in Azure for part of this chapter.

2.1 What is included in the SDK

Once installed, the SDK will create a folder structure on your computer filled with tools, goodies, and documentation. The default path is `C:\Program Files\Windows Azure SDK\v1.0`.

Figure 2.1 – The directories the Azure SDK installs for you. You should spelunk these as there are lots of handy tools.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

You should explore these folders briefly, to be familiar with what is available. There are several tools included in the SDK that are very powerful. Some tools can help with automated build environments, PowerShell, and automated management of your live Azure applications.

EXPLORING THE SDK FOLDERS

Some of the more important folders are:

- Bin

  This folder contains the required assemblies and tools for working with Azure packages and running the development fabric.

  The SDK provides tools for when you aren't using Visual Studio, or are automating some of your processes. In all cases, the VS tools will run these for you as need. Table 3.x lists the tools included in the `Bin` directory.

Table 2.1 The most important tools included in the SDK

VS Tool	Purpose
CSmonitor	This will load the local dev fabric. You should see its icon appear in your systray.
CSpack	This tool is used to manually create deployment packages to upload to the Azure portal when you are ready to run your application in the cloud.
CSrun	This lets you deploy a package generated by CSpack to run on the local dev fabric.
DevelopmentStorage	This will start the local development storage service, allowing you to simulate cloud storage locally.
DevTableGen	When pointed at your assemblies, will use reflection to create the required tables in the local development storage service.
DFUI	This script will start the management UI for the local development fabric.
DSinit	This tool is used to run the initialization needed for the development storage service on a new SQL instance. This is usually only run once.

- Doc

  Contains the help documentation for the SDK. This is a document that you will want to refer to on a regular basis while you are working with Azure.

- Inc

  Holds a header file for use if you are working with C++.

PROJECT TEMPLATES

The SDK install also created a new project template group in Visual Studio, called `Cloud Service`. This group will include several templates for you to start working with. You can use them when adding a new project to you solution. Never fear though, you can also add an existing project. This is really handy when you are migrating an application to the cloud.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 2.2 – You have several options when it comes to adding an Azure project to your solution. These project templates are provided with the SDK.

- **Blank Cloud Service** - This isn't really a template, but if you click OK without adding any projects, you will have a solution with just an Azure project without any supporting application projects.
- **ASP.NET Web Role** - This template creates an ASP.NET project, pre-configured with an accompanying Azure project.
- **ASP.NET MVC2 Web Role** – This creates a project similar to the above template, but prewired to support the MVC2 framework.
- **WCF Service Web Role** – Planning on hosting a WCF service instead of a normal web application? Then this is the project for you. It will be setup like a normal WCF project, with sample files for your first service.
- **Worker Role** - This template will create a class library project, pre-configured with a related Azure project. You should use this project if you are building a background processing service.
- **CGI Web Role** - This project template will create the required files needed to host a FastCGI project, which will be covered in chapter six.

RUNNING THE CLOUD LOCALLY

We talked in chapter one about the internal web server that Visual Studio has included for the past few versions, called Cassini. This is included to make it easy for a web developer to develop locally, without having to install full blown server software on their desktop. Cassini was also designed to be very lightweight, and to only respond to local web requests. This was done for performance and security reasons. The Azure SDK includes a similar tool, as a way to help the developer develop cloud web sites and services locally. The SDK installs both a Development Fabric Service, and a Development Storage Service. Both of these will be started automatically by Visual Studio when you run an Azure project. While they are separate processes, you can find them in the

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

system tray area of your task bar as a single icon. Each actually has several different processes related to it, to handle load balancing, node management, and a few others. These are provided so you can run and debug your cloud services locally, before deploying them to the cloud. They effectively work together, with a local instance of SQL Express, to simulate the real Azure storage and runtime environment. Because of this simulation, there are some limitations, which we will discuss later in this chapter.

The Development Fabric Service icon looks like a set of blue gears, as you can see in figure 2.3. Using this tool, you can monitor and manage the role instances of the local cloud environment. The Development Storage Service icon looks like an Xbox. It is used to configure the storage containers configured for your cloud services.

Figure 2.3 - You will find the local cloud service icons in your status bar. The blue Windows flag tells you the Development Fabric and the Development Storage services are running. You can right click to show the UI to manage the services.

If you aren't using Visual Studio, or if the services don't start for you, you can start them from the Windows Start menu, or from the Azure Command Prompt.

Once started, both services will continue to run until you stop them with the taskbar UI. I recommend you leave them running while you are developing, and then shutting them down when you are done for the day.

It is possible to use an existing local instance of SQL Server instead of SQL Express to support the Storage Service. The SDK includes a tool call DSinit.exe, which can be used to initialize the storage databases. To configure the storage service to use your local SQL instance, you need to use the **/sqlInstance** parameter with the name of the SQL instance you want to have initialized.

```
DSinit.exe /sqlInstance MSSQL
```

SDK SAMPLES

The SDK also includes a wide variety of samples. These samples can be used to get a good idea of how to approach a variety of supported scenarios. One of the best ways to learn Azure is to walk through the code in the samples, and understand how they work. The samples are provided in a zip file, so you will need to extract them to a folder to work with the code.

2.2 How the SDK and cloud environments differ

While the SDK tools try to provide a complete simulation of the production cloud environment, there are some limitations developers should be aware of. Most of these limitations are because the local environment is running on one machine, with limited resources. Other times the limits are because the local storage service is based on a local SQL Express instance. We believe, that over time, the gaps between the local environment, and the cloud environment will shrink. Table 2.2 summarizes the differences between each environment.

Table 2.2 Azure environment differences

Feature	Local Environment	Cloud Environment
Storage environment access	The local storage environment uses a special account key for access.	This will be different from your cloud key.
	The local storage does not support the use of HTTPS.	Cloud storage supports both HTTP and HTTPS.
Storage performance	The performance of the local storage is not representative of cloud storage. It is only meant for a few	The performance of the cloud based storage should be expected to be much faster.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

	local connections, nothing more.	
URI management	Because of the lack of the Azure DNS system locally, storage URI's are different in the local environment.	URI's are based on Azure DNS system.
Storage management	The local storage subsystem will not provide extended error information.	Cloud storage provides extended error information.
Blob storage	Blobs in the local store are limited to 2GB in size.	BLOBs in the cloud can be as big as 50GB.
Table authentication	Local tables only use Shared Key Lite authentication.	Uses Shared Key and Shared Key Lite authentication.

One way to minimize the impact of the local storage limitations is to shift to a blended model during your development. Early on in your project, you can configure your application to use local storage. As you start to bump up against the limits, simply adjust the configuration of the application to use the cloud based storage. In this blended mode, you would continue to run your application locally. Eventually, you will deploy your application to the cloud staging environment, with the application configured to use the cloud storage.

To get a feel for how the SDK and tools work, we are going to create a simple website.

2.3 Taking Hello World to the next level

We are going to build a second web application now, and it will be a bit more complicated than the Hello World sample in chapter 1. This new website isn't going to have a lot of functionality, just a bit of XHTML and CSS, with some simple code. We aren't going to be using any databases, or advanced topics. Our goal is to make sure that the SDK and tools are installed properly, and show you the general workflow of working with a web application in Azure.

This one-page website lists the different shirts available at Chris' Hawaiian Shirt Shop. In addition to walking through the basic workflow, we'll also show you that, in many scenarios, running a website in Azure is just like running a web site on your own server, and that it is just as easy, if not easier, to deploy it.

2.3.1 Newing up the project

Our first step is to 'new up the project', as we like to say. We need to open Visual Studio, and select File > New > Project. Visual Studio will show us a list of templates available, and we want to select the cloud service group. Select 'Windows Azure Cloud Service' in the menu. Give the project a name, such as "HawaiianShirtShop", and click OK. A menu will popup that will let you pick the project types you want to add to your Azure solution. Since we are building a simple web site, without any back end processing, choose the **ASP.NET Web Role** template. Once you add the web role to the project, you can rename it before hitting OK. You should see a screen like the one shown in figure 2.3.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 2.4 - Creating a new Azure Application project is easy. Just add the different roles you will need in your solution.

Visual Studio will start building a solution file for you with a few projects. The first project will be named with what you entered on the new project screen. This will be the Azure project, and it doesn't contain a lot. It merely contains the model and configuration Azure will need to have in order to run your web site. You can think of this as the metadata for your whole Azure application. The project also contains links to other projects in your solution, and what role type those projects play in your application. These links are stored in the `Roles` folder. The solution and project structure you should see is shown in figure 2.5.

Figure 2.5 - The projects Visual Studio Creates for you. The bolded project is the Azure project, which holds the cloud

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

configuration data for your application. It also shows the HSS Web App project linked as a web role.

Visual Studio also created an ASP.NET web application project for you, and linked it to the Azure project as a web role. Remember that you can name the sub projects when you select them from the role selector (figure 2.4) by clicking on the pencil icon.

This ASP.NET project is usual in almost every way. Since Visual Studio knew it was going to be part of an Azure application, it created an assembly reference for you to three Azure related assemblies. The three assemblies are part of the `Microsoft.WindowsAzure` namespace. The `Diagnostics` namespace covers logging needs, the `ServiceRuntime` provides methods for interacting with the Azure fabric, and the `StorageClient` assembly makes it easy to work with the Azure storage services.

You could hit F5 right now, and the project would compile and run locally. Visual Studio would start up the local development fabric and storage services, and then package up and deploy your application. Of course without any code or markup, you won't see too much, except for an empty web browser. This is the cleanest and most performant your app will ever be.

2.3.2 Laying down some markup with XHTML and CSS

Now that we have the empty shell in place, we need to put in the content for the site. This page is going to simply announce the name of our business, and list a few of the shirts we have for sale.

When we are completed, we will have a simple web page that looks like figure 2.6.

Figure 2.6 - This is what our simple web page will look like.

Open up the default.aspx file, and paste in the following markup. With Azure, you don't need any special tags or changes to your website, in most cases, to do what you have always done. This makes it really easy for a web developer to learn how to use Azure, and quickly move to the cloud.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Listing 2.1 The simple ASP.NET XHTML markup used in our web page.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="HSS_Web_App._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 id="Head1" runat="server">
   <title>Chris' Hawaiian Shirt Shop</title>
   <link rel="stylesheet" href="Main.css" type="text/css" />
</head>
<body>
   <form id="form1" runat="server">
   <div id="header">
      <h1>
         Chris' Hawaiian Shirt Shop</h1>
   </div>
   <div id="mainContent">
      <h3>
         Many Styles Available!</h3>
      <p>
         We have many styles available, to suit any special need or occasion. Please check out our amazing classic 50's
styles. Call now to place your order or to check on what we have in stock.</p>
      <asp:Repeater ID="rptProductList" runat="server">                                                    #1
         <ItemTemplate>
            <div class="item">
               <h3>
                  <%# Eval("productName") %>                                                               #2
               </h3>
               <div>
                  <b>
                     $<%# Eval("unitPrice")%>                                                              #2
                     each.</b>
                  <p><%# Eval("description") %></p></div>                                                  #2
            </div>
         </ItemTemplate>
      </asp:Repeater>
   </div>
   </form>
</body>
</html>
#1 A normal repeater is used to list our the shirts in our absolutely fabulous collection.
#2 We are using eval statements to render out the shirt data.
```

The numbers #1 and #2 should be code references.

As you can see in the markup above, this is pretty boring and run of the mill ASP.NET web page. The `asp:repeater #1` will be bound to a collection of shirt data, so that it will render out a list of products available. The repeater will render out our business data with the `eval` statements `#2`.

2.3.3 Binding our data in the code behind

The next step in our simple sample is to get the product data from a stub that will provide sample data, and then bind it to the repeater control in the markup. The stub will be called `ProductInfo,` and act as a simple factory. A lot of developers would call this class the `ProductService`, but I think the service word is overloaded as it is and just makes things confusing. You should feel free to roll the way you want, it's your code. We will also have a data transfer object (a simple class that has no methods, only properties) to hold our product data called `Product`. We will use a generic list collection to hold the multiple product classes as we bind them to the repeater control. You can place these classes in a new C# class file in your project.

```
public class Product
{
     public Product()
     {
     }
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
    public int sku { get; set; }                                1
    public string productName { get; set; }                     1
    public decimal unitPrice { get; set; }                      1
    public string description { get; set; }                     1
}
```

#1 Simple accesors for our product properties.

The #1 should be a cueball for the code.

Since we are trying to keep things simple, we are using simple property accessors #1 around the business data. Next, we'll add the Page_Load code that will gather the product data, and bind it to the fields on the form.

```
ProductInfo productInfoSource = new ProductInfo();
IList<Product> allProducts = productInfoSource.GetAllProducts();

rptProductList.DataSource = allProducts;                                1
rptProductList.DataBind();                                              1
```

#1 Connecting the data source and calling the databind method to tie everything together.

The #1 in the next two paragraphs should be code reference cueballs.

When the page is loaded, it will create a `ProductInfo` object, and use it to get a collection of `Product` objects, which are then bound #1 to the `ProductList` repeater on the web form.

Once you have the code entered properly, and run the application, Visual Studio will start up the local Azure services (you should see their icon in your task base like in figure 2.3), and launch the web site in your browser. When you are running Azure applications locally, you must run Visual Studio as an administrator. If you forget to do this Visual Studio will kindly remind you when you hit F5.

If you look really closely at the code, our web project doesn't really have or do anything a normal web site wouldn't have or do. The only thing difference is the Azure project, which tells either the local services, or the Azure cloud, how to run your application.

The Azure Fabric Controller knows how to deploy and manage your application based on the settings included in the Azure project in your solution. There are several parts to this configuration that we want to cover now.

2.3.4 Just another place to run your code

We are going to spend a lot of time in this book discussing the unique capabilities the Azure platform has. You have probably heard a lot about cloud computing, and might be a little confused or intimidated. Don't worry! Until you fully grok the powers of the cloud, the easiest way to wrap your head around it is to think of it as just another place to run your code. If you boil it down to this simple concept, then the most powerful aspect is the simplicity of the deployment and management of applications.

Each cloud project defines a service model for the application. This model defines how it should be deployed and run in the cloud. With this model, there isn't a 30 page deployment document with obscure and arcane instructions for deployment and upgrading. You can now simply deploy you application with a few clicks and few moments. Azure knows what to do with your code because of the configuration and service model in the Azure project in your solution.

2.3.5 Configuring the Azure service model

The most important aspects of the Azure project are the links, which tell Visual Studio which projects in the solution are parts of the application, and what role they play. These will show up in the portal as elements that can be configured and scaled serperately. A class library in your solution should be referenced with a project assembly reference like normal, and not linked to from the Azure project file. Besides these links, the Azure project also includes two configuration files:

- ServiceDefinition.csdef
- ServiceConfiguration.cscfg

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The ServiceDefinition file defines which services are part of the application, and if the services have any endpoints or connections that need to be provisioned. This would include the public port a web site might need to receive traffic from the internet. It basically concerns itself with the configuration of the infrastructure.

The ServiceConfiguration file determines the configuration of the services, and how they should be provisioned by Azure. This would include how many instances of each role should be deployed, and other operational characteristics. We will investigate each of these files in more depth, later in chapter three.

2.3.6 Running the web site in the local dev fabric

As we have mentioned, when you run an Azure application locally, Visual Studio will start up the dev fabric and storage services, and then launch the application. Part of this process includes removing old temporary files, and packaging the project to be deployed to the local services.

Once the services are loaded, you can run their UI's to see what is happening in the local environment. Each time you run your application, a new deployment is created, each with a successive number. It is possible to have other deployments from your applications show up in the UI besides just the one you are currently running.

The UI will show the different deployments listed down the left panel. Each deployment will display the details of that service, as well as a list of the different roles and their instances that are running. In figure 2.7, you can see we have one role, a web role called HSS Web App. It was the ASP.NET project in our solution that was linked to from the Azure project. That role has one instance running. In the UI the instances are numbered, starting with zero, since we are all geeks. To the right you can see the log history for that instance.

Figure 2.7 - The UI of the development fabric. This shows that there is one web role running, and the log history for that instance.

With this UI you can pause or stop the different services, which is helpful during debugging. You can also change the logging levels of each service, to better refine what the log is capturing, which again, is helpful during debugging.

Now that we know how to install and use the Azure SDK, and run our applications locally using the dev fabric and storage services, we are going to turn our attention to actually deploying our little web site to the cloud and then become rich selling Hawaiian shirts.

2.4 Deploying with the Azure portal

The Azure Portal is one of the three major components to managing your Azure applications. These components are shown in figure 2.8.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 2.8 - The management aspects of Azure include the portal, system logs, and the analytical tools.

The other two components are your system logs (often referred to as diagnostics), and the analytical and billing tools.

Before you can login to the portal to manage your applications you will need to create an Azure account.

2.4.1 Signing up for Azure

You will need to sign up, provide a Live ID, and billing details. You will need to visit www.Azure.com to sign up. Go to the account section and click 'Get Your Account'. You can provide two Live IDs, one for business purposes such as billing and contract details. The second account you can provide will be limited to the technical aspects of the system. This cleanly separates who manages the business aspects from the technical aspects. Having two accounts keeps the manager from accidentally shutting down the applications or reconfiguring them. The cloud doesn't remove the risk of human mistakes; we just try to automate them so they cause trouble faster and more efficiently.

Even if you have access to free time in the cloud through MSDN, or some other channel, you will still need to provide a credit card in case you over your free allocation.

2.4.2 What is the Azure Portal, anyway?

The Azure Portal is your central management tool for all of the aspects of running you application in the Azure environment. As shown in figure 2.9, across the top, you will be able to view the technical details of services, or the account details. In each section, your various services (web sites, services, storage, etc.) will be listed down the left.

- Summary tab—lists your projects, and give you access to managing them. From here you can upload new versions, flip staging and production, and manage their log files.
- Account tab— helps you manage your affinity groups, certificates, and any other broad configuration that affects all services under your account.
- Help and Resources tab—brings together all of the help options available to you, including technical support and public forums.

Now that you have your account provisioned, we need to create a project on Azure to host your account.

2.4.3 Setting up your service online

You will need to log into your Azure portal with the Live ID you registered with. On the home page, click the 'New Service' link in the top right. This will start the process of creating a new service, which will be where you push your code to. The term service is used as a general term on the Azure portal to represent some usage of the system. This might be a storage account, or one of several web sites you have deployed.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Azure will show you the different types of services available to you on your Azure account. You should see icons for Azure Storage, and Hosted Services. You might also see icons for the other services that are part of the greater Azure Services Platform, such as the .NET Services or SQL Azure.

Figure 2.9 - You will need to choose the 'Hosted Services' project type when creating your online project. Based on my account limits, I am allowed to create two more storage services. Create a hosted service when you have an application to run in the cloud.

The portal will display the type of projects that are available to you, as in figure 2.9. Click on the Hosted Service icon to start the new service wizard. Oh, how I love wizards.

The first step is to provide some basic properties for your project. You will need to provide a project name, and a description. These are for your own use, but by being careful in how you name and describe your project, you can make it easier to manage your projects over the long haul. After you provide the requested information, click next to proceed.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 2.10 - This is the second step in creating an Azure hosted services project. You can provide the name that will be in the URL for your site, as well as the affinity group for geo-distribution.

The more important part of this second screen, as seen in figure 2.10, is the service name. Remember that pretty much any code running in Azure is called a service, even if it is a simple website like our sample. This service name will carry over into almost everything you do with the service you are deploying, so choose wisely. While selecting an embarrassing childhood nickname is tempting (such as Doogie Howser or Brainy Smurf), this really should be a name to support how you choose to manage your Azure environment.

The name you provide will be the first part of the URL for your service. For example, if my service name was DayOldSushiOnline, the Azure DNS system will refer to your site as DayOldSushiOnline.cloudapp.net. Since this isn't the name you embroidered on thousands of t-shirts, preferring instead the URL www.DayOldSushiOnline.com, you can use your own DNS tricks to refer people to the proper URL. It is a simple CNAME record in your DNS server that can direct people to your service with any domain name you want to use.

The service name has to be unique across all of Azure. There is a button provided to help you find out if the name you want is available or not. I am going to enter AzureInAction, and see if it is available. Because I am doing this, it will not be available to you, go ahead and check.

The lower half of this form has to do with what Azure calls Affinity Groups. Affinity Groups are used to make it easy to have services that are related to each other deployed to the same regional datacenter for performance, or to different datacenters for disaster recovery or geo-distribution purposes.

Click 'Create' to finish the process. At this point, a project has been provisioned, and is ready for you to deploy an application to. The portal will then display the project page for the project you just created.

Each project has two zones in it, staging and production. The typical process of deploying your application will involve uploading it to the staging environment, testing it, and *flipping* it to the production environment. This flipping is called a VIP swap. VIP stands for virtual IP.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

When flipping your application to a new environment, the bits aren't actually moved. The current running instances are just labeled 'production', and the other instances are labeled 'staging' by adjusting the assignment of DNS and IP addresses in the Azure datacenter. This makes it fast to do a cut over, and reduces the chance of deployment issues. This also makes it easy to flip back to the old setup if the new configuration isn't working out as planned, not that that has ever happened to us mind you.

2.4.4 Putting our logging boots on

Logging is very important in Azure. Like most production environments, you cannot attach your debugger to the code running in the cloud. If a problem happens, you need to have good logging in place to help in diagnosing that problem.

You should learn to instrument your code with logging commands, so that you have the right information at your fingertips when you really need it. Every time you fix a bug, you should consider adding some information to the log as well.

Azure runs a diagnostics agent on your virtual server while it is running. This agent gathers logs and diagnostics information. It can be configured in code, and managed remotely with the Azure management APIs. The default project templates provided by the SDK include code that will start the diagnostics agent when your role instance starts.

The diagnostic agent can gather logs from several standard sources and from any source you can think up by using the custom error log extensions. The standard sources include Azure logs, Windows logs, IIS logs, Windows event log, and performance counters. The data will be transferred to an Azure storage account of your choosing. The diagnostic data is collected, and then transferred either on demand, or according to a schedule you set. Once it is there you can do anything with it that you want. Some data sources will be copied to an Azure table, while others will be copied to an Azure blog account. The type of destination storage depends on the type of the source.

We will cover logging in greater depth in chapter XXX. For now what we need to do is setup the storage environment we will need for our basic application. Open up your application in Visual Studio, and make one more run locally, to make sure that it is running fine. There is one default setting we need to change. The default configuration has the diagnostics monitor using development storage. Before we upload our code, we need to reconfigure it to use the cloud for storage since cloud apps can't reach your development storage on your local desktop.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 2.11 – We need to create a storage account to store our logs and diagnostics data in. Creating a storage account is very similar to creating a hosted service.

Return to the portal and create a new service. But instead of hosted services, choose storage account for the type of service that you create. Figure 2.11 shows that the process is very similar to creating a hosted service. As before, you will need to provide a service name, and a description. Once the storage account is created you will be given an account key to securely access your online data. We will go much deeper into storage accounts in chapter seven, but for now just roll with it.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 2.12 – You will be able to access your access keys once the storage account is setup. These keys give anyone access to your data, so keep them safe. They are redacted here for just that reason.

Open the ServiceConfiguration.cscfg file and replace the following line to include your storage account settings. This change will reconfigure the diagnostics system to store log data into your storage account.

Old:

```
<Setting name="DiagnosticsConnectionString" value="UseDevelopmentStorage=true" />
```

New:

```
<Setting name="DiagnosticsConnectionString"
value="DefaultEndpointsProtocol=https;AccountName=YourAccountName;AccountKey=YourAccountKey”
/>
```

The `accountname` parameter needs to be lowercase and URL friendly. We will cover why in the chapters on Azure storage. Your application will still run locally, you have merely pointed the diagnostics system to store it's logs in your cloud based storage system. Once you have made this change you are ready to package up your fabulous shirt web site and move it to the cloud.

2.4.4 Packing and moving on up to the East Side

Now we have an Azure account, a hosted services project created, and some bits for an application sitting on our local disk. We now need to take our application, and its configuration, and create a CS Package for deployment.

WHAT DOES CS STAND FOR?

You will see the acronym CS all throughout the SDK and the tools for Azure. VB.NET'ers don't need to get upset; CS doesn't stand for C#, but for Cloud Service.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

We now need to create a package for the bits. One way to use this is the CSpack.exe utility in the SDK. While this is great for scripting, it can be tedious for those that live inside Visual Studio all day. The quicker method is to right click on the Azure project in your solution (named AiA CH02 - Hawaiian Shirt Shopin our sample), and choose 'Publish.'

The publish command will call CSpack for you, creating the package, and opening a file explorer window to where the package files are placed, and then opening a browser window to your Azure project portal page. You will need to sign into your portal, and then select the project you want to deploy to on the left. Click the 'deploy' button under the staging jello cube, and use the screen to upload the package, and the configuration file from the file explorer that was opened for you. After you select these files, I suggest you use the text box on the bottom of the form to give the deployment a label for history purposes. I usually use either a build number or version number. It can be anything you want. In this example I am going to enter 'version 1, build 14, ch2 sample preview, sp1 spring refresh'.

Once you click the 'deploy' button, the files will be uploaded, and you will be redirected back to the service page on the Azure portal. You can choose to deploy directly to the production environment, but we recommend you always deploy to the staging environment first. When the files are uploaded, the state of the service will be set to 'stopped.'

Deployment is as easy as uploading a few files, and clicking the 'Run' button, at least for you. For Azure, the fabric controller kicks in, identifies some unused CPU cores, deploys a virtual server image, copies over your bits, wires up a VLAN for only your instances, reconfigures the load balancers, and then updates the service directories. While all this is happening, which can take from seconds to minutes depending on what you are deploying, you will see the status of your environment flip to 'Initializing.' Once all of the work is done, the status will change to 'Started.' When things are deployed, the jello cube will turn a nice shade of blue.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 2.13 - Our sample application, has been deployed to the staging environment, and fully initialized. It is assigned an easy to remember GUID based host name so you can test your deployment.

While in the staging environment, as seen in figure 2.13, your application will have a temporary URL, with a GUID as the host name of the service. The friendly name you picked while creating the Azure project won't be used until you migrate your application to production. Before you do that, test your application in staging by using the temporary URL, which is provided lower down on the screen.

Once your application is running the 'Run' button will turn into a 'Suspend' button. If you use this button you will take your application offline. Azure is stopping the IIS application in the background when you do this, and even though it is suspended, you are still using a VM, so the Azure hours for your billing will continue to accrue.

While on the portal screen you can also edit the project's affinity settings, as well as the project description.

2.4.5 Moving to production

Now that our application has been tested in staging, we can move it to the production environment. Remember what was we discussed above; you aren't actually moving the application, or the settings. When you flip the button, Azure just reconfigures the fabric to route all new traffic from the outside world to this particular instance. The old production becomes the new staging environment. A great benefit to this is an easy roll back to the old version if things go wrong. To move your application to production, just click on the circular arrows in the center of the screen. When you do so, you will be prompted to make sure, and then the magic happens.

Because Azure is only making a small configuration change, the cutover only takes a moment.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

2.4.6 Billing, and the tools accountants need

Wow. In only a few pages we have created a web site to sell awesome shirts, tested it in a local simulation of the cloud, published it to staging, and promoted it to production. Now that business is rolling in, we want to monitor how our application is costing us.

Microsoft provides detailed information on your usage of the Azure platform on a regular basis. You can look at these reports at any time, and see where you are.

***** The analytic tools are offline in preparation for RTW. Once the new tools are available, we will include the relevant information here. That is all. Return to your reading. We will show the billing screens, and go over what the billing system tracks for you.*****

2.5 Summary

In this chapter we took a broad tour of how to start working with Azure. The SDK is important to be able to build Azure applications, including all of the different tools, APIs and documentation needed to get started.

In just a few pages, if you were following along, we developed and deployed a web application. I have never been able to deploy, to production, with several load balanced nodes, a web application without weeks or months of meetings, arguments, and planning around deployment and the production configuration.

The one thing you should have noticed is how little difference there is, code wise, from what you would normally write for a web site. The web application we built had nothing special about it, and it would run on a normal on-premises server. Azure is just another place to run your code, much more dynamic than any data center you have ever worked with, but just the same. Of course our sample was simple, and didn't delve into the more advanced scenarios, but the point is still valid. The strength of Azure, for a developer, is how NOT different it is from what you do every day.

Deployment was almost a trivial matter. It didn’t take a lot of effort to create a service, upload the package, and promote it to production. The simple staging and production environments make it easy to roll back to the old code in case the new deployment goes horribly wrong.

We will look into what is under the covers in the next chapter, and see what Microsoft had to do to build such a powerful platform, and keep it so simple.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

3

How Windows Azure works

This chapter covers

- How Microsoft built Azure
- What a cloud operating system is
- How your application is provisioned and managed in the cloud

Now that we have a basic understanding of what you can do with Azure, we will start drilling deeper into the pieces of Azure, and how to best work with them. In this chapter we are going to discuss how Windows Azure is architected, and how it does the cloud magic that it does. Understanding this background will help you understand part 3 of this book, develop better services, how to be a better person, and get the most out of your Azure infrastructure.

3.1 Betting the house on the cloud

When Azure was first announced at the PDC in 2008, Microsoft wasn't a recognized player in the cloud industry. They were the underdog to the giants of Google and Amazon, who had been offering cloud services for years by that time. Building and deploying Azure was a big bet for Microsoft. It was a major change in culture, from where Microsoft had been, and where it needs to go in the future. Up until now, Microsoft had been a product company. They would design and build a product, burn it to CD, and sell it to customers. Over time the product would be enhanced, but the product was installed and operated in the client's environment. The trick was to build the right product at the right time, for the right market.

With the addition of Ray Ozzie to the Microsoft culture, there was a giant shift toward services. They weren't abandoning the selling of products, but they were expanding their expertise and portfolio to offer their products as services. Every product team at Microsoft was asked if what they were doing could be enhanced and extended with services. They wanted to do much more than just put Exchange in a datacenter, and rent it to customers. This became a fundamental shift in how Microsoft developed code, how they shipped it, and how they marketed and sold to their customers.

This wasn't just some executive whim, thought up during some exclusive executive retreat at some resort we will never be able to afford to even drive by. It was based on the trends and patterns the leaders saw in the market, in the needs of their customers, and the continuing impact of the Internet on our world.

They saw that people needed more flexible use of their resources, more flexible than even the advances in virtualization were providing. Companies needed to easily respond to their product's sudden popularity as social networking spread the word. Modern businesses were screaming that they can't wait six months for an upgrade to their infrastructure. They needed it now.

Customers were also becoming more sensitive to the massive power consumption and heat generated by their datacenters. Power and cooling bills were often the largest component of their total datacenter cost.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Coupling this with a concern over global warming, customers were starting to talk about the 'greening of IT.' They wanted to reduce the carbon footprint that these beasts produced. Reduce not only the power and cooling, but also the waste of lead, packing materials and the massive piles of soda cans from the server administrators.

3.1.1 The data centers of yore

Microsoft is continually improving all of the important aspects of their data center. They closely manage all of the costs of a data center, including power, cooling, staff, local laws, risks of disaster, availability of natural resources, and many other factors. While doing this, they have now designed their fourth generation of data centers. They didn't just skip to the end game; they built on their deep expertise in running global datacenters for the past few decades.

The first generation of data centers is still the most common in the world. Think of the special room with servers in it. It has racks, cable ladders, raised floors, cooling, UPSs, maybe a backup generator, and it is cooled to the temperature that could safely house raw beef. The focus is placed on making sure the servers are running, and absolutely no thought or concern for the operating costs of the data center. These data centers were optimized for the capital cost of building it, rarely thinking beyond the day the center opens.

Just as a side note, the collection of servers under your desk does not qualify as a generation 1 data center. Please be careful not to kick a cord loose while you do your work.

Generation 2 data centers take all of the knowledge learned through running generation 1 data centers, and apply a healthy dose of thinking about what happens on the second day of operation. Tactics are used to reduce the ongoing operational costs of the center through optimizing for sustainability and energy efficiency.

To meet these goals, Microsoft powers their Quincy, Washington data center with clean hydroelectric power. Their data center in San Antonio, Texas uses recycled civic gray water to cool the data center, reducing the stress on the water sources and infrastructure in the area.

3.1.2 The latest data centers of Azure

Even with the advances found in generation 2 data centers, a company cannot find the efficiencies and scale needed to combat rising facility let alone meet the demands that the cloud would generate. The density of the data center would need to go up dramatically, and the costs of operations would have to plummet. The first generation 3 data center, located in Chicago, Illinois, went online on June 20th, 2009. It is considered a 'mega data center' by Microsoft, which is a class designation that defines how large the data center is. The Chicago data center looks like a large parking structure, with parking spaces and ramps for tractor trailers. Servers are placed into containers, called Cblox, and parked in the structure.

Cblox are made out of the shipping containers that you see on ocean going vessels, and on eighteen wheelers on the highways. They are built very sturdily, and follow a standard size and shape that are easy to move around. A Cblox can hold anywhere from 1,800 to 2,500 servers, each. This is a massive increase in the density of the data center, 10 times more than a traditional data center. The Chicago mega data center holds about 360,000 servers, and is the only primary consumer from a dedicated nuclear power plant cre from Chicago Power & Light. How many of your data centers are nuclear powered?

Each 'parking spot' in the data center is rooted by a refrigerator sized device that acts as the primary interconnect to the rest of the data center. Microsoft defined a standard coupler that provides power, cooling, and network access to the container. Using this interconnect, and the super dense containers, they can add massive amounts of capacity in a matter of hours. Just compare how long it would take your company to plan, order, deploy, and configure 2,500 servers. It would take at least a year, and a lot of people, not to mention how long it would take just to recycle all of the cardboard and 'extra' parts you always seem to have after racking a server. Their goal with this strategy is to make it as cheap and easy as possible to expand capacity as demand increases.

The containers are built to Microsoft's specifications by a vendor, and delivered on site, ready for burn in tests, and allocation into the fabric. The container includes networking gear, cooling equipment, servers, racks, and are sealed against the weather.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Not only are the servers now packaged, and deployed in containers, but the needed generators, and cooling machinery are as well. In order to setup an edge data center, one that is located close to a large demand population, all they need is the power and network drops, and a level paved surface. The trucks with the power and cooling equipment first show up, and are deployed. Then the trucks with the computing containers back in, and drop their trailers, leaving the containers on the wheels that were used to deliver them. The facility is protected by a secure wall and doorway, with monitoring equipment. The use of laser fences is pure speculation, and just a rumor, as far as we know. The perimeter security is important, because the edge data center doesn't have a roof! Yes, no roof! This reduces the construction time, and the cooling costs. A roof isn't needed because the containers are completely sealed.

Microsoft opened a second mega data center, the first outside the United States, in Dublin, Ireland on July 1st, 2009. When Azure became commercially available in January, 2010 there were six known Azure data centers including Washington, Texas, Chicago, Ireland, Belgium, Singapore and Hong Kong. **[TODO: Need to confirm these are the real cities.]** While Microsoft will not list where all of their data centers are for security reasons, they say they have more than 10, and fewer than 100 data centers. Microsoft already has data centers all over the world to support their existing services, such as Virtual Earth, Bing Search, XBox Live, and others. If we just assume there are only 10, and each one is as big as Chicago, then Microsoft will need to manage 3.5 million servers as part of Azure. That is a lot of work

3.1.3 How many server huggers do you need?

Data centers are staffed with IT Pros to care and feed the servers. Data centers need a lot of attention, ranging from hardware maintenance, to backup, disaster recovery, and monitoring. Think of your company. How many people are allocated to manage your servers? Depending on how optimized your IT center is, the ratio of person to server can be anywhere from 1:10 to 1:100 in the typical company. With that ratio, Microsoft would need 35,000 server managers. Hiring that many server administrators would be hard, considering Microsoft employs roughly 90,000 people already.

To address this demand, they designed Azure for as much automation as possible, with a strategy called 'Lights out operations.' This strategy seeks to centralize and automate as much of the work as possible, by reducing complexity and variability. This results in a person to server ratio closer to 1:30,000, or higher.

Microsoft is achieving this level of automation with mostly off the shelf software, their own. They are literally eating their own dog food. They are using System Center Operations Manager, and all of the related products to oversee and automate the management of the underlying machines. They have built out custom automation scripts and profiles, much like any customer could do.

One key strategy in effectively managing a massive amount of servers is to provision them with identical hardware. In traditional data centers where we have worked, each year brought the latest and greatest of server technology, resulting in a wide variety of technology and hardware diversity. We even gave each server a distinct name, such as Protoss, Patty, and Zelda. With this many servers, you can't name your servers, you have to number them. Not just by server, but by rack, room, and facility. Diversity is usually a great thing, but not when you are managing millions of boxes.

The hardware in each Azure server is optimized for power, cost, density, and management. The optimization process drives out which exact motherboard, chipset, and every other component needs to be in the server. This is truly 'Bang for your buck' in action. Then they stick to that server recipe for a specific lifecycle, only moving to a new bill of materials when there are significant advantages to doing so.

3.1.4 Datacenter: The next generation

Microsoft isn't done. They have already spent years planning the fourth generation of data centers. Much like the edge data center mentioned above, the whole data center is outside. While the containers make it easy to scale out the computing resources as demand increases, prior generations of data centers still had to have the complete data center shell built out and provisioned. This meant provisioning the cooling and power systems as if the data center was at maximum capacity from day 1. The older systems were too expensive to expand dynamically. The fourth generation data centers are using an extendable spine of infrastructure that the

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

computing containers need, so that both the infrastructure and the computing resources are easily scaled out. All of this is outside, in a field of grass, without a roof. They will be the only data centers in the world that needs a grounds crew.

Figure 3.1 – Generation 4 datacenters are built on extensible spines. This makes it easy to not only add computational capacity, but the required infrastructure as well, including power and cooling.

Ok, we're impressed. They have a lot of servers, and some of them are even outside, and they have found a way to manage them all in an effective way. But how does the cloud really work?

3.2 Windows Azure, an operating system for the cloud

Think of the computer on your desk today. When you write code for that computer, you don't have to worry about which sound card it uses, which type of printer it is connected to, or which or how many monitors are used for the display. You don't worry, to a degree, about the CPU, about memory, or even how storage is provided (SSD, carrier pigeon, or hard disk drive). The operating system on that computer provides a layer of abstraction away from all of those gritty details, and frees you up to focus on the application you need to write, and makes it easy to consume the resources you need to get that The desktop operating system protects you from the details of the hardware, allocates time on the CPU to the code that is running, makes sure that code is allowed to run, plays traffic cop by controlling shared access to resources, and generally holds everything together. Now think of that enterprise application you want to deploy. You need DNS, networking, shared storage, load balancers, plenty of servers to handle load, a way to control access and permissions in the system, and plenty of other moving parts. Modern systems can get complicated. Dealing with all of that complexity by hand is like compiling your own video driver; it doesn't actually provide any value to the business.

Windows Azure, the cloud operating system, does the same work as the desktop operating system, but on a grander scale, and for distributed applications. We already revealed what the fabric was in the first chapter. In this section we are going to drill a little deeper into this fabric, and see how it really works. The fabric is comprised of the thousands of servers running, working together as a cohesive unit.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 3.2 – How the Fabric Controller is like the kernel of your desktop operating system. It is responsible for many of the same tasks.

Windows Azure takes care of the whole platform so you can focus on your application. The term fabric is used because of the similarity to a woven blanket. Each thread, on its own is weak, and can't do a lot. When they are woven together into a fabric the whole blanket becomes strong and warm. Understanding the relationships between your code, Azure, and the fabric controller will help you get the most out of the platform.

3.2.1 Managing the assets of the cloud

In Azure, you will not need to worry about which hardware, which node, what underlying operating system, or even how the nodes are load balanced or clustered. Those are just gritty details best left to someone else. You just need to worry about your application, and if it is operating effectively. How much time do you spend wrangling with these details for your on-premises projects? I bet it is at least 10-20% of the total project cost, just in meetings alone. There are savings to be gained by abstracting away these issues.

In fact, Azure manages much more than just servers. There are plenty of other assets that are managed. In addition to servers, Azure manages routers, switches, IP addresses, DNS servers, load balancers, and dynamic VLANs. There is a lot of complexity in managing these assets in a static data center. It is doubly more complex when you are managing multiple data centers that need to operate as one cohesive pool of resources, in a dynamic and real-time way.

If the Azure fabric is the operating system, then the fabric controller is the kernel.

3.3 The Fabric Controller

Operating systems have, at their core, a kernel. This kernel is responsible for being the traffic cop in the system. It manages the sharing of resources, schedules the use of precious assets (CPU time), allocates work streams as appropriate, and keeps an eye on security as well.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 3.3 – The relationship between Azure, the fabric, and the fabric controller. The fabric is an abstract model of the massive number of servers in the Azure data center. The fabric controller manages everything.

The Azure fabric has a kernel called the fabric controller, or FC. It handles all of the jobs a normal operating system's kernel would handle. It manages the running servers, deploys code, and makes sure that everyone is happy and has a seat at the table.

3.3.1 The Fabric Controller is just an application

The FC is an Azure application in and of itself, running multiple copies of itself for redundancy sake. It is largely written in managed code. The FC contains the complete state of the fabric internally, and this is real time replicated to all of the nodes that are part of the FC. In case one of the primary nodes goes offline, the latest state information is available to the remaining nodes, which then elect a new primary node.

The FC manages a state machine for each service deployed, setting a goal state based on what the service model for the service requires. Everything the FC does is in an effort to reach this state, and then to maintain that state once it is reached. We will go into detail of what the service model is in the next few pages, but for now, just think of it as a model that defines the needs and expectations that your service has.

3.3.2 How the Fabric Controller works: driver model

The FC follows a driver model, just like a normal operating system. Windows has no idea how to specifically work with your video card. It does know how to speak to a video driver, which in turn knows how to work with a specific video card. The FC works with a series of drivers, for each type of asset in the Azure fabric. This includes the machines, as well as the routers, switches, and load balancers.

While the variability of the environment is very low today, over time, new types of each asset is likely to be introduced. While the goal is to reduce unnecessary diversity, there will be business needs to offer breadth in the platform. Perhaps you might get a software load balancer for free, but you will have to pay a little bit more per month to use a hardware load balancer. A customer would do this to meet a specific need, and the FC would have different drivers for each type of load balancer in the fabric.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 3.4 - How the lifecycle of an Azure service progresses to a running state.

The FC uses these drivers to communicate the commands it needs to send to each device to reach the desired running state. This might be the command to create a new VLAN to a switch, or allocate a pool of virtual IP addresses. These commands help the FC move the state of the service towards the goal state. Figure 3.4 shows the progression of state, from the developer writing the code, and defining the service model, to the FC allocating and managing the resources the service requires.

3.3.3 Allocating Resources

One of the key jobs of the FC is to allocate resources to services. It does this by analyzing the service model of the service, including the fault and update domains, and the availability of resources in the fabric. Using a greedy resource allocation algorithm, it finds which nodes can support the needs of each instance in the model. Once it has reserved the capacity, it updates the FC data structures in one transaction. Once this is done, the goal state of each node has been changed, and the FC starts moving each node towards its goal state by deploying the proper images and bits, starting up services, and issuing other commands through the driver model to all of the resources needed for the change.

3.3.4 Instance Management

The FC is also responsible for managing the health of all of the nodes in the fabric, as well as the health of the services running. If it detects a fault in a service, it will try to remediate that fault, perhaps by restarting the node, or taking it offline and replacing it with a different node in the fabric.

When a new container is added to the data center, the FC performs a series of burn-in tests to ensure that the hardware delivered is working properly. Part of this process results in the new resources being added into the inventory for the data center, making it available to be allocated by the FC.

If hardware is ever to be determined to be faulty, either during installation, or during a fault, the hardware is flagged as unusable in the inventory, and left alone until later. Once a container has enough failures, the

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

remaining workloads are moved to different containers, and then the whole container is taken offline for repair. Once the problems have been fixed, the whole container is retested and returned into service.

3.4 The service model and you

The driving force behind what the FC does is the *service model* that you define for your service. You define the service model in an indirect manner. When you are developing a service, you define the following:

- some configuration on what the pieces to your service are
- how the pieces communicate
- expectations you have about the availability of the service

Figure 3.5 – The Service Model is comprised of several different pieces of information. This model helps Azure run your application correctly.

The service model is broken into two pieces of configuration and deployed with your service. Each piece focuses on a different aspect of the model.

3.4.1 Defining configuration

Your solution in Visual Studio will contain these two pieces of configuration in different files, both found in the Azure Service project in your solution.

- service definition file (`ServiceDefinition.csdef`)
- service configuration file (`ServiceConfiguration.cscfg`).

The service definition file defines what the roles are in your service, and what their communication endpoints are. This would include public HTTP traffic for a website, or the endpoint details for a web service. You can also configure your service to use local storage (which is different from Azure storage), and any custom configuration elements of the service configuration file. The service definition cannot be changed at runtime. Any change would require a new deployment of your service. Your service is restricted in using only the network endpoints and resources that are defined in this model. You can think of this piece of the configuration as defining what the infrastructure of your service is, and how the parts fit together.

The service configuration file includes the entire configuration needed for the role instances in your service. Each role has its own dedicated part of the configuration. The contents of the configuration file can be changed at runtime, which removes the need to redeploy your application when some part of the role configuration

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

changes. You can also access the configuration in code, in a similar manner that you might read a web.config file in an ASP.NET application.

3.4.2 Adding a custom configuration element

In many applications we store connections strings, default settings, and secret passwords (please don't!) in the app.config or web.config. You will often do the same with an Azure application. We first need to declare the format of the new configuration setting in the `.csdef` file. We do this by adding a `ConfigurationSettings` node inside the role we want the configuration to belong to. We do this to define the schema of the `.cscfg` file for that role. This essentially strongly types the configuration file itself. If there is an error in the configuration file during a build you will receive a compiler warning. This is a great feature, because there is nothing worse than deploying code when there is a simple little problem in a configuration file.

```
<ConfigurationSettings>
  <Setting name="SuperSecretPassword"/>
</ConfigurationSettings>
```

Now that we have told Azure the new format of our configuration files, namely that we want a new setting called `WelcomeBannerText`, we can add that node to the service configuration file. We add the following XML into the appropriate role node in the `.cscfg` file.

```
<ConfigurationSettings>
  <Setting name="SuperSecretPassword" value="KlatuBaradaNikto"/>
</ConfigurationSettings>
```

During runtime, we want to read in this configuration data, and use it for some purpose. Remember that all configuration settings are stored as strings and must be casted to the appropriate type as needed. In this case we want a string to assign to our text label control, so we can use it as is.

```
txtPassword.Text = RoleEnvironment.GetConfigurationSettingValue("BannerText");
```

Having lines of code like this all over your application can get messy, and hard to manage. Many times developers will consolidate their configuration access code into one location.

3.4.3 Centralizing file-reading code

It is a best practice to take your entire configuration file reading code, where ever it is sprinkled, and move it into a `ConfigurationManager` class of your own design. Many people use the term serviced instead of manager, but I think the term service is too overloaded, and manager is just as clear. This centralizes all of the code that knows how to read the configuration in one place, making it easier to maintain. More importantly, it removes the complexity of reading the configuration from the relying code, which is the principal of separation of concerns. This also makes it easier to mock out the implementation of the `ConfigurationManager` class for easier testing purposes. Over time when the APIs for accessing configuration change, or if where your configuration lives changes you will have only one place to go to make the changes you need.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 3.6 – A well designed ConfigurationManager class can centralize the busywork of working with the configuration system.

Reading configuration data in this manner may look very familiar to you. You have probably done this for your current applications, reading in the settings stored in a `web.config` or an `app.config` file. When migrating an existing application to Azure you might be tempted to keep the configuration settings where they are. While keeping them in place reduces the amount of change to your code as you migrate it to Azure, it does come at a cost. Unfortunately the configuration files that are part of your roles are frozen, and are read only at run time. You cannot make changes to them once your package is deployed. If you want to change settings at runtime you will need to store those settings in the `.cscfg` file. Then when you want to make a change, you only have to upload a new `.cscfg` file, or click configure on the service management page in the portal.

The FC takes these configuration files, and builds a sophisticated service model that it uses to manage your service. At this time, there are about three different core model templates that all other service models inherit from. Over time, Azure will expose more of the service model to the developer, so that they can have a more fine grained control over the platform their service is running on.

3.4.4 The many sizes of roles

Each role defined in your service model is basically a template for a server you want deployed in the fabric. Each role can have a different job, and a different configuration. Part of that configuration includes local storage, and the number of instances of that role that should be deployed. How these roles connect and work together is part of why the service model exists.

Since each role might have different needs there are a variety of virtual machine sizes that you can request in your model. Each step up in the VM size is basically doubling the resources of the step below it.

Table 3.1 The available sizes of the Azure virtual machines

VM Size	Dedicated CPU Cores	Available Memory	Local Disk Space
Small	1	1.7 GB	250 GB
Medium	2	3.5 GB	500 GB
Large	4	7 GB	1,000 GB
Extra Large	8	15 GB	2,000 GB

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Each size is basically a slice of how big a physical server is. This makes it easy to allocate resources, and keeps the numbers round. Since each physical server has eight CPU cores, allocating an Extra Large VM to a role is like dedicating a whole physical machine to that instance. This gives you all the CPU, RAM, and disk available on that machine. Which size you want is defined in the ServiceDefinition.csdef file, on a role by role basis. The default size, if you don't declare one, is small.

```
<WorkerRole name="ImageCompresser" vmsize="ExtraLarge">
```

If you are using Visual Studio 2010, you can define the role configuration in the properties screen of the role in the Azure project, like in figure 3.x.

Figure 3.x – Configuring your role doesn't have to be a gruesome XML affair. You can easily do it in Visual Studio 2010 when you view the properties sheet for the role you want to configure.

The service model is also used to define fault domains and update domains, which we'll look at next.

3.5 It's not my fault

Fault domains and update domains determine what portions of your service can be offline at a single time, but for different reasons. They are the way that you define your uptime requirements to the fabric controller, and how you describe how your service updates will happen when you have new code to deploy.

3.5.1 Fault domains

Fault domains are used to make sure that a set of elements in your service are not tied to a single point of failure. Fault domains are based more on the physical structure of the data center than it is on your architecture. Your service would typically want to have three or more fault domains. If you had one fault domain, all of the parts of your service could possibly be running on one rack in the same container, connected to the same switch. This would lead to a high likelihood of catastrophic failure for your service if there is any failure in that chain. If that rack fails, or the switch in use fails, then your service is completely offline. By breaking your service into several fault domains, the FC will make sure those fault domains do not share any dependent infrastructure, to protect your service against single points of failure. In general you will define three fault domains, meaning only about a third can become unavailable because of a single fault. In a failure scenario, the fabric controller will immediately try to deploy your roles to new nodes in the fabric to make up for the failed nodes.

At this time the Azure SDK and service model does not let you define your own number of fault domains, and is believed to default to three domains.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

3.5.2 Update domains

The second type of domain defined in the service model is the update domain. The concept of an update domain is similar to a fault domain. An update domain is the unit of update you have declared for your service. An update is rolled out across your service, one update domain at a time. Cloud services tend to be big, and tend to always need to be available. The update domain allows a rolling update to be used to upgrade your service, without having to bring the entire service down. These domains are usually defined to be orthogonal to your fault domains. In this manner, if an update is being pushed out, while there is a massive fault, you won't lose all of your resources, just a piece of them.

You can define the number of update domains for your service in your `servicedefinition.cfg` file, as part of the `serviceDefinition` tag at the top of the file.

```
<ServiceDefinition
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition"
name="DayOldSuchiOnline" upgradeDomainCount="3">
```

If you do not define your own update domain setting the service model will default to five update domains. Your role instances are assigned to update domains as they are started up, and the fabric controller tries to keep the domains balanced with regards to how many instances are in each domain.

3.5.3 A service model example

If we had a service running on Azure, we might need six role instances to handle the demand on our service, but request nine instances instead. We do this because we want a high degree of tolerance in our architecture. We would have three fault domains and three update domains defined. This way, if there is a fault, only a third of our nodes are affected. Also, only a third of the nodes will ever be updated at one time, controlling the amount of nodes taken out of service for updates, as well as reducing the risk of any update taking down the whole service. Figure 3.7 illustrates this architecture.

Figure 3.7 - How fault and update domains help increase fault tolerance in your cloud service. In this figure we have three instances of each of three roles.

In this above scenario, the broken switch might take down the first fault domain, but the other two fault domains would not be affected, and be able to keep operating. The FC can manage these fault domains because of the detailed models it has for the Azure assets.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The cloud is not about perfect computing, it is about deploying services, and managing systems that are fault tolerant. You need to plan for the faults that **will** happen. The magic of cloud computing makes it easy to scale big enough that a few node failures doesn't really impact your service.

All of this talk about service models and an overlord FC is nice, but at the end of the day, the cloud is built from individual pieces of hardware. And there is a lot of hardware, and it all needs to be managed in a hands-off way. There are several approaches to apply updates to a service that is running. You can perform both manual and automated rolling upgrades, or you can perform a full static upgrade (also called a VIP swap).

3.6 Rolling out new code

No matter how great your code is you will have to perform an upgrade at some point in time, if for no other reason than to deploy a new feature a user has requested. It is important that you have a plan for updating the application, and have a full understanding of the moving parts. There are different approaches for different upgrade scenarios.

There are two broad approaches to rolling out an upgrade, either a static upgrade, or a rolling upgrade. You should carefully plan your application architecture to avoid the necessity of a static upgrade because it impacts the uptime of your service, and can be more complicated to roll out, where a rolling upgrade keeps your service up and running the whole time. You should also always perform the upgrade in the staging environment first to make sure the deployment goes well. Once a full battery of end to end and integration tests are passed, you can proceed with your plans for the production environment.

Another gotcha is if the number of endpoints for a role has changed you will not be able to do either type of upgrade, you will be forced to use the old fashioned 'nuke and pave' strategy. You will have to completely stop your service, delete your deployment completely, and then redeploy from scratch. This is not a great solution, so you should try to avoid it if you can.

3.6.1 Static upgrades

A static upgrade is sometimes referred to as a 'fork lift upgrade,' because you are touching everything all at once. You usually need to do a static upgrade when there is a significant change in the architecture and plumbing of your application. Perhaps there is a whole new architecture of how the services are structured, and the database has been completely redesigned. In these cases it can be hard to upgrade just one piece at a time because of interdependencies in the system. This type of upgrade is required if you are changing the service model in any way.

This approach is also called a VIP swap because the fabric controller is just swapping the virtual IP addresses that are assigned to your resources. When a swap is done, the old staging environment becomes your new production environment, and your old production environment becomes your new staging environment. This can happy pretty fast, but your service will have downtime while it is happening, so you should plan on that. The one great advantage to this approach is you can easily swap things back to the way they were if things aren't working out. Your upgrade plan should consider how long the new staging (aka old production) environment should stay around. You might want to keep it around for a few days until you know the upgrade has been successful. At that point you can completely tear down the environment to save resources and money.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 3.8 – Performing a VIP swap upgrade is as easy as clicking the arrows. If things go horribly awry you can always swap back to the way things were. It's like rewind for your environment.

You can perform a VIP swap by logging into the Azure portal, and going to the summary screen for your service. At this point you need to deploy your new version to the staging environment. Once everything is all setup, and you are happy with it, you can click the circular button in the middle. The changeover will only take a few minutes.

You can use the service management API to perform the swap operation as well. You need to POST the following command to the address `https://management.core.windows.net/<subscription-id>/services/hostedservices/<service-name>`. Your subscription id can be found at the bottom of the account tab on the Azure portal. Your service name is the unique name you gave to the service when you created it.

```
<?xml version="1.0" encoding="utf-8"?>
<Swap xmlns="http://schemas.microsoft.com/windowsazure">
  <Production>Hello Fabric</Production>
  <SourceDeployment>v2v2</SourceDeployment>
</Swap >
```

You need to supply the names of the deployments you want to swap in the payload of the message. In my example, I want to swap the Hello Fabric deployment with the v2v2 deployment. This is one case where you really want to make sure you have named your deployments clearly, at least more clearly than I did in this example.

3.6.2 Rolling Upgrades

VIP swaps are nice, but some customers needed more flexibility in how they performed their rollouts. If your roles are carrying state and you don't want to lose that state as you completely change to a new set of servers, then this is for you. Another key scenario is when you only want to upgrade the instances of a specific role, instead of all of the roles. For example, you might want to deploy an updated version of the web site, without impacting the processing of the shopping carts that is being performed by the backend worker roles. You can of course choose to upgrade all of the roles, or just a specific role. Remember that when doing a rolling upgrade

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

you cannot change the service model of the service you are upgrading. This means if you have changed the structure of the service configuration, the number of endpoints, or the number of roles you will have to do a VIP swap instead.

When you perform an automatic rolling upgrade, the fabric controller drains the traffic to the set of instances that are in the first update domain (they are numbered starting with zero) by removing them from the load balancer's configuration. Once the traffic is drained, the instances are stopped, the new code is deployed, and then they are restarted. Once they are back up and running, they are added back into the load balancer's list of machines to route traffic to. At this point the fabric controller moves onto the next update domain in the list. It will proceed in this fashion until all of the update domains have been serviced. Each domain should only take a few minutes.

If your scenario requires that you control how the progression moves from one domain to the next you can choose to do a manual rolling upgrade. When you choose this option, the fabric controller will stop after a domain, and wait for your permission to move onto the next. This gives you a chance to check the status of the machines, and the environment before moving forward with the rollout.

Figure 3.9 – Performing a rolling upgrade is easy. Choose 'upgrade' on the service detail page and choose your options. You can

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

upgrade all of the roles or just one role in an upgrade.

To perform a rolling upgrade you will need to go to the service detail screen on the Azure portal, and click the upgrade button for the deployment you want to upgrade. I did this on my production deployment in figure 3.x. At this point you can choose the automatic roll or the manual roll for the upgrade, and if you are going to upgrade all the roles in the package, or just one of them. Like a normal deployment you will also need to provide a service package, configuration, and a deployment name.

If you choose a single role to upgrade then only the instances for that role in each domain will be taken offline for upgrading. The other roles' instances will be left untouched.

You can perform a rolling upgrade by using the service management API as well. When using the management API you will have to store the package and configuration files in blob storage before starting the process. Like with a VIP swap, you will need to POST a command to a specific URL. You will need to customize the URL to match the settings for the deployment you want to upgrade.

```
https://management.core.windows.net/<subscription-id>/services/hostedservices/<service-
name>/deployments/<deployment-name>/?comp=upgrade
```

The body of the command will need to contain the following elements. It will need to be changed to supply the parameters that match your situation. This sample is to do a fully automatic upgrade on all the roles.

```
<?xml version="1.0" encoding="utf-8"?>
<UpgradeDeployment xmlns="http://schemas.microsoft.com/windowsazure">
  <Mode>auto</Mode>
  <PackageUrl>http://azureinaction.blob.core.windows.net/deplyment_container/new_code.cspkg
</PackageUrl>
  <Configuration>***the contents of the config file***</Configuration>
  <Label>v3.2</Label>
</UpgradeDeployment>
```

Performing a manual rolling update with the service management API is a little trickier, and requires several calls to the `WalkUpgradeDomain` method. The upgrades are performed in an asynchronous nature, so the first command will start the process. As it is being performed you can check on the status by using `Get Operation Status` with the operation id that was supplied to you when you started the operation.

We have covered how to upgrade running instances, and we have talked about what the fabric is. Now we will go one level deeper and explore the underlying environment.

3.7 The bare metal

No one, outside of the Azure team, truly knows the nature of the underlying servers and other hardware, and that is ok, since it is all abstracted away by the cloud OS. But we can still look at how our instances are provisioned and how automation is used to do this without hiring the entire population of southern Maine to manage it.

Each instance is really a virtual machine running Windows Server 2008 Enterprise Edition x64 bit, on top of *Hyper-V*. Hyper-V is Microsoft's enterprise virtualization solution, and is available to anyone. Hyper-V is based on a *hypervisor*, which manages and controls the virtual servers running on the physical server. One of the virtual servers is chosen to be the host OS. The host OS is a virtual server as well, but has additional responsibilities in managing the hypervisor, and the underlying hardware.

Hyper-V has two features that help in maximizing the performance of the virtual servers, while reducing the overall cost of running those servers. Both features need to be supported by the physical CPU.

3.7.1 Free parking

The first is core and socket parking. Hyper-V can monitor the utilization of each core and CPU as a whole (which is in a socket on the motherboard). Hyper-V will move the processes around on the cores to consolidate the work to as few cores as possible. Any cores not needed at that time will be put into a low energy state. They can come

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

back online quickly if needed, but will consume much less power while they wait. Hyper-V can do this at the socket level as well. If it notices that each CPU socket is only being used at 10%, for example, it can condense the workload to one socket, and park the unused sockets, placing them in a low energy state. This helps data centers use less power, and require less cooling.

3.7.2 A special blend of spices

The version of Hyper-V used by Azure is a special version that the team created through the removal of anything they didn't need. Their goal was to reduce the footprint of Hyper-V as much as possible to make it faster, and easier to manage. Since they knew exactly the type of hardware, and guest operating systems that will run on it, they could rip out a lot of code. For example, they removed support for 32bit hosts and guest machines, they removed the support needed for other types of operating systems, and for hardware they weren't supporting at all.

Not stopping there, they further tuned the hypervisor scheduler for better performance while working with data center workloads. They wanted the scheduler to be more predictable in its use of the resources, and fairer to the different workloads running, since each would be running at the same priority level. They also enhanced it in certain ways to support a heavier I/O load on the VM Bus.

3.7.3 Creating instances on the fly

When a new server is ready to be used, it is booted. At this point it is a naked server, with a bare hard drive. We can see this process in figure 3.10.

Figure 3.10 - The structure of a physical server, and virtual instance servers in Azure.

[The following #'s are cue balls to the diagram above.]

During boot up, the server locates a maintenance OS on the network using standard PXE protocols. PXE stands for pre-execution environment, and it is a process for booting to an operating system image that can be

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

found on the network. It then downloads the image and boots to it (#1). This maintenance OS is based on Windows PE, and is a very thin OS that is used by many IT organizations for low level trouble shooting and maintenance. The tools and protocols to do this are available on any Windows server, and are used by a lot of companies to easily distribute machine images and automate deployment.

The maintenance OS connects with the FC, and acts as an agent for the FC to execute any local commands needed to prepare the disk and machine. The agent prepares the local disk, and streams down a Windows Server 2008 Server Core image to the local disk (#2). This image is a VHD (Virtual Hard Drive), and is a common file format to store the contents of hard drives for virtual machines. VHD's are basically large files, representing the complete or partial hard drive for a virtual machine. The machine is then reconfigured to boot from this Core VHD. This image becomes the host OS that will manage the machine, and interact with the hypervisor. The host OS is Windows Server 2008 Core because it has almost all, but the most necessary modules, removed from the operating system. This is a version you may be running in your own data center.

The Azure team worked with the Windows Server team to develop the technology needed to boot a machine natively from a VHD that is stored on the local hard drive. The Windows 7 team liked the feature so much they added it to their product as well. Being able to boot from a VHD is a key component of the Azure automation.

Once the machine has rebooted using the host OS image, the maintenance OS is removed, and the FC can start allocating resources from the machine to services that need to be deployed. A base OS image is selected from the prepared image library that will meet the needs of the service that is being deployed (#3). This image (a VHD file) is streamed down to the physical disk. The core OS VHDs are marked as read only, allowing multiple service instances to share a single image. A differencing VHD is stacked on top of the read only base OS VHD to allow for changes specific to that virtual server (#4). It is possible that different services will have different base OS images, based on the service model applied to that service.

On top of the base OS image is an application VHD that contains additional requirements for your service (#5). It is attached to the base OS image, the bits for your service are downloaded to the application VHD (#6), and then the stack is booted. As it starts up, it reports its health to the FC. The FC then enrolls it into the service group, configuring the VLAN assigned to your service, updating the load balancer, IP allocation, and DNS configuration. Once this is all completed, the new node is ready to service requests to your application.

Much of the image deployment can be made before the node is needed, cutting down on the time it can take to start a new instance, and add it to your service.

Each server can contain several virtual machines. This allows for the optimal use of computing resources and the flexibility to move instances around as needed. As a second or third virtual server is added, it may use the base OS VHD that has already been downloaded (#7), or it can download a different base OS VHD, based on its needs. This second machine then follows the same process of downloading the application VHD, and booting up, and enrolling into the cloud.

All of these steps are coordinated by the FC, and is usually accomplished in a few minutes.

3.7.4 Image is everything

If the key to the automation of Azure is Hyper-V, then the base virtual machine images and their management is the cornerstone. Images are centrally created, also in an automated fashion, and stored in a library, ready to be deployed by the FC as needed.

A variety of images are managed, allowing for the smallest footprint each role might need. If a role doesn't need IIS, then there is an image that doesn't have IIS installed. This is an effort to shrink the size and runtime footprint of the image, but also to reduce any possible attack surfaces, or patching opportunities.

All images are deployed using an 'xcopy deployment' model. This model keeps deployment really simple. If they relied on complex scripts, and tools, then they would never truly know what the state of each server would be, and it would take a lot longer to deploy an instance. Again, diversity is the devil in this environment.

This same approach is used when deploying patches. When the operating system needs to be patched, they don't download and execute the patch locally as you might on your workstation at home. This would lead to too much risk in having irregular results on some of the machines.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Instead, the patch is applied to an image, and the new image is stored in the library. The FC then plans a rollout of the upgrade, taking into account the layout of the cloud, and the update domains defined by the various service models that are being managed.

The updated image is copied in the background to all of the servers used by the service. Once the files have been staged out to the local disk, which can take some time, each update domain group is restarted in turn. In this way, the FC knows exactly what is on the server. The new image is merely wired up to the existing service bits that have already been copied locally. The old image is kept local for a period of time, as an escape hatch in case something goes wrong with the new image. If that happens, the server is reconfigured to use the old image, and rebooted, according to the update domains in the service model.

This process dramatically reduces the service window of the servers, increasing uptime, and reducing the cost of maintenance on the cloud.

3.8 The innards of the Windows Azure Web Role Virtual Machine

Your first experience with roles in Azure is likely to be with the Web Role. To help us develop our web applications more effectively it's worth looking in more detail at the virtual machine that our web role is hosted in. In this section we will look at the following items:

- The details of the VM
- The hosting process of our web role (RDRoleHost)
- How the RoleEnvironment class communicates with Windows Azure
- Other processes running in Windows Azure

3.8.1 Exploring the Virtual Machine Details

We can use the power of native code execution to see some of the juicy details about the Virtual Machine that our web role runs on. Figure 3.11 shows an ASP.Net Web Page that shows some of the internal details including:

- Machine Name
- Domain Name
- Username
- Etc

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 3.11 Using native code we can see some of the machine details of a web role in Windows Azure. We can see that Microsoft is using Windows Server 2008 x64. Notice the username the process is running as is a GUID.

If we want we can easily generate the above web page by creating a simple ASPX page with some labels that represent the text as shown below:

```
<div>
   <a href="MachineDetails.aspx">MachineDetails.aspx</a>
   Manufacturer: <asp:Label ID="lblManufacturer" runat="server" />
 </div>
```

Finally we can display the internal details of the Virtual Machine using the code-behind in Listing 3.1.

Listing 3.1 Code-behind for machine details

```
public partial class MachineDetails : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            // Initialize
            AzureEnvironment environment = new AzureEnvironment();

            var computer = new Microsoft.VisualBasic.Devices.Computer();
            lblMachineName.Text = computer.Name;

            // OS Details
            var osDetails = environment.GetOSDetails();
            lblOSName.Text = computer.Info.OSFullName;
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
            lblOSVersion.Text = computer.Info.OSVersion;
            lblMachineName.Text = computer.Name;
            lblWindowsDirectory.Text = osDetails.WindowsDirectory;

            // Computer System Details
            var computerSystemDetails = environment.GetComputerSystemDetails();
            lblManufacturer.Text = computerSystemDetails.Manufacturer;
            lblModel.Text = computerSystemDetails.Model;
            lblSystemType.Text = computerSystemDetails.SystemType;
            lblTotalPhysicalMemory.Text = computerSystemDetails.TotalPhysicalMemory;

            lblProcessorCount.Text = System.Environment.ProcessorCount.ToString();
            lblCLRVersion.Text = System.Environment.Version.ToString();
            lblCurrentDirectory.Text = GetCurrentDirectory();
            lblTimeSinceLastRestart.Text = GetTimeSinceLastRestart();
            lblDomainName.Text = System.Environment.UserDomainName;
            lblUserName.Text = System.Environment.UserName;
            lblCPUName.Text = GetCPUName();
            //lblPhysicalMemory.Text = GetPhysicalMemory();
            lblArchitecture.Text = GetArchitecture();
        }

        private string GetCurrentDirectory()
        {
            try
            {
                return System.Environment.CurrentDirectory;
            }
            catch
            {
                return "unavailable";
            }
        }

        private string GetTimeSinceLastRestart()
        {
            try
            {
                TimeSpan time = new TimeSpan(0, 0, 0, 0, System.Environment.TickCount);
                return time.ToString();
            }
            catch
            {
                return "unavailable";
            }
        }

        private string GetCPUName()
        {
            try
            {
                using (ManagementObject Mo = new
ManagementObject("Win32_Processor.DeviceID='CPU0'"))
                {
                    return (string)(Mo["Name"]);
                }
            }
            catch
            {
                return "unavailable";
            }

        }

        private string GetArchitecture()
        {
            try
            {
                using (ManagementObject Mo = new
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
ManagementObject("Win32_Processor.DeviceID='CPU0'"))
                {
                    ushort result = (ushort)(Mo["Architecture"]);
                    switch (result)
                    {
                        case 0:
                            return "x86";
                        case 9:
                            return "x64";
                        default:
                            return "other";
                    }
                }
            }
            catch
            {
                return "unavailable";
            }

        }
    }
```

You can now of course deploy your web page to Windows Azure and see the inner details of your web role. The really interesting things that we can tell from these machine details in Figure 3.11 are:

- Web Roles run on Windows 2008 Enterprise Edition x64 edition
- They run quad core AMD processors and you have 1 core assigned
- The domain name of the Web Role is CIS
- My VM has been running for an hour
- The Windows directory lives on the D:\ drive
- The Web Application lives on the E:\ drive

This is of course just the beginning, feel free to experiment and discover whatever information you need to satisfy your curiosity about the internals of Windows Azure by using similar calls in Listing 3.1.

PROCESSES LIST

Now that we're having a good rummage around the virtual machine, it might be worth having a look at what processes are actually running on the virtual machine. To do that we will build an ASP.NET Web Page that will return all the processes in a pretty little grid.

Figure 3.12 The Process List of a Windows Azure Virtual Machine. The two processes that start with 'RD' are related to "Red Dog", which was the code name for Azure while it was being developed.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

To generate the page shown in Figure 3.12 you should create a new web page in your web role with a GridView component called processGridView.

```
<asp:GridView ID="processGridView" runat="server"/>
```

You should then add the following code to the page_load event of your codebehind.

```
var processes = Process.GetProcesses();

processGridView.DataSource = from process in processes
                             orderby process.ProcessName
                             select new
                             {
                                Name = process.ProcessName,
                                Id = process.Id.ToString() };
                            }
processGridView.DataBind();
```

The code above will list all the processes on a server and bind the returned list to a GridView on a web page, as displayed in Figure 3.12. Since viewing processes is a security sensitive method you will need to enable native code execution on your web role to make this sample work.

If we look at the process list displayed in Figure 3.12 there are two Windows Azure specific services that we are interested in:

- RDRoleHost
- RDAgent

We will now spend the next couple of sections looking at these processes in more detail

3.8.2 The hosting process of our website (RDRoleHost)

If you were to look at the process list for a live web role as shown in Figure 3.12 or if you were to fire up your web application in Windows Azure and look at the process tab, you will notice that the typical IIS worker process (w3wp.exe) is not present when your web server is running.

You will also notice that if you stop your IIS Server (as shown below) then your web server continues to run.

```
IISReset -stop
```

We know from pre-requisites of installing the Windows Azure SDK that Web Roles are run under IIS7 so why can't we see our roles in IIS, or restart the server using iisreset?

HOSTABLE WEB CORE

Although Windows Azure uses IIS7.0 it makes use of a new feature called "Hostable Web Core" which allows you to host the IIS runtime in-process. In the case of Windows Azure, the **RDRoleHost.exe** process hosts the IIS7.0 runtime. If you were to look at process list on the live server or on the development fabric you will see that as you interact with the web server the utilization of this process changes.

WHY IS IT RUN IN-PROCESS RATHER THAN USING PLAIN OLD IIS?

The implementation of the Web Role is quite different from a normal web server. Rather than a system administrator managing the running of the web servers our datacenter overlord the Fabric Controller performs that task. Therefore the Fabric Controller needs the ability to interact and report on the web roles in a consistent manner. Rather than attempting to use the WMI routines of IIS the Windows Azure team opted for a custom WCF approach.

This custom in-process approach also allows your application instances to interact with the RDRoleHost processing using a custom API via the RoleEnvironment class.

3.8.4 Other processes running in Windows Azure

Now that we have an idea of how IIS7.0 is used to host your web role, it's worth looking at some of the other processes hosted in Windows Azure.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

RDAGENT

The RDAgent Process is responsible for collecting the health status of the role and for collecting management information on the VM including:

- Server Time
- Machine Name
- Disk Capacity
- OS Version
- Memory
- Performance Statistics (CPU Usage, Disk Usage)
- Etc

The role instance and the RDAgent processes will again use named pipes to communicate with each other. If the instance needs to notify the Fabric Controller of its current state of health then, this will be communicated from the web role to the RDAgent process using the named pipe.

All of the information collected by the RDAgent process is ultimately made available to the Fabric Controller to allow it to determine how to best run the data center. The Fabric Controller will use the RDAgent process as a proxy between itself, the VM and the instance. If the Fabric Controller decides to shutdown an instance it will instruct the RDAgent process to perform this task.

3.8 Summary

Hopefully you have learned a little bit of how Azure is architected, and how Microsoft runs the cloud operating system. Microsoft has spent billions of dollars and millions of work hours in building these datacenters, and the operating system that runs them.

Windows Azure is truly an operating system for the cloud, abstracting away the details of the massive datacenters, servers, networks, and other gear so you can simply focus on your application. The fabric controller controls what is happening in the cloud, and acts as the kernel in the operating system.

With the power of the fabric controller, and the massive data centers, you can define the structure of your system, and dynamically scale it up or down as needed. The infrastructure makes it easy to do rolling upgrades across your infrastructure, leading to minimal downtime of your service.

The service model is comprised of the service definition and service configuration files, and describes to the fabric controller how your application should be deployed and managed. This model is the magic behind the automation of the data center. Fault and update domains are used in the model to describe how the group of servers running your application should be separated to protect against failures and outages.

The amazing thing is Azure was built primarily from off the shelf parts and software. The team didn't code this from the ground up, but were able to tune the behavior of Hyper-V, and of Windows to suit their needs. They leveraged their deep experience in running large data centers, and worked with the Microsoft product teams to add the features they needed. These innovations, like the ability to boot natively from a VHD, will trickle into the mainstream as part of future versions of the products.

In the section we are going to walk through running web roles in Azure. A web role is what hosts a web application, and can be used in a variety of ways, as we will see in chapters 4 and 5. We will look at scaling the roles in chapter 6, and see how you can basically run any platform in Azure by running native code in chapter 7.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

4

It's time to run with the service

This chapter covers:

- Interacting with Windows Azure via the Service Runtime
- Defining your Windows Azure Role
- Configuring your Windows Azure Role

In the last chapter, we really got into the guts of the infrastructure and architecture of Windows Azure. During that chapter we introduced the concept of the Service Model and how it used by the Fabric Controller to manage our Role.

In this chapter, we will take some timeout to look at those bits of the Service Model that we get to mess around with (specifically the Service Definition and Service Configuration files).

4.1 Using the Windows Azure Service Runtime API

In both chapter 1 and chapter 2, we created a brand new Windows Azure Web Role from scratch. As we pointed out at the time, a Windows Azure hosted web role is just a regular old ASP.NET Web Application with a little bit of extra magic that allows you to interact with Windows Azure. That extra magic is 3 new assemblies that are automatically added to your new web application when you create a new Windows Azure Cloud Project within Visual Studio. Figure 4.1 shows these assemblies listed in the Visual Studio 2010 Solution Explorer.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 4.1 - Lots of new assemblies to play with in our new web role

In this chapter we will focus only on the **Microsoft.WindowsAzure.ServiceRuntime** assembly. We will look at the other 2 assemblies in future chapters.

The Service Runtime assembly acts as a bridge between the Windows Azure runtime and our application. Although we don't need to include the ServiceRuntime assembly in our web role, it's worth our while to do so. Without this assembly our applications would have no way to interact with Windows Azure and make use of the API's that it exposes to us. This assembly provides a bunch of helper routines that we will explore throughout this chapter including:

- Checking if our application is running in the cloud
- Retrieving Configuration Settings (see later)
- Getting a reference to a file held in the local cache (see later)

As you can see already, this new assembly provides some pretty valuable methods that we will need to fully utilize the power of Windows Azure (okay so that was a little over dramatic but hey this is a book). We'll now take a look at how we can include this assembly in our projects.

4.1.1 Adding the Service Runtime Assembly to our application

As you saw earlier, by default when you create a brand new ASP.NET Web Role within Visual Studio these new assemblies from the SDK are automatically referenced within your project. Well that's great in that instance but what do I do if I'm migrating an existing ASP.NET application to the Cloud, or if I want to access the API from another assembly?

As we said earlier, the ASP.NET Web Role project created in Visual Studio is just a normal ASP.NET Web Application with 3 extra assembly references. So if you need to migrate your existing application (or use it in another project), you can always just add those extra assemblies via the Add Reference dialog. The assemblies that you need to reference can be found at the following location:

```
c:\Program Files\Windows Azure SDK\v1.0\ref\
```

Now that we know how to reference the Service Runtime Assembly, let's now take a look at some of the API calls and how we can use them.

4.1.2 Is our application running in Windows Azure?

One of the static properties that the service runtime exposes via the **RoleEnvironment** static class is the ability to check if your application is running within Windows Azure. This check can performed using the following property:

```
RoleEnvironment.IsAvailable
```

To see this in action, we will now create a very quick web page that will display a label that states if your application is running in Windows Azure. Figure 4.2 shows our web page running in the Development Fabric.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 4.2 - A funky little web page that tells us that we are running in the Fabric

To create the web page shown in Figure 4.2, you should create a new Cloud Service Solution called "ServiceRuntime" with a new ASP.NET Web Role called "ServiceRuntimeWeb". In the web project you should now modify the Default.aspx page to include the following label:

```
<asp:label id="runningInTheFabricLabel" runat="server"/>
```

Once you have added the above label to your web page, you can then display the result of the RoleEnvironment.IsAvailable call in the contents of the label by adding the following code to the Page_Load event.

```
runningInTheFabricLabel.Text = RoleEnvironment.IsAvailable ?
                  "I am running in the fabric" : "Not in the fabric";
```

As mentioned earlier in the book, Web Roles are just standard ASP.NET Web Applications so they can still be run on standard IIS Web Server (if you like retro computing). If you therefore launched this page in IIS (or the ASP.NET Web Development Server) then it would display the "Not in the fabric" message. If you were to now fire up your web page in the Windows Azure Development Fabric it would display "I am running in the fabric" (as shown in Figure 4.2).

RoleEnvironment.IsAvailable is not only useful for announcing to the world that your web application is in heaven (I mean the cloud), but is also useful when building applications (or libraries) that will be hosted both inside and outside of Windows Azure. Since most Windows Azure API's are not available outside of Windows Azure you may wish to check if you are running inside Windows Azure before using said API. Later on this section we will look at some of these situations and possible patterns that you can use when working with shared code.

Now that we have introduced you to the Service Runtime Assembly, let's now take a look at some of the other differences between a standard ASP.NET Web Application, and a Windows Azure ASP.NET Web Role.

4.2 Defining your service

In chapter 3, we spent quite a bit of time discussing the infrastructure and architecture of Windows Azure. In that chapter we introduced the concept of the Service Model and how it comprised of 3 elements:

- The Service Definition
- The Service Configuration
- The Operating Model

In this section we will look at the first piece of the Service Model puzzle, which is the Service Definition file (we will look at some of the other stuff later on). In chapter 3 we described how the Fabric Controller uses the Service Definition file to manage your service; in this chapter we will show you how you can effectively define your service.

The sort of information that the Service Definition file holds includes:

- Number of required upgrade domains (see chapter 3)
- Endpoint of our Web Service (port and protocol)
- Does the role run in Partial or Full Trust?
- Does the role have any configuration settings?
- How much local disk space does the role required for local file storage?
- Size of the VM required?

In the following subsections we will take a look at how we can define some of that information. Before we do that, let's return to looking at the Service Definition file itself.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

4.2.1 The format of the Service Definition file

Earlier on when we created our Cloud Service project, the Service Definition file was automatically added to our project. Listing 4.1 shows the service definition file for the ServiceRuntimeWeb project that we created.

Listing 4.1 Service Definition file of a Web Role

```
<ServiceDefinition name="ServiceRuntimeWeb"                          #A
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
  <WebRole name="ServiceRuntimeWebsite">                             #B
    <InputEndpoints>
      <InputEndpoint name="HttpIn" protocol="http" port="80" />      #1
    </InputEndpoints>
    <ConfigurationSettings>                                          #2
      <Setting name="DiagnosticsConnectionString" />
    </ConfigurationSettings>
  </WebRole>
</ServiceDefinition>
```

#A - Same name as our Cloud Project
#B - Same name as our Web Role

As you can probably see from the Listing 4.1, the Service Definition file adheres to the following format.

- Cloud Project (ServiceDefinition element)
  - Role Definition (Web Role)
    - Input Endpoints (#1)
    - Internal Endpoints (not shown in Figure 4.1)
    - Configuration Settings (#2)
    - Certificates (not shown in Figure 4.1)
    - Local Storage (not shown in Figure 4.1)

Throughout the course of the next few sections we will explore the items that define your role in more detail.

Since our Cloud Service project only contains a single Web Role, we can therefore only see a single role definition our Service Definition file. If our project contained multiple roles then these would be reflected in the file also. The following XML shows how this would be structured in the Service Definition file:

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition name="ServiceRuntimeWeb"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
  <WebRole name="ServiceRuntimeWebsite">
      …
  </WebRole>
  <WorkerRole name="WorkerRole1">
      …
  </WorkerRole>
</ServiceDefinition>
```

As we can see from the XML above, we now have two roles in this project. A Web Role called "ServiceRuntimeWebsite" and a Worker Role called "WorkerRole1" (please note that the definition of a worker role is exactly the same as the configuration of a web role, except the definition element is called WorkerRole instead of WebRole).

Now that we have an idea of the structure of the Service Definition file, let's look in a little more detail at the sort of information that we would describe about our service.

4.2.2 Configuring the endpoint of your web role

If we return to the Service Definition file for our web role as shown in Listing 4.1 at #1 we defined the http port and protocol that our application runs on.

Windows Azure only allows web roles to receive incoming http (via port 80) or https (via port 443) messages via your virtual IP address (see chapter 3). Any other traffic that is sent to your virtual IP address will either be

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

filtered out by the firewalls or not forwarded to your web role from the load balancer. Figure 4.3 shows how the interaction between a client browser, the load balancer and your web role.

Figure 4.3 - The load balancer protects the galaxy (or at least your web role) from the threat of invasion

By locking down the available ports and protocols, Windows Azure reduces the surface area of attack for your web role. Any incoming requests to your web role (outside of that port and protocol combination) will be filtered out by the firewalls and load balancers meaning that the request will never reach the servers that your web role is hosted on. This level of protection protects your web role from port attacks as well as distributed denial of service (DDOS) attacks.

By default Visual Studio will correctly configure your web role to use http and port 80, we can see this has happened at #2 (Listing 4.2) in our Service Definition file:

```
<InputEndpoint name="HttpIn" protocol="http" port="80" />
```

If we wished to expose our service via https then we could change the InputEndpoint at #2 (Listing 4.2) in the Service Definition file to:

```
<InputEndpoint name="HttpsIn" protocol="https" port="443" />
```

Finally, if you need to run your application in both http and https, you can define this by doing the following:

```
<InputEndpoints>
   <InputEndpoint name="HttpIn" protocol="http" port="80"/>
   <InputEndpoint name="HttpsIn" protocol="https" port="443"/>
</InputEndpoints>
```

I HATE XML GIVE ME A GUI YOU FREAK

If you are currently thinking to yourself, oh god, I'll never remember all the XML syntax then good news, the Service Definition file does have an XSD Schema associated with it. This means that you get full intellisense when editing this file in Visual Studio. If you edit in notepad you get unintellisense ☺

Alternatively, if you feel that we have moved beyond text files and are in a WPF Minority Report style GUI interface era then you will be pleased to hear that you can edit the Service Definition file with a pretty little dialog box (well a dialog box at least). To access the editor just double click on the name of your web role in the Roles folder of your Cloud Service project (e.g. Double Click ServiceRuntimeWebsite in the roles folder in the ServiceRuntimeWeb cloud project, see figure 4.1).

To modify the endpoints within the editor just select the "Endpoints" tab as shown in Figure 4.4.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 4.4 - Using a pretty little GUI to configure endpoints cos XML is too scary ;)

After looking at Figure 4.4, you might be thinking "hang on buddy boy, why are the port textboxes editable if we can only use port 80 for https, or port 443 for SSL?"

IGNORING BEST PRACTICES WHEN DEVELOPING

Although security best practices are great for production servers, they can be a right pain in the left toe during development. We can all thank the Ozzie almighty for making this a little easier in the Development Fabric by allowing us to run our web role on any port.

As you can see from Figure 4.4 we can easily set this value via the editor, however since I have an XML fetish, let's look at how it's done in the Service Definition file. The following bit of configuration shows how we could run our web role on port 87 over http in the development fabric.

```
<InputEndpoint name="HttpIn" protocol="http" port="87" />
```

The Development Fabric is pretty lenient when it comes to configuring available ports. If a port is already taken (such as IIS hogging port 80) then it will just give you the next available port. For example if you asked for port 80, then it would just fire your application up on port 81 instead.

INTRA ROLE COMMUNICATION

You may have noticed in Figure 4.4 that there is a little section called "Internal Endpoint" (go on have a look, if you didn't see it already). Basically if you need to host a web role internally (i.e. you don't want that web role to be available outside of the Windows Azure Fabric) but you wish to make that web role available to other role, then this is checkbox for you. A typical reason for wanting this functionality maybe a Web Role or Worker Role that is reliant on an internal web service (Service Oriented Architecture type applications).

You can configure the internal endpoint of this role by setting the name, port and protocol of the internal endpoint as shown below.

```
<InternalEndpoint name="MyInternalRole" protocol="http"/>
```

Alternatively you can use the editor, as shown in Figure 4.4

It is worth pointing out that Web Roles only support http internal endpoints and cannot be secured with certificates. It's also noting that you should really only use the http protocol for internal endpoints that are legacy web services (old ASMX services etc). If you are considering using this approach for WCF Services then

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

you should really host your service with a worker role and expose it via TCP instead. TCP generally provides better performance for internal services than http.

WORKER ROLE ENDPOINTS

Although the Service Definition file is a common file that is used by both web roles and worker roles, we are only going to look at web roles in this chapter. Therefore we will leave the examples on how to host worker roles that accept incoming requests across various protocols and ports to later in the book.

However, it is worth pointing out that configuration of a worker role uses the same InputEndpoint tag as the web role. The following XML shows the Endpoint for a worker role hosted on TCP Port 10000:

```
<InputEndpoint name="MyEndpoint" protocol="tcp" port="10000" />
```

Figure 4.5 shows the GUI for configuring a worker role endpoint:

Figure 4.5 - Setting a worker role to be hosted via TCP on Port 10000 to the external world

The editor shown in Figure 4.5 can also be used to set internal endpoints. Please note internal endpoints are dynamically assigned ports and you cannot manually set the port number. The following XML defines an internal endpoint called MyEndpoint that uses TCP (rather than http or https) as a protocol:

```
<InternalEndpoint name="MyEndpoint" protocol="tcp" />
```

As we said earlier, we only intended to give a very brief description on how this configuration applied to worker roles. So we will move on for just now and return to this, later on.

Now that you've seen how to configure endpoints and you've seen the Windows Azure Role Editor, it's worth looking at some of the other tabs in the editor (as shown in Figure 4.5).

4.2.3 The Configuration Tab

So we've looked at the Endpoints tab already. So let's take a look at some of the tabs, starting with the configuration tab as shown in Figure 4.6.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 4.6- The configuration tab for your role in Visual Studio

Later on in the book, we will be looking at these sections in more detail but for the moment we will look at following sections (from Figure 4.6), very briefly:

- .NET trust level
- Instances
- Startup action

.NET TRUST LEVEL

As you can see from Figure 4.6, Windows Azure supports two levels of trust (Full trust & Partial trust).

Partial trust is very similar to ASP.NET's medium trust level and therefore restricts the operations that your application can perform (it doesn't trust it enough ☺). Wherever possible you should attempt to run your application in partial trust mode as it provides a greater level of security (I sleep like a baby at night whenever my application is running under partial trust).

If however you need to perform big scary actions that would make a security freak's skin crawl (such as C++, Reflection, P/Invoke) then you will need to set your application to Full Trust.

Using the pretty editor shown in Figure 4.6 is probably the easiest way of setting your trust level but since we all love messing with configuration files; you can of course modify the Service Definition file directly.

If you need to configure partial trust then you would set the **enableNativeExecution** attribute on your role to false. For full trust you can either set the attribute to true or not configure it at all (full trust is the default level). The following Xml shows how we could set our earlier web role to partial trust:

```
<WebRole name="ServiceRuntimeWebsite"
           enableNativeExecution="false">
```

In chapter 6, we will look in more detail at the supported trusted levels.

INSTANCES

Let's return to Figure 4.6 for a second to look at the Instances section. This section allows us to perform the following 2 actions:

- Set the number of instances of your role
- Set the size of your VM (Small, Medium, Large, Extra Large)

In the "Scaling Web Roles" chapter we will look at these settings in more detail.

STARTUP ACTION

The final section that we will look at in the configuration tab (Figure 4.8) is "Startup Actions". This section isn't really part of the Service Definition but is instead a wee bit of Visual Studio configuration. As you can see in Figure 4.8 there are 2 checkboxes:

- Launch Browser for http endpoint
- Launch Browser for https endpoint

As you might have already figured, these checkbox just tell Visual Studio which endpoints you wish to launch in the development environment when you hit F5.

Now that we have finished looking at the configuration tab, let's see what other tabs in the Visual Studio editor are left to look at (see Figure 4.4).

4.2.4 Local Storage

We have so far looked at both the Configuration & the Endpoints tab in the Role Editor. It kinda makes sense to keep looking at some of the tabs, so we will now take the time to explore the Local Storage tab.

Before we start looking at the tab, let's take a little time to work out what Local Storage is.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

WHAT IS LOCAL STORAGE

Local Storage is simply a **temporary** file system storage area that is made available to a **role instance** to store and retrieve data locally. Did you notice that we highlighted the words temporary and role instance?

The first key point about local storage is that this storage area is only available to a single role instance and cannot be shared across multiple role instances.

The second key point is, if the Virtual Machine for your Role Instance dies and never recovers then you will lose the data stored in this area forever. Therefore you should only ever store volatile data in this storage area and never store any data that you may later rely up on in a court of law. Any data that you store in Local Storage should also be stored in a non-volatile storage area such as Blobs, Table Storage or SQL Azure.

IF IT'S SO VOLATILE, WHY DO I NEED IT?

Although Blobs, Table Storage or SQL Azure can be accessed by all instances of a role, the price of centralized storage mechanisms comes at a cost, that cost is Latency. Quite simply Blobs, Table Storage and SQL Azure are all hosted on separate servers in another part of the data center whereas Local Storage is part of your VM (i.e. the disks are hosted on the same server as your VM). This means that Local Storage is much faster than the other storage mechanisms.

Although Blobs, Table Storage and SQL Azure will be suitable for most scenarios, if you are processing high volumes of data then you may wish to use Local Storage to temporarily cache that data.

Now that we know what Local Storage is and why we need it, let's look at how we set it up?

SETTING UP LOCAL STORAGE

As always there are two ways to configure something. We can either do it manually via the Service Definition file, or we can use the Role Editor and let it modify the Service Definition file for us. Figure 4.7 shows the GUI for configuring Local Storage in Visual Studio.

Figure 4.7 - Configuring some temporary storage space in your role

As we can see from Figure 4.7, when defining a Local Storage Resource you can define the following 3 items:

- The name of the resource (MyStorage)
- The amount of space required in Megabytes (100MB)
- Whether we want the temporary data deleted when the role is recycled

It's worth pointing out that maximum amount of Local Storage are that you can use for a single role instance is 20GB. If you need more than 20GB temporary storage space then you may wish to relook at the architecture of your application ;)

Now that we have defined our Local Storage in Visual Studio, let's see how this is reflected in the Service Definition file:

```
<WebRole name="ServiceRuntimeWebsite">
   <LocalResources>
      <LocalStorage name="MyStorage"
                    cleanOnRoleRecycle="true"
                    sizeInMB="100" />
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
        <LocalStorage name="MoreStorage"
                      cleanOnRoleRecycle="false"
                      sizeInMB="50" />
    </LocalResources>
  </WebRole>
```

A key point that we really wish to spell out is that this lives in the Service Definition file not the Service Configuration file, so if you incorrectly size your temporary storage area and need to request a larger size then it will require a redeployment of your application.

In the current version of Windows Azure you cannot dynamically change this setting at runtime hence why this setting lives in the Service Definition file and not in the Service Configuration file. So why is this case?

The Fabric Controller will use the requested size of Local Storage as part of its algorithm to decide which physical servers will host your virtual machine. If the Fabric Controller allowed you to modify the Local Storage dynamically without a redeployment then it may not be able to satisfy your request with the servers currently hosting your role instances (there maybe too many other roles hosted on that server with very high Local Storage Requirements). By forcing a redeployment of the application, the Fabric Controller can safely redeploy your role instances to the appropriate servers satisfying your request.

Now that we have seen how to setup local storage, let's take a quick look at how to use it in our application.

USING LOCAL STORAGE

If we need to be able to use Local Storage in our role (web or worker) then we can just make a request to retrieve information about our role using the **RoleEnvironment** class in the Service Runtime. The following call will allow us to request information about our Local Storage resource.

```
LocalResource myStorage =
                   RoleEnvironment.GetLocalResource("myStorage");
```

As we can see from the code above we just need to call **RoleEnvironment.GetLocalResource** passing in the name of the Local Storage Resource that we defined earlier (myStorage). The object returned by GetLocalResource exposes three properties (Name, MaximumSizeInMegabytes and RootPath). The Name and MaximumSizeInMegabytes properties return the information that we set in the Service Definition file as shown below:

```
string name = myStorage.Name;
int maxSizeInMB = myStorage.MaximumSizeInMegabytes;
```

The RootPath property returns the physical path of the folder where you temporary storage area has been assigned. Using the RootPath property you can use the standard .NET methods to store and retrieve data in this folder. The following code will create a text file called "HelloWorld.txt" which will contain the text, "Goodbye World".

```
System.IO.File.WriteAllText(myStorage.RootPath + "HelloWorld.txt",
                            "Goodbye World");
```

As you can see it's pretty simple to use Local Storage as it's all built upon all the existing System.IO classes that we have come to know and love.

Before we leave the subject of Local Storage, I want to cover one final thing which is regarding the "Clean on Role Recycle" setting.

CLEAN UP ON RECYCLE

We saw earlier that we could set our Local Storage Resource to be "Cleaned on Role Recycle". This setting is pretty self-explanatory really, it basically means if your role gets killed do you want to lose or keep your Local Storage data.

Recycling of roles can happen in a few circumstances i.e.:

- Upgrades (you deploy a new version, or an OS patch is applied)
- Faults (server dies)

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

- Manual (you request the role to be recycled)

If you need to test how your application responds to losing your temporary data then it's pretty hard to test this as part of an upgrade or a fault but you can do this manually. Let's look at how this can be done

RECYCLING A ROLE

Recycling a role within our code is really pretty easy. All we need to do is call the RecycleRole method in RoleEnvironment class as shown below

```
RoleEnvironment.RecycleRole();
```

Using the above code will not only allow you to test that your application handles Local Storage correctly on your role instance being recycled but it will also allow you to test that the rest of your application behaves gracefully.

If your application needs to know when a Role Instance is stopping (cleaning up resources, notifying a monitor etc) then you can always use the RoleStopping event, e.g.:

```
protected void Page_Load(object sender, EventArgs e)
{
   RoleEnvironment.Stopping +=
       new EventHandler<RoleEnvironmentStoppingEventArgs>
       (RoleEnvironment_Stopping);
}

void RoleEnvironment_Stopping(object sender,
                        RoleEnvironmentStoppingEventArgs e)
{throw new NotImplementedException();}
```

As you can see from the code above you can easily stick any cleanup code that you may need in the event handler for the RoleEnvironment.Stopping event.

Wow, I think it's fair to say we pretty much know almost everything we need to know about Local Storage. Later on in the book we will revisit this topic in the Blobs chapters to show you how you can use this in combination with Blob and how Local Storage can help you when building massively scalable worker processes built upon MapReduce.

Now that we understand Local Storage its worth relooking at what tabs we have left in the Role Editor. If we look at Figure 4.7, we can see that we have so far looked at:

- Configuration
- Endpoints
- Local Storage

This really leaves us certificates and settings to look at. When we looked at configuring endpoints, we discussed https but we didn't mention how to configure the SSL certificate for your site. Let's return to that subject and look at certificate management in Windows Azure.

4.3 Setting up certificates in Windows Azure

There are really 3 parts to the subject that we really want to look at:

- How to add certificates
- How to generate certificates
- How to configure certificates

Before we can really add a certificate to Windows Azure we really need a certificate to add, so let's look at how this is done.

4.3.1 Generating a certificate

For the live production applications you should use a purchased certificate from a trusted authority. If you are just experimenting or testing an application you can of course use a test certificate. Since you have already

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

shelled out for this lovely book, we will save your wallet from more troubles and show you how to generate a test certificate (that you can use both on the production or development fabric).

To generate a test X.509 certificate you can use a tool called makecert, which ships with both Visual Studio and the .NET Framework. To start using the tool you should fire up an instance of the Visual Studio Command Prompt as an Administrator. Using the command prompt you can generate a test certificate using the following command:

```
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 MyCertificate.cer
```

The command above will generate a test X.509 certificate called MyCertficate.cer which is suitable for use within Windows Azure, and store it the CurrentUser/Personal store. For further details on the makecert tool, you can always visit the following URL:

http://msdn.microsoft.com/en-us/library/bfsktky3(VS.80).aspx

With our brand new certificate in hand, we can now look at how we will install said certificate in both the production system and the development fabric.

4.3.2 Adding Certificates

The production fabric and the development fabric both have different methods of managing certificates. Let's look at how to do this on the live system first.

Adding Certificates to Production Fabric

As you might expect certificates are managed via the Azure Portal. Figure 4.8 shows how you can upload a certificate via the "Manage Certificates" page.

Figure 4.8 - Clicking buttons on a web page to install a certificate

If you need to install your certificate through your own code then you can of course the management api's to perform this. We won't cover how to do this in this book as it's frankly quite dull and I need to get back to watching American Idol (that's how dull it is). Actually, the real reason we're not covering it, is that it's not a typical scenario for most folks. If you are automatically installing lots of websites then this feature could be useful, the rest of us should just use the portal.

Adding Certificates to the Development Fabric

If you need to test https in your development Fabric then you will need the appropriate certificate on your development machine. To upload your certificate into the Development Fabric you should use the certificates tab in our role editor, as shown in Figure 4.9.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 4.9 - Adding certificates to the development fabric

As you can see from Figure 4.9 you can set the name, location (LocalMachine, CurrentUser), Store Name (My, Root, CA, Trust) and the certificate.

To set the certificate you can either enter the thumbnail manually or select a certificate from your personal store. If you click on the Thumprint button (the one with the two dots), it should bring up the select certificate dialog (armed with the test certificate that we generated earlier), as shown in Figure 4.10.

Figure 4.10 - The select certificate dialog should show the certificate we generated earlier with makecert

Now that we have selected our certificate you may be wondering how this is reflected in the Service Definition file? Don't worry your XML obsessed author is here to help you out. The following XML shows how the certificate is represented:

```
<WebRole name="ServiceRuntimeWebsite">
    <InputEndpoints>
      <InputEndpoint name="HttpsIn" protocol="https" port="443" />
    </InputEndpoints>
    <Certificates>
      <Certificate name="MyCertificate"                          #A
         storeLocation="LocalMachine"                                  #B
         storeName="My" />                                             #C
    </Certificates>
</WebRole>
```

#A - Same name we chose in Figure 4.9
#B - Same store location we chose in Figure 4.9
#C - Same store name we chose in Figure 4.9

As we can see from the example above, everything that we set in Figure 4.9 is present in the Service Definition file except the Thumbnail. The reason the thumbnail isn't present is because the thumbnail is a

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

configurable setting not a definable attribute of the service. For the moment, we won't show you where the thumbnail is stored and will instead return to this a little later.

4.3.3 Configuring your https endpoint to use the certificate

Now that we know how to generate and install our certificates, we now need to look at how we configure our endpoint to use the new certificate.

With our certificate installed, we can either use SSL Certificate dropdown as shown in Figure 4.4, which will now be populated with our test certificate, or we can manually configure it in the Service Definition file. To configure the endpoint manually, we just need to set the certificate attribute of InputEndpoint element to the name of our certificate, as shown below:

```
<InputEndpoint name="HttpsIn" protocol="https" port="443" certificate="MyCertificate"/>
```

WORKER ROLE CERTIFICATES

If we do expose a worker role externally using an InputEndpoint, we can also secure that service with a certificate. Certificate management and Configuration is the exact same process for a worker role as it is for a web role.

Now that we have looked at how you define the endpoints that your role can run on, let's look at some of the other things that we can define about our service.

4.4 Summary

So it's probably a good time to stop and review where we are. In this chapter we have taken everything that we have learned about how Windows Azure works (from chapter 3) and we've started to define how we want our application run in the environment.

We've looked at the Service Definition file and we can use it to instruct the Fabric Controller on how to manage our application.

In the next chapter we'll take what we'll follow on from defining our service and look at how we can configure it work in Windows Azure.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

5

Configuring your service

This chapter covers:

- Understanding the configuration file
- Handling configuration at runtime
- How we handle non app settings based configuration
- Patterns for sharing configuration between Azure and Non Azure applications

In the last chapter, we spent our time concentrating on how we define our role using the Service Definition file. We will now look at the second part of the Service Model picture, which is the Service Configuration file.

5.1 Working with the Service Configuration File

As we discovered earlier the Service Definition describes our Role and is used by the Fabric Controller to effectively manage our Role. If any of the settings within our Service Definition file needs to change then we will also need to redeploy our role.

Not all settings however need a redeployment of our Role to be applied. We may even wish to change some our settings dynamically without even restarting the Role (surely not we develop on Microsoft products, we love a good restart). These dynamic settings are typically stored in the Service Configuration file (ServiceConfiguration.cscfg).

The Service Configuration really provides two purposes:

- Allow you to dynamically configure standard Windows Azure runtime settings
    - Number of instances of your Role
    - Certificate Thumbprint
- Allow you to dynamically configure your own custom settings

We will now look at you can perform both of these actions, and then finally look at how you can dynamically modify these settings at runtime.

5.1.1 The Format of the Service Configuration File

Let's now take a look at the standard Service Configuration file that will be created by Visual Studio when you create a Web Role. Listing 4.2 shows the Service Configuration file that was generated when we created our ServiceRuntimeWeb cloud project at the beginning of this chapter.

Listing 4.2 - The Service Configuration file for our ServiceRuntimeWeb cloud project

```
<?xml version="1.0"?>
<ServiceConfiguration serviceName="ServiceRuntimeWeb"                    #A
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration">
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
  <Role name="ServiceRuntimeWebsite">                                    #B
    <Instances count="1"/>
    <ConfigurationSettings>
      <Setting name="DiagnosticsConnectionString" value="UseDevelopmentStorage=true"/>
    </ConfigurationSettings>
  </Role>
</ServiceConfiguration>
```

#A - Same name as our Cloud Project
#B - Same name as our Web Role

Before you starting thinking, "oh blooming heck, not another xml file", don't worry everything you in this file can be configured in the Visual Studio editor. It is still useful to understand the structure of this file, as it allows you to understand which settings are dynamic. Also if things go wrong then you may need to recover the file with notepad.

As you can probably see from the Listing 4.2, the Service Configuration file adheres to the following format.

- Cloud Project (ServiceConfiguration element)
    - Role Definition
        - Instances
        - Configuration Settings
        - Certificates

Since our Cloud Service project only contains a single Web Role, we can therefore only see a single role definition our service configuration file. If our project contained multiple roles then these would be reflected in the file also. The following XML shows how this would be structured in the Service Configuration file:

```
<ServiceConfiguration serviceName="ServiceRuntimeWeb"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration">
  <Role name="ServiceRuntimeWebsite">
      …
  </Role>
  <Role name="WorkerRole1">
      …
  </Role>
</ServiceConfiguration>
```

As we can see from the XML above, we now have two roles in this project. A Web Role called "ServiceRuntimeWebsite" and a Worker Role called "WorkerRole1" (please note that the configuration of a worker role is exactly the same as the configuration of a web role).

Now that you have a pretty good idea now of how the Service Configuration file is structured, let's look at how we can configure some of the standard settings.

5.1.2 Configuring Standard Settings

As of the PDC09 release there are really only 2 types of standard settings that you can configure within Windows Azure:

- Number of Instances of your Role
- Certificate Thumbprint

All other settings are custom settings (which we will look at in the next section).

NUMBER OF INSTANCES

We are now going to briefly look at the most important piece of configuration in Windows Azure. The following XML is used to define the number of instances of your role:

```
<Role name="ServiceRuntimeWebsite">
    <Instances count="5"/>
</Role>
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

As you can see from the above XML, we require 5 instances of our Web Role. If we need to either increase the number of instances required for our web role, we just need to modify this value in our Service Configuration. During development, we can also modify this value using Configuration tab.

If you're currently wondering the reason why this is in the Service Configuration file and not in the Service Definition file, then the answer is quite simple. Basically this setting doesn't really define the service (it hasn't changed), we just want more instances of it. By storing this setting in the Service Configuration file, we can scale the number of services up and down dynamically without us having to redeploy our application (after all no code has actually changed). Later on in this section we will look at how we can modify the Service Configuration file at runtime via the Azure Portal.

For just now let's return to a topic that we had parked earlier.

CERTIFICATES

Now that you have an understanding of the Service Configuration file, it's worth returning to the example we built earlier in the certificates section. To recap, in that section we showed you how to:

- Generate a Test Certificate
- Install a Certificate into the Development Fabric and Production Fabric
- Associate a certificate with our https endpoint

When we installed our certificate in the Development Fabric earlier, we saw that it stored the following XML in our Service Definition:

```
<WebRole name="ServiceRuntimeWebsite">
    <InputEndpoints>
      <InputEndpoint name="HttpsIn" protocol="https" port="443" />
    </InputEndpoints>
    <Certificates>
      <Certificate name="MyCertificate"
         storeLocation="LocalMachine"
         storeName="My" />
    </Certificates>
</WebRole>
```

If we return to the Certificates editor in Visual Studio (as shown in Figure 4.9), we can see that the thumbprint shown in this dialog, is not present in the Service Definition file. The reason that it is not present is that this value is instead stored in the Service Configuration file as shown below:

```
<?xml version="1.0"?>
<ServiceConfiguration serviceName="ServiceRuntimeWeb"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration">
  <Role name="ServiceRuntimeWebsite">
    <Instances count="1"/>
    <Certificates>
      <Certificate name="Certificate1" thumbprint="E6AE81BB1E818D04BE3EBBE09E8A4B4EB42D5B73"
thumbprintAlgorithm="sha1" />
    </Certificates>
  </Role>
</ServiceConfiguration>
```

So why is this split across 2 configuration files (Service Definition and Service Configuration)? Well let's think about the intention of this functionality. The definition of our role is, the Web Role ServiceRunTimeWebsite is hosted on port 443 using the protocol https and requires a certificate that we have uploaded called MyCertificate.

In reality certificates are not a fixed entity. We may wish to use a test certificate in the staging environment but only use our production certificate in the production environment. Certificates expire, so we may need to change our certificate through time. Therefore the actual certificate associated with our service is dynamic and configurable hence why the thumbprint lives in the Service Configuration file. Do we really want to redeploy our application to change certificates when we switch our application from staging to production?

So what is the thumbprint used for. Well quite simply the name of the certificate in the Service Definition file (MyCertificate, the one you typed in Visual Studio) is internal to Windows Azure. This name is not tied back to

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

the name of the certificate that you generated earlier. Therefore we need a method of retrieving the correct certificate from the store, the thumbprint is that search parameter. Windows Azure uses the FindByThumbprint functionality built into Windows to retrieve the actual certificate.

If you therefore have a production certificate from a trusted authority that you cannot install in your development environment and you need to configure that in Windows Azure via the portal then you can always manually configure the certificate in the Service Configuration file using the Thumbprint.

Now that we have understand some of the standard configuration settings, let's look at how we configure our own custom settings.

5.1.3 Configuring runtime settings

Like any other application, when working with web or worker roles we need to be able to dynamically configure runtime settings without rebuilding the application from its source. In normal web applications, typical settings that are stored in configuration files may include:

- Database connection strings
- Service endpoints
- File system paths
- Timeout settings
- Etc.

Although these configuration settings are considered dynamic, in most applications they are rarely changed. The main reason for storing these runtime configuration settings in a configuration file is to allow you to easily deploy your application between different environments (e.g. development, test, staging and production). Typically your production environment will talk to different web services, a different database or different storage account from your development or staging environment. Using configuration files (rather than rebuilding your source code) just to modify these endpoints greatly reduces the complexity, maintainability and deployment process of your application.

In this section we will look at the following:

- How we would normally configure runtime settings in normal web apps
- Define our runtime configuration settings in Windows Azure
- Initially configure our runtime configuration settings
- Read our configuration settings
- Modify our configuration settings dynamically at runtime

Before we look at how we define our runtime configuration settings in Windows Azure, let's look at how we would do this in normal ASP.NET Web Applications.

CONFIGURING RUNTIME SETTINGS IN NORMAL ASP.NET WEB APPLICATIONS

In normal ASP.NET web applications application you will typically store any configurable runtime settings in the web.config. Windows Azure still provides the ability to read application settings from the web.config (as we will now demonstrate) but this is not the method that you should use to read the configuration settings.

We will now build a small web page that will read and a display a setting from the appSettings section of the web.config. We will modify this page in future sections to use the Windows Azure configuration settings functionality. Figure 5.1 shows the output of the web page that we will create. The text "Hello Birds Hello Trees" is read from the web.config.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 5.1 - Displaying Configuration Settings in a web page

In your ASP.NET Web Project you should now add a new ASP.NET Web Page called ConfigurationSettings.aspx that we will use to develop the page shown in Figure 5.1. In this page you should add the following label:

```
<asp:label id="mySettingLbl" runat="server"/>
```

The text "Hello Birds Hello Trees" that we will display in the label "`mySettingLbl`" will be stored in the appSettings section of the web.config. In the web.config file of your web application you should replace the appSettings tag with the following:

```
<appSettings>
   <add key="mySetting" value="Hello Birds Hello Trees"/>
</appSettings>
```

On the Page_Load event of the ConfigurationSettings.aspx page you should add the following code to display the contents of the "mySetting" application setting:

```
mySettingLbl.Text = ConfigurationManager.AppSettings["mySetting"];
```

If you were to now run this application in the Web Development Server, IIS, the Development Fabric, or on the live Windows Azure production Fabric, the application would run correctly and display the page shown in Figure 5.1.

ISSUES WITH USING WEB.CONFIG

Although Windows Azure can read anything stored in the appSettings section of your web.config, it does not provide any ability to modify these settings at runtime. If you need to modify a value in your web.config file then Windows Azure would require you to redeploy your entire application.

In typical web applications you would modify a web.config individually if you needed to change runtime settings. Unfortunately this approach is not scalable beyond a single server. If you need to modify 100 instances of a web application then dealing with each individual web.config is likely to be slow and cause synchronization issues. In such a scenario you will need to store the configuration settings centrally and then distribute the changes out to each instance. It therefore makes sense to separate out the configuration settings from the web.config (after all the web.config doesn't just store application settings) and provide a new mechanism to feed runtime settings.

Let's now take a look at how Windows Azure has implemented its own configuration settings system.

SETTING CONFIGURATION IN THE SERVICE DEFINITION

Any configuration settings that our application uses must be first defined in the Service Definition file (ServiceDefinition.csdef).

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In the service definition file we need to define a new setting that will be used to display the message ("Hello Birds") in our web page. To do this we need to add the name of the new setting (mySetting) to the ConfigurationSettings section of the service definition file. The following configuration shows how the ConfigurationSettings section should now look:

```
<ConfigurationSettings>
   <Setting name="mySetting"/>
</ConfigurationSettings>
```

The service definition file should now look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition name="ServiceRuntimeWeb"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
  <WebRole name="ServiceRuntimeWebsite">
    <InputEndpoints>
      <InputEndpoint name="HttpIn" protocol="http" port="80" />
    </InputEndpoints>
    <ConfigurationSettings>
      <Setting name="mySetting"/>
    </ConfigurationSettings>
  </WebRole>
</ServiceDefinition>
```

Notice that in the Service Definition we only specify the name of the setting; we do not set the configured value at this stage.

Setting our Configuration Value in the Service Configuration file

After defining the configuration settings that will be used by our application in the service definition file, we now need to set the actual value of the setting that will be used by the website. The configuration settings are stored within the ServiceConfiguration.cscfg file in your cloud service project.

To set our runtime configuration setting, we can simply take a copy of the configuration setting from the service definition file and paste it into the service configuration file. To set the configuration setting, we just add a new attribute called "value" which we can set to our default runtime value. The following shows how our new setting should look:

```
<Setting name="mySetting" value="Hello Birds Hello Trees"/>
```

This means our ServiceConfiguration.cscfg file should now look like this:

```
<?xml version="1.0"?>
<ServiceConfiguration serviceName="ServiceRuntimeWeb"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration">
  <Role name="ServiceRuntimeWebsite">
    <Instances count="1"/>
    <ConfigurationSettings>
      <Setting name="mySetting" value="Hello Birds Hello Trees"/>
    </ConfigurationSettings>
  </Role>
</ServiceConfiguration>
```

Reading the configuration setting

Now that we have setup our Service Definition and Configuration Settings files to contain our runtime configuration settings, we need to modify our application to use the Windows Azure Service Runtime to read the configuration data.

We can access the runtime values of the configuration setting by using the **RoleEnvironment** static class that we used earlier. To retrieve the value of a configuration setting we can use the following method:

```
RoleEnvironment.GetConfigurationSettingValueValue(SettingName);
```

To modify our existing web page to read the configuration setting from the Windows Azure Configuration Settings file rather than the web.config, we should change the code-behind of our website to the following:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
mySettingLbl.Text =
        RoleEnvironment. GetConfigurationSettingValueValue("mySetting");
```

You should now be able to run the cloud service project to see the value displayed in your web page as shown in Figure 4.11.

5.2 Handling Configuration at Runtime

Now that we understand the Service Configuration file and how to read those configuration settings, it's worth looking at how modify these configuration settings at runtime and how we track when a change occurs.

5.2.1 Modifying configuration settings in the Azure Portal

Now that we can set and read values from the Service Configuration file, let's look at how we can modify those values dynamically at runtime using the Azure Portal.

I'm not going to insult your intelligence and show you how to deploy your application again. If you need a little help, feel free to skip back to chapter 2. Once your application is deployed and running in the Fabric, if we need to modify any of our configuration settings then we can just select either the production or staging version of the role and select the "Configure" button. You will then be redirected to a page as shown in Figure 5.2 which will allow you to modify the runtime configuration settings for the selected role.

Figure 5.2 - Modifying configuration settings in the Azure Portal

As you can see from Figure 5.2, the Azure Portal doesn't have any fancy editors and is really has just a big textbox with your service configuration file in it. Bet your glad you paid attention during XML school now ;)

The Azure Portal allows us to directly edit the contents of the configuration settings or upload a new version of the Service Configuration file. Once we apply our changes they are instantly replicated out to all instances of our web role (without requiring a restart of any of the roles).

In Figure 4.12 you can see that we have modified the configuration setting "mySetting" from "Hello Birds Hello Trees" to "Hello Birds" which results in the output shown in Figure 5.3

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 5.3 - Our web page with a configuration setting modified via the Azure Portal

Wow, as you can see modifying my configuration settings is pretty easily achieved through the Azure Portal. What is even cooler is that my Role can just pick up and run with the changes without needing a restart. The only issue with that however is that I may need have some code that has local copy of that setting and will need refreshed whenever the setting changes, let's look at how that is done.

5.2.2 How do I track if my Service Configuration has changed

In the case that a configuration setting is changed, Windows Azure will raise two events that our application can handle:

- RoleEnvironment.Changing
- RoleEnvironment.Changed

RoleEnvironment.Changing is fired prior to the settings change being applied. This allows you to be able to still read the existing setting prior to the change being applied. RoleEnvironment.Changed is fired after the settings changed has been applied.

A typical scenario where you may use RoleEnvironment.Changed is on the cleanup of a shared resource. RoleEnvironment.Changed would be used in that instance to setup the new shared resource.

To hook up and define one of the above events you can use the following code:

```
protected void Page_Load(object sender, EventArgs e)
{
   RoleEnvironment.Changing +=
       new EventHandler<RoleEnvironmentChangingEventArgs>
       (RoleEnvironment_Changing);
}

void RoleEnvironment_Changing(object sender,
                              RoleEnvironmentChangingEventArgs e)
{
   throw new NotImplementedException();
}
```

So far we've have had a good look at how we use the ConfigurationSettings functionality in Windows Azure. However the ConfigurationSettings section of the ServiceConfiguration file is really only a replacement for the data that you would normally store within the appSettings section of the web.config. So how do we handle other types of configuration data?

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

5.3 What about non app-settings config?

In this section we will take a look at when you would use the Service Configuration file and when you would use regular old app-settings to setup configuration data. Typical scenarios include:

- Database Connection Strings
- Application Build Configuration
- Tweakable Configuration
- Endpoint Configuration

5.3.1 Database Connection Strings

In the case of database connection strings (such as SQL Azure DB) you may have different development, staging and production connection strings. What options do we have in Windows Azure?

In standard ASP.NET Web Applications you would store the connection string in the connectionStrings element of the web.config. Like appSettings, Windows Azure does not provide you the ability to be able to change data stored in the connectionStrings element at runtime.

Therefore you may wish to treat a connection string like an app setting and store it in the Service Configuration. Using configuration settings allows you to modify the connection string at runtime via the Azure Portal.

5.3.2 Application Build Configuration

Some configuration data can be tweaked on an individual server via the web.config but would never be different between various deployments. In these cases this configuration should remain in the web.config and really be treated as part of the application build rather than as a runtime setting. Typical examples would include the httpModules or httpHandlers sections of the web.config.

5.3.3 Tweakable Configuration

There are some configuration settings that can be useful to tweak when an unexpected issue comes up without redeploying the application. A typical example maybe that a user may upload a file that is larger than you originally configured the web server to handle. In this case you may decide to modify the maximum http request size in the web.config to allow larger file upload. The following configuration shows this web.config setting:

```
<httpRuntime maxRequestLength="1048576"/>
```

In a normal production environment you would probably test this change on a staging environment and the rollout the change to the live environment without redeploying the entire application. Typically this change would be reflected back into source control so all future builds would have the new settings.

Since you cannot change the web.config dynamically in Windows Azure you would need to redeploy the full application in the Azure Portal. Rather than trying to reflect every possible web.config setting as a configuration setting it is advised that you just accept that to tweak such a setting would require a redeployment of your application.

In this example redeployment of the application is not really an issue as you would normally want to make this configuration change to be applied across all deployments of your application.

5.3.4 Endpoint Configuration

In this section we will discuss how you would configure our Windows Azure hosted applications to use external endpoints such as WCF Services in Windows Azure.

For example let's say we have an ASP.NET Web Page that communicates via WCF to an external WCF Service. In this case we may have a different endpoint that we communicate with between the development, production and staging systems.

The following extract from the web.config represents the WCF Configuration for a typical WCF client proxy (in this example we have omitted the binding configuration).

```
<client>
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
    <endpoint address="http://myservice.com/services/timeservice.svc"
              binding="basicHttpBinding"
              bindingConfiguration="BasicHttpBinding_ITimeService"
              contract="TimeService.ITimeService"
              name="BasicHttpBinding_ITimeService" />
</client>
```

Since Windows Azure provides us no-way of modifying this information in the Azure Portal we cannot change the endpoint between staging and production without redeploying the application. In such scenarios we should extract the information that can change dynamically e.g. (http://myservice.com/services/timeservice.svc) and store it in the Service Configuration but leave all other configuration as it is (accepting that if needs to change it will require a redeploy). We could then modify the endpoint in a WCF proxy by modifying the endpoint address on the proxy e.g.

```
var timeServiceProxy = new TimeServiceClient();

Uri timeServiceEndpointAddress =
   new Uri(GetSetting("timeServiceEndpoint"));

timeServiceProxy.Endpoint.Address =
   new EndpointAddress(timeServiceEndpointAddress,
   new UpnEndpointIdentity(""));
```

Wow, we're doing pretty well so far. We've pretty much nailed how to configure our own custom runtime settings, so it's worth stepping back a little and looking at how we can apply that knowledge.

5.4 Patterns for sharing code between Azure and Non Azure Apps

Now that we have looked at how you would configure an ASP.NET Web Application to work in Windows Azure and we've looked at what sort of things you can and can't do in Windows Azure, it's worth revisiting how we can develop an application that will work in both environments.

5.4.1 Developing a common configuration settings code base

Since we have just seen how configuration settings are dramatically different between Windows Azure and standard ASP.NET, let's look at how we could develop a unified system that will work in both environments. Typically there are 2 situations that you may wish to use a common codebase for configuration settings:

- Common Library shared across multiple projects
- 2 versions of the web application (cloud version and on-premise version)

To successfully share configuration settings across Windows Azure and Non Windows Azure hosted applications, you should abstract your configuration settings so they can be read either directly from the web / app.config or via the RoleEnvironment class. There are 2 ways of implementing this

- RoleEnvironment.IsAvailable
- Configuration Settings Inversion of Control (IOC) Container

CHECK IF WE ARE RUNNING IN THE FABRIC

Unfortunately if you attempt to access configuration setting outside of the Windows Azure Fabric using the RoleEnvironment.GetConfigurationSettingValue method then a very large nasty exception will be thrown. The reason for this is that the GetConfigurationSettingValue method is a Windows Azure only method that is not supported in standard .NET Applications.

We can get around this issue by performing a check to see if we are running within Azure using the RoleEnvironment.IsAvailable property, as shown below:

```
public static string GetSetting(string settingName)
{
   if (RoleEnvironment.IsAvailable)
   {
      return RoleEnvironment.GetConfigurationSettingValue(settingName);
   }
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
      else
      {
         return ConfigurationManager.AppSettings[settingName];
      }
   }
```

This is probably the simplest method of sharing configuration settings that span across Windows Azure and Non Windows Azure hosted applications.

In the above code,the GetSetting method simply checks if the ASP.NET Web Application is running in the fabric using the RoleEnvironment.IsAvailable property. If the application is hosted in Windows Azure (development fabric or live system) then it will use the RoleEnvironment class /to retrieve the configuration setting otherwise it will use the standard method of retrieving a setting from the appSettings.

Although the previous example is very simple, it will mean that your non Windows Azure applications will reference Windows Azure assemblies even though your application is not running within the Windows Azure environment. In this case your common libraries (and non-Windows Azure hosted versions of your applications) will need to reference the Microsoft.WindowsAzure.ServiceRuntime assembly.

PLUGGABLE CONFIGURATION SETTINGS USING INVERSION OF CONTROL (IOC)

If you wish to keep your application layers clean then you should consider using the Inversion of Control (IOC) pattern. The Inversion of Control pattern is pluggable architecture that allows you to decouple the execution of an operation from its implementation. In this case we really just want to call GetSetting in a common layer and let the application figure out the implementation to use (appsettings or RoleEnvironment) as appropriate. In the following example we will use the Unity Application Block (which is part of Microsoft Enterprise Library 4.1) to implement this pattern but you can of course use any IOC framework.

DEFINING OUR CONFIGURATION SETTINGS INTERFACE

In order to support a pluggable architecture we need an interface that both the RoleEnvironment and AppSettings versions of our ConfigurationSettings classes can access such as:

```
interface IConfigurationSettings
{
   string GetSetting(string settingName);
}
```

The interface above exposes a single method which will accept the name of the setting to retrieve and return the value of the setting.

IMPLEMENTING OUR CONFIGURATION SETTINGS INTERFACE IN STANDARD WEB APPLICATIONS

We will then implement the above interface to return the setting value from the appSettings section of the web.config or app.config file using the following code:

```
class AppConfigConfiguratonSettings : IConfigurationSettings
{
   public override string GetSetting(string settingName)
   {
      return ConfigurationManager.AppSettings[settingName];
   }
}
```

As we can see the class above implements IConfigurationSettings and uses the System.Configuration ConfigurationManager to return the setting value. The AppConfigConfigurationSettings class should ideally be located in the same assembly as the interface and both IConfigurationSettings and AppConfigConfigurationSettings should be located in a common assembly that all our applications should be able to access.

IMPLEMENTING OUR CONFIGURATION SETTINGS INTERFACE IN WINDOWS AZURE WEB ROLES

Finally we need to provide an implementation of IConfigurationSettings that can be used to retrieve the configuration settings from the Service Configuration file via the RoleEnvironment.

```
class RoleEnvironmentConfigurationSettings : IConfigurationSettings
{
   public override string GetSetting(string settingName)
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
    {
       return RoleEnvironment.GetConfigurationSettingValue(settingName);
    }
}
```

The above class can be provided in a separate assembly that would not be shipped with non-Windows Azure instances of your web application and would not be referenced by any of the common layers. As we can see from this implementation the common layers have no references to any of the Windows Azure SDK Assemblies.

CALLING THE CORRECT IMPLEMENTATION

So now that we have defined our pluggable classes how do we call the correct implementation of the interface? First of all we need to our common assembly to reference the Unity Application block assembly (Microsoft.Practices.Unity.dll) and add the following namespaces:

```
using Microsoft.Practices.Unity;
using Microsoft.Practices.Unity.Configuration;
using Microsoft.Practices.Unity.StaticFactory;
```

Finally when need to retrieve a configuration setting in our application we just need to call the following code:

```
IUnityContainer myContainer = new UnityContainer();

IConfigurationSettings configurationSettings =
  myContainer.Resolve<IConfigurationSettings>();

configurationSettings.GetSetting("mySetting");
```

As you can see from the code above when we call the Resolve method it will work out what configuration settings provider is registered and return that instance of the class. We can then retrieve the configuration setting from the registered provider (AppSettings version or RoleEnvironment version) using the interface method GetSetting.

Finally if we wish to use the AppSettings version in our non Windows Azure hosted applications then we would register this class using the following code:

```
IUnityContainer myContainer = new UnityContainer();

myContainer.RegisterType<IConfigurationSettings,
                         AppConfigConfiguratonSettings>();
```

To register the Windows Azure version we could call the following code instead from our application:

```
IUnityContainer myContainer = new UnityContainer();

myContainer.RegisterType<IConfigurationSettings,
                         RoleEnvironmentConfigurationSettings>();
```

Unity is not the only implementation of IOC available it just happens to be the one used in this example. For more information about Unity or IOC please read the following link:

http://msdn.microsoft.com/en-us/library/cc440954.aspx

Now that we've looked at how we can share our data across Windows Azure and Non-Windows Azure applications, let's look at how we can centralize our configuration into a manager class.

5.4.2 Centralized file-reading code

It is a best practice to take your entire configuration file reading code, where ever it is sprinkled, and move it into a `ConfigurationManager` class of your own design. Many people use the term serviced instead of manager, but I think the term service is too overloaded, and manager is just as clear. This centralizes all of the code that knows how to read the configuration in one place, making it easier to maintain. More importantly, it removes the complexity of reading the configuration from the relying code, which is the principal of separation of

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

concerns. This also makes it easier to mock out the implementation of the `ConfigurationManager` class for easier testing purposes. Over time when the APIs for accessing configuration change, or if where your configuration lives changes you will have only one place to go to make the changes you need.

Figure 5.4 – A well designed ConfigurationManager class can centralize the busywork of working with the configuration system.

5.5 How the RoleEnvironment class communicates with Windows Azure

In this chapter we have spent a lot of time using the RoleEnvironment class to interact with the Windows Azure runtime. I thought it be kind of interesting to rip away some of the covers and look at how this happens under the covers.

As we already know The RoleEnvironment class is hosted by the Microsoft.WindowsAzure.ServiceRuntime assembly which is provided as part of the Windows Azure SDK. On startup of your application the RDRoleHost service will attempt to call RoleEnvironment.Initialize on any hosted web role application. This call will populate a singleton instance of the RoleEnvironment class (private static variable called runtime) which is used by static method wrapper methods such as GetConfigurationSettingValue and IsAvailable.

Callbacks

On initialization of the RoleEnvironment class, a callback will be created between the RoleHost and the RoleEnvironment runtime singleton. This means if any configuration changes are made on the Azure Portal and propagated to the VM then the RoleHost will invoke the callback and notify the Web Role that some configuration has changed. This callback mechanism is created across a named pipe. Figure 5.5 shows the communication between your application and the role host.

Figure 5.5 - Named Pipe Communication between the Web Role Host and the Service Runtime

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

ROLEENVIRONMENT.ISROLEENVIRONMENTRUNNING

Now that you understand some of the internals of the RoleEnvironment class then it may be no surprise how RoleEnvironment.IsAvailable is implemented.

If the application is not running in the fabric then the Initialize method will never be called so the singleton instance will be null. RoleEnvironment.IsAvailable simply checks if that singleton object is null.

ROLEENVIRONMENT.GETCONFIGURATIONSETTINGVALUE

So what happens under the hood when the RoleEnvironment.GetConfigurationSettingValue is called? The GetConfigurationSettingValue static method is just a wrapper to the GetConfigurationSettingValue method in the non static version of the RoleEnvironment class. As explained in the Callback Section that any changes to the configuration are propagated via a callback over the named pipe.

This means any configuration settings can be retrieved quickly from an in-memory copy of the configuration settings that were populated as part of the AcceptConfigurationChanges callback.

WHY DO I CARE?

Now that you understand the internals of the RoleEnvironment class exposed via the Service Runtime class you will now understand some of the earlier points that we made in this chapter i.e.

- Why RoleEnvironment.IsAvailable will work in both Azure and Non Azure environments
- Why you can't use RoleEnvironment.GetConfigurationSettingValue outside of Azure (i.e. there is no named pipe interface in a non Azure environment)

Finally, by understanding how these RoleEnvironment class works internally you also understand there is no performance hit involved when you use the RoleEnvironment class as all data is cached in memory.

5.6 Summary

In this chapter, we looked at the final piece of the Service Model puzzle, i.e. Configuration.

You should now really fully understand the Service Model of Windows Azure and be able to effectively define and configure your service inside and outside of Windows Azure.

With this knowledge you should be able to understand the effect of the Service Model and how it applies to your application.

Finally, when you define and your configuration, you should now understand what impact and what this means to your application when it runs within the Windows Azure environment.

With all our newly gained knowledge, it's time to move on.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

6

Scaling web roles

This chapter covers:

- What happens to your Web Server in Extreme Load
- Scaling your Web Role
- The Load Balancer
- Session Management
- Caching

One of the coolest things about Windows Azure is the ability to dynamically scale our application. Whenever we need more computing power we can just ask for it and get it (as long as we can afford it). The downside is that in order to harness such power, you need to design your application correctly. In this chapter we'll look at what happens when your application is under pressure and how you can use Windows Azure to effectively scale your web application.

6.1 What happens to your web server under extreme load?

Back in chapter 1, we talked about the challenges of handling and predicting growth for typical websites. In this section, I really want to show you what happens to a web server when it's under extreme load and how it handles itself. So using the Ashton Kutcher example from Chapter 1; what would happen to your web server if Ashton Kutcher twittered about your little Hawaiian Shirt Shop and you suddenly found thousands of users trying to access your website at the same time?

In an ideal world, if your website (or a service) reached its maximum operating capacity then all other requests would be queued and the application could handle the load at a graceful yet throttled rate. In the real world, your website is likely to explode into a ball of flames. The Web Server will continue to attempt to process all requests (regardless of the rate). This means the processing time of the requests will increase meaning a longer response time to the client. Eventually the server will become flooded with requests and it will no longer be able to service those requests. This will effectively render the server useless until the requests reduce.

In this section we will look at how a web server performs both under normal and extreme load, we will do this by:

- Building a sample application that can run under extreme load
- Building an application that can simulate extreme load on our web server
- Looking at how our sample application responds when the server is under load
- Increasing the ability to process requests by scaling up or out

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

To do that we need to build a small ASP.NET web page sample application that we can use in all these scenarios.

6.1.1 Web Server under normal load

The web page that we are about to build will simply perform an AJAX poll that returns the time of the request. In normal operation we expect the page to return the current server time, every 5 seconds. Figure 6.1 shows this AJAX web page in normal operation.

Figure 6.1 – An ASP.NET Web Page making an AJAX request returning the server time every 5 seconds

As we can see from Figure 6.1, our web page is returning the server time every 5 seconds (as the timer prescribes). In this case we can be happy that our web server seems to perfectly capable of handling the load.

Let's take a look at how we can build this web page. Listing 6.1 gives the code listing for the ASP.NET Ajax Web Poll show in Figure 6.1.

Listing 6.1 Ajax Poll Web Form

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:UpdatePanel   ID="RequestsPanel" runat="server"
                   UpdateMode="Always">                        #1
   <ContentTemplate>
      <asp:Timer ID="Timer1" runat="server"
                 Interval="5000" OnTick="Timer1_Tick" />       #2
      <asp:Label ID="resultLabel" runat="server" />            #3
   </ContentTemplate>
</asp:UpdatePanel>
```

At #1 we have a standard ASP.NET AJAX update panel that contains a timer at #2, set to poll the web server every 5 seconds. On expiration of the timer the resultLabel at #3 will be concatenated with the current date and time using the following code.

```
protected void Timer1_Tick(object sender, EventArgs e)
{
   resultLabel.Text += DateTime.Now.ToString() + " <br/>";
}
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

So far everything is fine and dandy, let's now try and simulate what would happen if our server came under extreme load.

6.1.2 Simulating Extreme Load

To simulate extreme load, we are going to build a web page that will put our server on an extreme diet (i.e. starve it). In this case we will build an ASP.NET Web Form that will send the current thread to sleep for 10 seconds. If we then make enough requests to our web server, we should starve the thread pool making our website behave like a server under extreme load.

THE LONG RUNNING WEB PAGE

The following code shows the markup for our empty ASP.NET page.

```
<form id="form1" runat="server">
  <div>There is no need for any code in this sample</div>
</form>
```

As you can see from the above code, we really don't need our web page to display anything exciting. We really just need the code behind for our page to simulate a long running request by sending the current thread to sleep for 10 seconds, as shown below.

```
protected void Page_Load(object sender, EventArgs e)
{
   Thread.Sleep(10000);
}
```

Now that we have our web page that simulates long running requests, we need to put our web server under load by hammering it with requests.

HAMMERING OUR LONG RUNNING WEB PAGE WITH LOTS OF REQUESTS

To simulate lots of users accessing our website, we will now create a new console application that will spawn up 100 threads, and each thread will make 30 asynchronous calls to our new web page.

> **NOTE:**
>
> Your mileage may vary! You may need to increase the number of threads or calls to be able to effectively hammer the web server.

Listing 6.2 shows the code for the console application.

Listing 6.2 Load Simulation Console Application

```
static void Main(string[] args)
{
   Console.WriteLine("Creating 100 threads");
   var webAddress =
      new Uri("http://localhost:49399/Monitoring/CPUThrash.aspx");         #A
   for (int i = 0; i < 100; i++)
   {
      var t1 = new System.Threading.Thread                                  #1
      ( () =>
        {
           for (int j = 0; j < 30; j++)
           {
             WebClient client = new WebClient();
             client.DownloadStringAsync(webAddress);                        #3
           }
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
        }
      );

      t1.Start();                                                    #4
   }

   Console.ReadKey();
}
```

#A – URI for the long running request page
#1- Define the thread
#2 - Call the page
#3 – Spawn the thread

In the Console application Listing 6.2, we define the URI of our long running web page at #1. We then iterate through a For loop 100 times, creating a new thread at #1, which we will spawn at #4. This means that we will spawn up 100 threads.

Each thread will execute the code defined at #1, within the lambda expression. Essentially this thread will loop around 30 times, making an asynchronous request to the web page at #3. In the end we should really be making around 300 simulatenousish requests to our website. Let's now look at how the web server performs in such a scenario.

6.1.3 How the Web Server responds under extreme load

In figure 6.1 we saw the response time of a simple AJAX website being polled every 5 seconds under normal load. In that example there were no real issues and it could serve up the page with ease.

Figure 6.2 shows our same web application, this time coughing and spluttering as it struggles to cope with a simulated extreme load. The extreme load that this page is suffering from is caused by running our Console Application from section 6.1.2 at the same time as our polling application.

Figure 6.2 - Ajax Poll under extreme load (i.e. running at the same time as our console application.

As we can see from Figure 6.2, our polling web page is no longer consistently taking 5 seconds to respond return a result. At one point (18.40:33 and 18:41:04), it took more than 20 seconds to return the result. This type of response is typical of a web server under load. Since the Web Server is under load, the web server attempts to service all requests at one until it is so loaded that it can no longer effectively service any requests.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

6.1.4 Handling increased requests by scaling up or out.

If you are hitting the limits of a single instance then you should really consider hosting your website across multiple instances for those busy periods (you can always scale back down when you're not busy). Let's take a quick look at how that can be done manually in Windows Azure.

SCALING OUT

By default, a Windows Azure Web Role is configured to run in a single instance. If we wish to manually set the number of instances that our web role is configured to run on, we would set this value in our Service Configuration file as show below.

```
<WebRole name="MyWebRole"
  <Instances count="1" />
  <ConfigurationSettings />
</WebRole>
```

The configuration above shows that our web role is configured to run in a single instance. If we need to increase the number of active web roles to 2, then you could just modify that value from "1" to "2" as shown below.

```
<Instances count="2" />
```

Since the number of instances that a web role should run on is stored in the Service Configuration file we are able to modify the configured value via the Azure Service Portal at runtime. For more information on how to modify the service configuration file at runtime, see chapter 5.

Scaling Out Automatically

If you don't fancy increasing your web role instances manually, and wish to take advantage of the ability to increase and decrease the number of instances depending on the load, you can do this automatically.

In this instance you can monitor the number of requests / CPU usage etc using the Diagnostics API and then if you hit a threshold use the Service Management API's to increase the number of instances of your web role.

Alternatively, you could increase/decrease the number of instances based on the time of day based on a model derived from your web logs, i.e. Run one instance between 0300 and 0400 hours but 4 instances between 1900 and 2100 hours. This can be performed by create a Windows Scheduler Job to call the Service Management API at those times.

In the Worker Roles Chapter we will look at how you can automatically scale worker roles. The techniques used in that chapter also apply to scaling Web Roles.

Now that we have increased the number of instances that hosts our Web Role, if we were to re run our AJAX Polling Sample and our Hammering Console Application, we should be able to see that our polling application should be able to respond every 5 seconds as if it was under normal load (rather than taking 30 seconds or so to respond).

SCALING UP

If you have not designed your application to scale out i.e. your application has an affinity to a particular instance of a web role (e.g. In-Process Sessions, In-Memory Cache's etc) then you may not be able to scale out your web role instance.

In this case, you may just decide to run your web application on a bigger box. We can do this my modifying your vmsize in the Service Definition file (as shown below):

```
<WebRole name="MyWebRole" vmsize="Medium">
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

By default the web role is hosted on a small Virtual Machine which has 1GB of Memory and 1 CPU Core. In our example we have upgraded our Web Role to run under a medium Virtual Machine meaning that our Web Role has an extra CPU Core and more memory. For full details of the available Virtual Machine sizes see Chapter 3, section 3.4.4.

By increasing the size of the Virtual Machine, we should now find that our ASP.NET AJAX Web Polling application will be able to handle the extreme load being placed on the server (or at least process requests a little quicker).

Try to avoid Scaling Up and Scale Out instead

Although Scaling Up will get you out a hole, it's not an effective long term strategy. Wherever possible you should scale out rather than scale up. At the end of the day, when you scale up you can only scale up to the largest Virtual Machine size (and no further).

Also it's not easy to dynamically scale up and down your application based on load. Scaling Out just requires a change to the Service Configuration file. Scaling Up requires you to upgrade your application (as it requires a change to the Service Definition file).

Now that we understand what happens when our server is placed under load, and how we can effectively handle that situation by scaling our application. Let's take a look at what happens in a little more detail on how requests are distributed across multiple web role instances via the Load Balancer.

6.2 How the Load Balancer distributes requests

In this section we will look at how the load balancer distributes requests across multiple servers. By doing this we should be able to understand how our application will react / behave when we have multiple instances of our web role. Not only that, we will look at how the load balance reacts under failover conditions.

In this section we will look at both the:

- Development Fabric Load Balancer
- Production Load Balancer

Before we can do that we need to first build a sample application to demonstrate the effects of the load balancer coordinating requests between multiple servers.

6.2.1 Multi-Instance Sample Application

The sample application that we are about to build is a web page that consists of a label, which displays the name of the web server that processed the request, and a button, which will post back to the server when clicked. Every time the page is loaded (i.e. on first load or when the button is clicked), the page will write a message to the log. Figure 6.3 shows the web page and its log output in the Development Fabric.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 6.3 - A single instance of our web role running in the development fabric, writing out to the Diagnostic Log

Now we'll walk you through the steps of creating this simple web role so that you'll be able to be able to see the output as shown in Figure 6.3

CREATE THE WEB PAGE

To build the sample application shown in Figure 6.3, you should create a new ASP.NET Web Role (if you are unsure how to do this you should refer back to chapter 1). In the newly created web role project, you should now add a new ASP.NET web page called MultipleInstances.ASPX.

THE ASPX MARKUP

In the newly created ASPX page you should now add the following markup:

```
<div>
   Machine Name: <asp:Label ID="lblMachineName" runat="server" />
</div>

<asp:Button ID="btnClickMe" runat="server" Text="Click Me"/>
```

The above markup represents the page displayed in Figure 6.3; this includes a label that displays the name of the machine and a button that will post back to the web server when clicked.

THE CODE BEHIND

As you can see in figure 6.3, each time the page is loaded we wish to display the name of the web server that processed the request. To display the web server name in the lblMachineName label you should add the following code to the page load event of the asp.net page.

```
var computer = new Microsoft.VisualBasic.Devices.Computer();
lblMachineName.Text = computer.Name;
```

NOTE

Before you can run the above web page you must add a reference to the Microsoft.VisualBasic assembly and enable native execution in your Service Definition file in your cloud service project as shown below:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
<WebRole name="MyWebRole"
         enableNativeCodeExecution="true">
```

Finally in order to write some data to the log as shown in Figure 6.3 we need to write a message to the log on page load. You should now add the following code to the page load event of the ASP.NET Page:

```
System.Diagnostics.Trace.WriteLine("Hello Log");
```

You should now be able to fire up the application in the Development Fabric and see a message written to the log displayed in the Development Fabric UI every time the page is loaded.

Now that we've got our application working, we can take a look at how the load balancers route requests.

6.2.2 The Development Fabric Load Balancer

In typical ASP.NET Web Farms, it's really not easy to simulate the load balancing of requests. With Windows Azure a development version of the load balancer is provided to allow us to simulate the effects the real load balancer. The development fabric load balancer really helps find and debug any potential issues that we may have in our development environment (with a debugger too ☺). So let's take a look at how the Development Fabric Load Balancer behaves.

If you were to now fire up our web application in the development fabric with 2 web roles configured, you would notice that in the UI there would now be two instances of your web role displayed. Similar to the live production system the Development Fabric will distribute requests between the instances of your web role. Each time you click the button on the web page the request could be distributed to either instance of the web role. Figure 6.3 shows the 2 instances of the web role in the Development Fabric UI.

Figure 6.3 - Multiple instances of the web role in the Development Fabric

In figure 6.3 we can see that we have 2 instances of our web role in the development fabric but how is this represented on our development machine?

MULTIPLE INSTANCES OF RDROLEHOST.EXE

In chapter 4 we discovered that Windows Azure hosts the IIS 7.0 runtime in-process in the RDRoleHost.exe process using the "Hostable Web Core" feature of IIS7.0 rather than using the default w3wp.exe process. Since

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

our web application is hosted by the RDRoleHost.exe process if we increase the number of instances of our web role that need to be hosted then the Development Fabric will therefore instantiate multiple instances of the RDRoleHost.exe (one process per instance). Figure 6.4 shows the process list of my development machine when running multiple instances of the Web Role.

Figure 6.4- Multiple instances of RDRoleHost shown in the Process List

As you can see from Figure 6.4 there are 2 instances of RDRoleHost in the process list, one per Web Role instance.

Development Fabric Load Balancer Process

In order to help simulate the live production environment, the Windows Azure SDK includes a development load balancer that is used to simulate the real hardware load balancers that run in the Windows Azure data centers. Without the Development Fabric Load Balancer it would be very difficult to simulate issues that occur across multiple requests, as we cannot attach a debugger in Live (but can in the development environment). Si

The process that simulates the Load Balancer in the Development Fabric is called DFLoadBalancer.exe as shown in Figure 6.4. All http requests that you make in the Development Fabric will be sent to this process first and then distributed to the appropriate RDRoleHost instance of the web role.

If you were to kill one of the RDRoleHost instances, then all requests made to the Development Fabric load balancer would be redistributed to the other instance of RDRoleHost whilst the other RDRoleHost.exe process is automatically restarted. By performing this test we can simulate what will happen to our application if there is a hardware / software failure in the live environment.

If you were to kill your DFLoadBalancer.exe process, then the entire Development Fabric on your machine would be shutdown and would require restarting.

MULTIPLE INSTANCES

If you are testing your application in the development fabric to see how it responds when requests occur across multiple instances, then you should check that your requests have not just been processed by a single instance of the process.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Instance Favoring by the Development Fabric Load Balancer

In the development fabric it will tend to a favor a particular instance (per browser request) unless that role is under load. If we look at Figure 6.3 you will see that although we have two instances of our web role running the development load balancer seems to be routing all traffic to a single instance.

Since each browser window in the development fabric tends to have affinity with a particular web role instance, you should test your application using multiple browser instances. Figure 6.5 shows both instances of the web role being utilized by alternately clicking the button in 2 different browser instances.

Figure 6.5 - Multiple instances of the web role with requests distributed

As you can see from Figure 6.5, now that we are running 2 different web pages, the requests are distributed across both instances of the Development Fabric.

Although spinning up multiple instances of your web application with multiple browser instances will allow you to test in your development fabric that your application will work on multiple roles, it will not test the effects of being redistributed to another server.

In order to test that your application behaves as expected in your development fabric when running under multiple instances you should perform the following 2 tests:

- Restart one of the RDRoleHost instances
- Test your application under load

RESTARTING THE RDROLEHOST INSTANCE

Although the Development Fabric Load Balancer will create affinity between your process and an instance of RDRoleHost, if you kill that RDRoleHost instance the affinity will be broken. Once affinity is broken, the request will be redistributed to another instance of RDRoleHost and will therefore create a new affinity between your web browser and that instance of the web role. Figure 6.6 shows the changing of affinity between a browser and a web role instance when an RDRoleHost.exe process is killed.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 6.6 changing of affinity between browser and role host in development fabric

In the example shown in Figure 6.6, we had 2 instances of our Web Role running (instance 0 and instance 1), and a single web browser. As you can see from figure 6.6 instance 1 did not process any requests prior to the RDRoleHost.exe process of instance 0 being restarted as the browser had affinity with instance 0. Once instance 0 was restarted, instance 1 then processed all incoming requests and instance 0 no longer processed any requests from the browser as a new affinity was created.

This is a pretty important test to perform within the development fabric as it will allow you to ensure that your application can truly run against multiple instances of your web role and can recover itself from a disaster. We will look at these situations in more detail when we look at Session, Cache and Local Storage.

TESTING UNDER LOAD

The second method of testing that your application behaves correctly when redistributed to multiple instances is to test your application under load. The best method of testing this scenario would be to use a load testing tool such as Visual Studio Team System Web Load Tester. If you just want to ensure that the Development Fabric Load Balancer will redistribute requests under load you can simulate this by modifying the sample application that we built earlier. In this case we will simulate load by sending the thread to sleep for 10 seconds on page load. We can do this modifying our page load event to include the following code:

```
Thread.Sleep(10000);
```

If you were to now spin up multiple instances of your web page, you will see that due to the increase in load the Development Fabric Load Balancer will no longer maintain affinity and start redistributing the requests more evenly.

Asynchronous AJAX Requests

Although the Development Fabric Load Balancer will keep affinity between your browser instance and a web role (unless under load), it will tend to redistribute requests evenly when performing AJAX requests.

In Section 6.1 we created an ASP.NET AJAX web page that would asynchronously call the backend web page every 5 seconds displaying the name of server that processed the request. If you were to modify that sample to write to the log then you would find that the Development Fabric Load Balancer would evenly distribute the request rather than maintaining affinity with a particular web role instance.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Now that we have an understanding of how Development Fabric Load Balancer behaves, and how we can effectively test our application will behave in failover situations, let's look at how things happen in the live environment.

6.2.3 Load Balancing in the Live Environment

So we have spent quite a bit of time looking at the Development Fabric Load Balancer hopefully you will now have a good understand of how the load balancer works and how it interacts with multiple instances of your web role. Although the Development Fabric Load Balancer does not behave exactly like the real environment there are some tricks that you can do to ensure that your application will behave correctly in the live environment. That said there are some cases where your application may not behave as expected when distributed across physical servers, this is not something you can easily test for in the development fabric. In order to ensure that your application will behave correctly prior to make your application live you will need to perform some testing within the staging environment.

STAGING ENVIRONMENT

In chapter 2 we showed you how to deploy your application to the staging environment and how to move your staging web application to the production environment via the Windows Azure Portal. In that chapter we also said that when you switch your application from staging to production that the application will continue to run on the same server as before and the load balancers will simply redirect traffic to the correct servers. If you want you can actually prove this is the case by deploying the sample that we built earlier to the staging environment (noting the machine name) and then switching over to the production environment (noting the machine name once more). Figure 6.7 shows our application running in the staging environment and the live environment.

Figure 6.7 Web Application running in Staging then in Production

If we look at Figure 6.7, the left hand browser is pointed against the staging environment and is running on machine RD00155D3021EB. We then switched the web application from staging to production. The right hand browser is our web application running in production after the switchover, notice that that our application is still running on the same server even though we are now running in production (rather than staging).

Since the staging servers will eventually become the production servers we should be able to iron out any issues during our staging testing phase any errors that may occur when running multiple instances of our web role.

> **Nothing beats production environments**
>
> You can't always rely on the fact that an application that works in you dev environment will work in the staging environment. On your development machine, RDRoleHost will run under your user account. Anything you can do, it can do. That's not necessarily true in the production servers

STAGING + PRODUCTION LOAD BALANCERS

If were to take our ASP.NET Web Polling application from section 6.1 and modify to display the machine name, and then ran it under the staging environment. We would get a result to what is shown in Figure 6.8.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 6.8 - Web roles running in the staging environment

You will notice from figure 6.8 that the requests are distributed between two different servers. This is great news as we can be sure that the staging environment will host each instance of our web role on a separate physical machine (a test that we cannot perform in our development environment).

Quirky affinitization

Both the Staging + Production environment can be a little quirky in how it distributes requests between servers. In some instances it will evenly distribute requests between all servers (e.g. AJAX requests) but generally will maintain affinity between a connection and a web role. Therefore if you are testing you're application works with multiple instances of your web role, you should probably capture the machine name in your request to ensure that your server can handle requests distributed across instances. You can test this by running our sample applications on the staging + production environments and monitoring how the machine name changes.

INDUCING FAILOVER

In the development fabric we tested how our application handled failover by killing the RDRoleHost.exe process then monitoring how the application handles the failover. We can perform the same test in the staging and production environments if necessary. In the live environment the Web Role is also hosted in a process called RDRoleHost.exe, so we can kill the process on the live environment using the following command (remember native execution must be enabled for this to work):

```
System.Diagnostics.Process.GetProcessesByName("rdRoleHost").First().Kill();
```

To test this scenario you should create a new web page with a "kill process" button that will execute the above command when clicked. This will then allow you to run the AJAX Polling application in one browser then kill the RDRoleHost process in another browser and watch how the load balancer handles the redirection of traffic. Typically on the live system (at the time of writing), it will redirect all traffic to the single node and then once the Fabric Controller is convinced that the failed role is behaving again the load balancer will start to direct requests to that server also.

If you wish to test what happens when all roles are killed then you can execute the "kill process" request on each role until they are all dead. If no web roles are running in the live environment your web application will no longer process requests and your end user will be faced with an error. Typically your service will be automatically restarted and will be able to service requests again within about a minute.

Now that we have a good understanding of how requests are load balanced across multiple web roles, we should now take a look at those aspects of a website that are generally affected when running with multiple roles, namely:

- Session Management

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

- Caching
- Local Storage

Let's start with session management

6.3 Session Management

HTTP is a stateless protocol each http request is an independent call with no knowledge of state from any previous requests. Session is one method of persisting data so that it can be access across multiple requests. In ASP.NET there are various methods of persisting data across requests including:

- Session
- Viewstate
- Cookies
- Application State / Caching
- Profile / Database

Throughout the course of this chapter (and future chapters) we will be looking at the methods of persisting data that you will use that will be affected when you scale to multiple web roles. We will not look at ViewState or Cookies in this book as the use of the technologies do not change in a Windows Azure environment.

In this section we will look at how running Windows Azure across multiple roles will affect your ASP.NET Session and the different types of session provider that you can use. Specifically we will look at:

- How Session works?
- Building a session sample application that we can use
- In-Process Sessions
- Table Storage Session

Although the concept of session is probably familiar to most of you, I do want to recap on the purpose of the session and the Session object as it will help throughout the course of this section.

6.3.1 How Session Works?

A session is effectively a temporary store that is created server side for a limited window of time for a particular browser instance. Your ASP.NET Web Application can then use this temporary store to store and retrieve data throughout the course of that session.

For example in the case of the Hawaiian Shirt Shop, we may choose to store the shopping basket in the session. Using the session as a storage area for our session would allow us to store items in the cart and then still have access to the cart across multiple requests. Once the session is terminated (e.g. the user closes his browser) then the session is destroyed and the data stored in the session is no longer accessible. Similarly if the user was to open a new browser instance then a new independent session would be assigned to this browser instance and the new browser instance would have no access to session data associated with the other browser instances (and vice-versa). Figure 6.9 shows how session is treated in respect to the browser instance and the web server.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 6.9 - Session Independence and Temporary (errghh ness)

As we can see from Figure 6.9, Browser Instance 1 is associated with Session Id: 12345 and the session key "Foo" has an associated value of "Bar". Browser Instance 2 is associated with Session Id 12346 and the session key "Foo" has an associated value of "Bar". If we look at the output of Browser Instance 1 and Browser Instance 2 they will display the correct values from their associated temporary store.

If you need to be able to access data beyond a browser session then you should probably consider a more permanent storage mechanism such as Table Storage or SQL Azure Database.

ASSIGNMENT OF SESSIONS

Finally to finish up our discussion of how sessions work, when a browser makes a request to an ASP.NET Website it will pass across a Session Id in a cookie as part of the request. This Session Id is used to marry up the request to a session store. For example in Figure 6.9 Browser Instance 1 has a Session Id of 12345 and Browser Instance 2 has a Session Id of 12346. If no Session Id is passed in the request then a new session will be created and that Session Id will be passed back to browser in the response to be used by future requests.

Now that we have reminded ourselves how sessions work in ASP.NET, we will build a small sample application that we can use to demonstrate the effects of Session and how this affects our web applications in Windows Azure.

6.3.2 Session Sample Application

To started we need a web page where we can store a value in the session for later retrieval. You should add this new ASP.NET Web Page called SessionAdd.aspx to an ASP.NET Web Role project. You should add the following markup to the page:

```
<asp:TextBox ID="txtSessionText" Text="" runat="server"/>
<asp:Button ID="btnAdd" Text="Add"
            runat="server" onclick="btnAdd_Click"/>
```

As you can see from the above markup our page consists of a TextBox and a Button. On click of the button you will set the value of the session key Foo as shown below:

```
protected void btnAdd_Click(object sender, EventArgs e)
{
   Session["Foo"] = txtSessionText.Text;
}
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

SESSION TIMER PAGE

Now that we can store some session data in our page we need a page that we can use to display whatever is stored in Foo. In this case we will use an ASP.NET AJAX polling timer similar to the one that we used earlier that will display whatever is stored in Foo every 5 seconds. Figure 6.10 shows how this web page should look:

Figure 6.10 Session Timer Page, notice that our session is maintained for each request

The output displayed in Figure 6.10 shows the value of the Session Key "Foo" every 5 seconds. Prior to running this page, I used the Session.aspx page to the set the value of "Foo" to bar.

To create the page displayed in Figure 6.10 you should now add a new ASP.NET page to the project called SessionTimer.aspx which should contain the following markup:

```
<div>
   <asp:ScriptManager ID="ScriptManager1" runat="server" />

   <div style="padding-bottom: 10px;">
      This page polls the webserver using Ajax every 5 seconds
   </div>

   <div>
      <asp:UpdatePanel ID="RequestsPanel"
                       runat="server"
                       UpdateMode="Always">
            <ContentTemplate>
               <asp:Label ID="lblResult" runat="server" />
                  <asp:Timer ID="Timer1"
                             runat="server"
                             Interval="5000"
                             OnTick="Timer1_Tick" />
         </ContentTemplate>
      </asp:UpdatePanel>
   </div>
</div>
```

To display the result of Foo in the web page you should add the following code-behind to the SessionTimer.aspx page:

```
protected void Timer1_Tick(object sender, EventArgs e)
{
   RoleManager.WriteToLog("Warning", "called");                          #1

   var computer = new Microsoft.VisualBasic.Devices.Computer();

   lblResult.Text +=
      string.Format("Time: {0}, Machine Name: {1}, Session (Foo): {2}<br/>",
DateTime.Now.ToString(), computer.Name, Session["Foo"] as string);         #2
}
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Every 5 seconds the Web Page will make an AJAX request back to the Web Server where we will log the request at #1 and then return the name of the computer, the time of the request and the value stored in the Session for Foo, which will then be displayed in the web page at #2.

This sample will allow us to see both in the development and live environment which machine processed the request and what the value of Foo is at any particular time

6.3.3 In-Process Session Management

ASP.NET by default uses an in-process session provider to store session data. The in-process session provider simply stores all session data in memory scoped to the web worker process (w3wp in standard web servers, or RDRoleHost in Windows Azure).

KILLING OUR SESSION BY KILLING RDROLEHOST

Since the data is stored in memory if the worker process was to be restarted then you would lose any session data. We can actually simulate this situation in our development environment using our Session Timer page.

> **NOTE**
>
> Before we attempt to lose our session, please ensure that your ASP.NET Web Role is running with a single instance.

If you now fire up the SessionAdd.aspx page that we created earlier and set the value of "foo" to "bar". Once this value is set you should then open up SessionTimer.aspx within the same browser instance. After the session value is displayed a few times you can then kill the RDRoleHost process. As we discovered earlier if you kill the RDRoleHost process the Development Fabric will automatically restart the process however all session data will then be lost. Figure 6.11 shows the result of killing the process.

Figure 6.11 - killing the web role with in-proc session

In figure 6.11 you can see that "bar" was displayed up until 13:49:55 and just after that point we killed the RDRoleHost process. From that point onwards the session was lost and no data was returned for all other requests.

IN PROCESS SESSION STATE WITH MULTIPLE INSTANCE OF THE WEB ROLE

Already we have seen some issues with using the in-process session provider in Windows Azure but this is the real killer. If you are using multiple web role instances Windows Azure does not consistently implement sticky sessions

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

in the production environment so any requests made to a web role may not be routed to the same web role. As we discovered earlier the production systems will generally maintain affinity with a web role but will sometimes evenly distribute requests amongst roles. In the case of AJAX applications, since requests are likely to be distributed across multiple roles an in-process session provider cannot be used as the other role will not have access to session data stored in a previous request. Figure 6.12 shows our AJAX polling application running across multiple web roles.

Figure 6.12 - Loss of session data across multiple web role instances

As we can see from Figure 6.12 any request made to the first role will return the session data but anytime the request is distributed to another web role the session data stored in the first web role is no longer accessible.

Without our understanding of the load balancers and of how development + production load balancers we may not have discovered this issue until we deployed our application to production thinking that Azure implemented sticky sessions.

MEMORY CONSUMPTION

If at first you only need to run your web role on a single instance then you can get better performance by running your application with in-process session management. This should be considered as option if you are unconcerned that your users session may be trashed if the web role is moved to another server (e.g. during a Windows Update). If you need to scale out to multiple servers at a later date you can always move to a session provider that will work cross multiple web roles (such as table storage) when required.

Before you decide to run with the in-process session provider there is one other issue that you should be aware of. If you have a large amount of users on your website storing a large amount of session data you may quickly run into "Out of Memory" exceptions. The Web Role host will not automatically free up any active session data until sessions start to expire. Remember that your VM only has 1Gb of memory allocated to it, so you will run out of memory quite quickly.

If you wish to test how your application responds to adding a large amount of session data, you can modify the SessionAdd.aspx page to include a button that will add a large amount of data to the session when clicked. You should add the following markup to your SessionAdd.aspc page.

```
<asp:Button ID="btnLarge" Text="Large"
            runat="server" onclick="btnLarge_Click"/>
```

The following code will add some large data to the session when the button is clicked:

```
protected void btnLarge_Click(object sender, EventArgs e)
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
{
   StringBuilder sb = new StringBuilder();

   for (int i = 0; i < 100000; i++)
   {
      sb.Append("Hello World");
   }

   for (int i = 0;  i < 10; i++)
      Session[Guid.NewGuid().ToString()] = sb.ToString();
}
```

By repeatedly clicking this new button on your website you will find that the memory usage of your RDRoleHost.exe process will increase until you start getting "Out of Memory" exceptions.

OUT OF PROCESS SESSION MANAGEMENT

In Windows Azure the Out of Process session state provider is not supported.

6.3.4 Table Storage Sample Session Provider

In order to maintain session state that can be accessed by multiple web roles that could have requests evenly distributed between them, we need to use a persistence mechanism that can be accessed by all web roles. In typical ASP.NET Web Farms, SQL Server is typically used especially since ASP.NET has a built-in provider that supports it.

In Windows Azure as part of the SDK a table storage session provider is available for you to use in your ASP.NET Web Applications.

GETTING STARTED WITH THE TABLE STORAGE SESSION PROVIDER

To start using the Table Storage Session Provider you will need to build the sample provider and then reference that provider in your project. The sample provider can be found at the following location:

```
http://code.msdn.microsoft.com/windowsazuresamples
```

To build the project, just double click the **buildme.cmd** file in the directory. Once built you should add a reference to the assembly in your web role project. Since the table storage provider is implemented as a custom provider you will need to modify your web.config to use the provider as shown below:

```
<sessionState mode="Custom"
              customProvider="TableStorageSessionStateProvider">
   <providers>
      <clear/>
         <add name="TableStorageSessionStateProvider"
type="Microsoft.Samples.ServiceHosting.AspProviders.TableStorageSessionStateProvider"
             allowInsecureRemoteEndpoints="false"
             accountName="devstoreaccount1"

sharedKey="Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPTOtr/KBHBeksoG
MGw=="
             containerName="sessionstate"
             applicationName="ProviderTest"
             blobServiceBaseUri="http://127.0.0.1:10000"
             tableServiceBaseUri="http://127.0.0.1:10002"
             sessionTableName="Sessions" />

   </providers>
</sessionState>
```

TODO: Need to update when online sample is updated

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The above configuration is to use the table storage provider in the development fabric. On build of your application you will be prompted to create the required table in your development storage account.

On deployment of your application you will need to modify the web.config to use your live table storage account. The creation of the appropriate tables in the table storage account is automatically taken care of for you by the provider.

If you now run your application in either the development fabric or the live environment you will find that you can now store and retrieve session data across multiple instances of your web role.

Ever growing tables

One word of warning about the table storage provider is that does not clean up expired sessions after itself. Since the table storage is a paid metered service you would be advised to either have a worker role, simulated worker role (see later on in this chapter), or background thread that clears up any expired sessions from the table. If you do not clean up this data you will be paying storage costs for data that is no longer used.

PERFORMANCE CONSIDERATIONS

Although table storage will give you a session state that is accessible across multiple server instances in a load balanced environment it does incur a performance hit. To test the performance of the live system, I modified the table storage provider to record the time taken between requesting an item from the session and retrieving a response. Since session is reloaded from the session provider on every page load this test will allow us to see the impact of the table storage session provider on our website. Figure XX.11 is a modified version of the SessionTimer page that we built earlier that also displays the time recorded to load the session during the page load.

Figure 6.13 Table Storage session provider response times for a single small item

As you can see from Figure 6.13 although we are only storing one item in the session ("bar" in session key "foo") it still takes somewhere around 0.1 to 0.2 seconds to retrieve the session state. This load time is probably acceptable for most applications and therefore table storage is probably a good solution for the session in Windows Azure until a more performant solution such as a cache based session provider is available (see later).

If you store a large amount of data in the session then you may find the performance of the table storage provider a little too slow at the moment. Figure 6.14 shows the sessiontimer.aspx page after I added a large amount of data to the session by clicking the large session button that we built earlier twice.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 6.14 large session state response load times using the table storage provider

In figure 6.14 you can see that the performance of the session provider seriously degrades with a large amount of data stored in the session. In the above example it takes around 1-2 seconds just to load the session alone.

In those cases where you need minimum session load time or where you are storing large amounts of data you may wish to consider another session provider solution (e.g. SQL Azure Database or a cache based session provider).

SQL Azure Session Provider

In typical ASP.NET Web Farms, SQL Sessions Providers are generally used as the Session Provider. Although this works it is not really the best use of a SQL database (i.e. we never query across sessions, it's just acting as central storage area.

To date, there is no SQL Azure Session Provider available (although this could change). Rather than trying to mess with SQL Azure to make it work with sessions, it's probably best to either stick to Table Storage, use a cache based session provider, use in-process, or architect your application so it's not so reliant on sessions.

6.4 Cache Management

In any typical website there is usual some element of static reference data in the system. This static reference may never change or only change very infrequently. Rather than continually querying for the same data from the database, storing the data in a cache can provide large performance benefits.

A cache is a temporary in-memory store that contains duplicated data populated from a persisted backing store such as a database. Since the cache is an in-memory data store retrieving data from the cache is very fast (compared to database retrieval). Since a cache is an in-memory temporary store then if the host process or underlying hardware dies then the cached data will be lost and the cache will need to be rebuilt from its persistent store.

In most systems there is typically two layers of cache that will be utilized:

- In-Process Cache
- Distributed Cache

Individual Server Cache

For small sets of static reference data then a copy of the cached data would reside on each server instance. With the data residing on the actual server then there is no latency with cross-server roundtrips giving the fastest possible response time.

Let's take a look at the first and most simple type of cache we can have in Windows Azure, the ASP.NET Cache.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

6.3.1 In-Process Caching with ASP.NET Cache

As of the PDC09 release the only available caching technology to Windows Azure Web Roles is the built-in ASP.NET Cache which is an in-process individual server cache. Figure 6.15 shows how cache related to your web role within Windows Azure.

Figure 6.15 ASP.NET Cache, notice that each server maintains it's own copy of the cache

As you can see from Figure 6.15 that both Server A and B would maintain its cache of the data retrieved from the data store (Table Storage or SQL Azure Database). So although there is some duplication of data the performance gains make this worthwhile.

In-Process Memory Cache

You should also notice from Figure 6.15 the default ASP.NET cache is an Individual Server Cache that is tied to the web server worker process (RDRoleHost). Therefore in Windows Azure any data you cached will be held in the RDRoleHost processes memory space. If you were to kill the RDRoleHost process then the cache would be destroyed and the cache would need to be repopulated as part of the process restart.

Since the Virtual Machine has a maximum of 1GB of memory available to the server then you should keep your server cache as lean as possible.

Although In-Memory caching is suitable for static data, as we can see it's not so useful when we need to cache data across multiple load balanced web servers. To make that scenario possible we need to turn to a distributed cache such as Memcached.

6.3.3 Distributed Caching with Memcached

Memcached is an open source caching provider that was originally developed for the blogging site Live Journal. Essentially it is really a big hash-table that you can distribute across multiple servers. Figure 6.16 shows 3 Windows Azure web roles accessing data from a cache hosted in 2 Windows Azure worker roles.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 6.16 – 3 Web Roles accessing data stored in 2 worker role instances of memcached.

As you may have gathered from figure 6.16, that Microsoft has developed a Solution Accelerator that you can use as an example of how to use Memcached in Windows Azure. This acceleration contains a sample website and the worker roles that host memcached.

Download the Memcached Solution Accelerator

You can download the accelerator from:

http://code.msdn.microsoft.com/winazurememcached

Please note that Memcached.exe is not included in the download. You should use version 1.2.1 from http://jehiah.cz/projects/memcached-win32/

To get started with the solution accelerator, you just need to download the code and follow the instructions to build the solution.

Hosting Memcached

In Chapter 7, we will look at how you can launch executables (such as Memcached) from a Windows Azure role.

Although we have used a worker role to host our cache, you could also host memcached in your web role (saves a bit of cash)

Although we won't go through the downloaded sample, it is taking a peak at how we store and retrieve data using the accelerator.

Inter Role Communication

In Figure 6.16 we can see that our web roles are able to directly communicate with worker roles. In the Worker Roles chapter later in this book, we will look at how you can do this within Windows Azure

SETTING DATA IN THE CACHE

If we want to store some data in memcached, we can use the following code:

```
AzureMemcached.Client.Store(Enyim.Caching.Memcached.StoreMode.Set, "myKey", "Hello World");
```

In the above example, we would store the value "Hello World" against the key "myKey". Now that we have data stored, let's take a look at how we can get it back (regardless of what web role we are load balanced to).

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Caches are unreliable

Never rely on data stored in Cache, you should always populate cache data from a persisted storage medium such as Table Storage. This allows you to persist back to that medium if the data is not present in the Cache.

RETRIEVING DATA FROM THE CACHE

Retrieving data from the cache is pretty simple, the following code will retrieve the contents of the cache using the AzureMemcached library:

```
var myData = AzureMemcached.Client.Get<string>("myKey"));
```

In the above example, the value "Hello World" that we set earlier for the key "myKey" would be returned.

Although our Memcached example is very cool, you will notice that we are not using the ASP.NET Cache object to access and store our data. The reason for this is that unlike the Session object, the Cache object (in .NET Framework 3.5SP1, 3.5 or 2.0) doesn't use the Provider factory model meaning that the Cache object can only used be in-conjunction with the ASP.NET Cache provider.

6.3.4 Cache extensibility in ASP.NET 4.0

From ASP.NET 4.0, you will be able to specify a cache provider to be used other than the standard ASP.NET in-memory cache. Although this feature was introduced to support Microsoft's new distributed cache product (Velocity), it can be used to support other cache providers such as Memcached. The configuration of a cache provider is similar to the configuration of a session provider. The following configuration would be used to configure your cache to use Velocity:

```
<system.caching>
   <cache defaultProvider="FrameworkCacheProvider">
      <providers>
       <add name="myVelocityInstances"
                type="System.Data.Caching.VelocityCacheProvider,System.Data.Caching"
                remoteServerName="myServer"
                remoteServerPort="4435"
                namedCache="myCache"
                securityToken="DEC3D34CA29112" />
      </providers>
   </cache>
</system.caching>
```

Although the Windows Azure team will allow you to hook into any cache provider that you like, the real intention will be for you to use a Windows Azure hosted shared Cache Role that will probably be based upon Velocity.

Cache Based Session Provider

Now that you can see the benefits of a distributed cache it's worth re-visiting session providers. As stated earlier most ASP.NET web farms tend to use SQL Server as the session provider. With the increasing popularity of distributed caches it is now getting more common for web farms to use a cache based session provider rather than a SQL Server based provider.

Once a distributed cache is utilized within a web farm it makes sense to leverage it wherever possible to maximize its value. Typically distributed cache's have better performance than databases such as SQL Server meaning that you can improve the performance of your web application by using a faster session provider. Since session state is ultimately volatile data (not unlike cached data) a transactional data storage mechanism such as SQL Server is typically overkill for the job required.

If you want to use a Memcached based session provider (i.e. your session data is stored in your memcached instance), then you can download a readymade provider from:

http://www.codeplex.com/memcachedproviders

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

6.4 Summary

Ok, so you probably have learned everything that's relevant about scaling your web tier, from this chapter (and even from some of the earlier chapters. You should now come to realize that websites really can't cope with being under pressure and that best thing that you can do is design your web tier to scale out. If for whatever reason (and there really isn't a good one that I can think of) that you can't scale out, you can always host your website on a bigger box in the mean time.

In this chapter we also looked at how Windows Azure distributes requests and how we can test them in our own environment using the Development Fabric Load Balancer. Although we can't test every scenario it will tell us how our application will behave when we run under multiple instances. Finally we learned how to handle sessions and caching across multiple servers (if we want to).

Now that we're starting to look at some of the more advanced web scenarios, it's probably worth taking a peak at how we can use Windows Azure support for full trust applications and look at building non ASP.NET based websites and executing non .NET Framework applications, in the next chapter.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

7

Running Native, Full Trust and Other Code

You will be enlightened in this chapter by learning:

- How to run any CGI interpreter you want
- Spawn processes and call local executables
- Call native libraries with p/invoke

Microsoft is committed to making Windows the best place to run any type of application. To do this they are making Azure an open system, where you can run anything you want. Microsoft could have easily made Azure .NET only. This would make it easier for them to manage it, and easier for them to design the infrastructure. But they didn't. They opened it up, as wide as the on-premises version of Windows is, so that their customers could run almost anything on Azure that can be run on Windows today. Azure will run unmanaged code (C++ for example), any code that needs full trust on the local machine, and code from any other platform that runs on Windows. There is support for PHP, Python, Ruby and Java. Microsoft didn't stop there.

They worked with a series of open source teams to provide useful and valuable SDKs for each platform, so that they are equal citizens in the cloud. A plugin for Eclipse had been developed in a partnership between Microsoft and an open source team so that Eclipse developers can have an integrated experience. The openness of Azure is very powerful.

The core to this openness is Azure's support for running in a full trust environment. Once this was enabled, you could run anything on Azure, including FastCGI. Once you could run FastCGI you could run most other web platforms, including PHP. For some, the challenge isn't about just running a different web platform; it is about running legacy code in a better way. Azure also supports spawning processes and calling into native libraries with p/invoke so that developers can squeeze all the power out of those massive eight way servers you can use.

7.1 Enabling Full Trust Support

When any code is run in Windows, it is run in a certain trust level. This trust levels defines what the code is allowed to do. For example, code in a partial trust level will not be able to access local hardware and system resources, whereas code running in a full trust environment will have access to just about anything. Trust is only one piece of the equation. The user account permissions must allow an operation, in addition to the trust level. This trust level is enforced by CAS. CAS is a way to define what an application is allowed to do, and is applied at the CLR level. A CAS policy can determine what methods and libraries you use, and what level of local system access your code has access to. ASP.NET comes with several standard trust policies, expressed in CAS. One of them is the ASP.NET Medium Trust policy, and restricts the application to just being able to run itself, without any access to the broader system at play. The Azure team took this policy, and tweaked it to fit their needs, and published a modified medium trust policy.

You should try to run in the lowest trust level possible. This minimizes the damage that can be done if your application is hijacked, or if your code runs amok. This concept is called 'least privileged' and refers to always

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

running your code with the least amount of privileges needed to get the job done. If your code doesn't need access to the registry, it shouldn't have access. This restricts what the bad guys can do if your system is compromised, or the damage you may cause if some code in your application becomes self-aware and start to run amok. Bad things might include placing files in the system folders, changing your desktop wallpaper to LOLcats, or writing naughty things to the registry.

You should only run your code in full trust when you absolutely have to. Unfortunately this is required for any unmanaged code you may want to run and to access the Azure diagnostics systems.

There are times however when your application legitimately needs advanced permissions. It could be because you are referencing a library that requires them, or you are accessing the local system somehow. When this is the case you can change the configuration of your cloud service to run in a full trust model. While you are running in a full trust model you have access to do just about anything you want. The identity your application is running under is still a limited user on the server. This is to keep you from creating Windows user accounts, and formatting the hard drive. So while your code is able to run any opcode through the CLR that can possibly be run, the local user permissions are limited, such that you can only do things for your application, and not system wide.

Full trust is enabled by default. To disable full trust, you can set the `enabledNativeCodeExecution` setting to false, as shown in listing 7.1..

Listing 7.1 - Enabling full trust and running unmanaged code is easy.

```
<WorkerRole name="VerifyOrder" enableNativeCodeExecution="false">
    <ConfigurationSettings>
      <Setting name="InboundQueue"/>
      <Setting name="OutboundQueue"/>
      <Setting name="AccountName" />
      <Setting name="AccountSharedKey" />
      <Setting name="QueueStorageEndpoint" />
      <Setting name="TableStorageEndpoint" />
    </ConfigurationSettings>
  </WorkerRole>
```

Full trust is the doorway to running just about any code you can think of. Another option that will be released soon is the ability to run your own virtual machines in the Azure data center. There isn't a lot of information about this yet as it was just announced at the PDC2009, but it promises to let customers run any machine they have today up in the cloud, including any off the shelf applications.

7.2 FastCGI in Windows Azure

FastCGI is a module in IIS7 that provides a way to run CGI based applications. CGI is a standard interface that modules can be written to, to plug into any web server. The web server then pipes each request through these modules, letting each module work on responding to the request. Sometimes it is a simple static resource module that immediately responds with an image file. Other modules execute a whole application to respond with HTML. This is how ASP.NET and PHP work. IIS7 supports this standard with its FastCGI module that is run in the IIS7 pipeline, when it is configured properly.

In order to take advantage of this in Azure we first need to configure the FastCGI support for your development workstation, and then configure your web role to enable the PHP interpreter. Once we do this our Azure application will be able to host a PHP application, and respond to PHP requests from web clients.

7.2.1 Enabling FastCGI in your local cloud environment

The first step we need to take to run PHP in Azure is actually a local step. If we want to run and debug PHP running in the local cloud environment, we need to reconfigure the local instance of IIS.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 7.x – If you use Windows 7 for your development workstation you will need to enable CGI support for your local instance of IIS. You can do this by using the Windows Features applet in the control panel.

The readme for the SDK will have all of the gory details for enabling FastCGI. The short steps are to enable the CGI feature of the Web Server role if you are running Windows Server 2007. If you are running Vista or Windows 7, you will need to enable the CGI feature in the Application Development Features group in the Windows Features applet.

7.2.2 Configuring Azure for FastCGI and PHP

In order to run FastCGI in Azure you need to do more than just tweaking one setting in the service definition, but it isn't too bad. You will need to include a new file in your web role project call the web.roleConfig file. It needs to be in the root of the web project, and needs to be modified from the default contents when added.

Listing 7.2 - The web.roleConfig configures IIS in Azure to allow for the use of FastCGI

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.webServer>
    <fastCgi>
      <application fullPath="%RoleRoot%\approot\myinterpreter.exe"/>
    </fastCgi>
    </system.webServer>
</configuration>
```

Once you add the `web.roleConfig` (a sample is in Listing 7.2) you will also need to include the CGI interpreter for your platform in the project. Both the interpreter and the `web.roleConfig` file need to have their build action changed to content. This will tell Visual Studio to do nothing with the files, but include them in the output package that is uploaded to Azure. The `fullPath` attribute needs to be modified to point to the interpreter that you include in your project.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

At this point you will have enabled FastCGI, and uploaded and enabled the interpreter (perhaps PHP) that IIS7 will use to execute your web pages. We still need to tell IIS7 what type of requests should be routed to this new interpreter. Should it be every request, or only requests that end in .php. These routing instructions are called handlers, and are configured in the plain old web.config of your web project. As an example, the handler needed for PHP is shown below.

```
<add name="PHP with FastCGI" path="*.php" verb="*" modules="FastCgiModule"
scriptProcessor="%RoleRoot%\php\php-cgi.exe" resourceType="Unspecified" />
```

The handler definition defines a name for the handler and the 'path.' The path is what form of requests should be routed to the interpreter the handler is configuring. In this case, any request to the web server that ends in `.php` is sent to the `FastCgiModule` for processing. The FastCGIModule then passes it on to the PHP interpreter for processing. All file types have a handler in the configuration. This is how requests for .aspx are executed by ASP.NET, and how requests for .gif, .jpg and other simple files are routed to the static file handler. When writing the configuration, keep in mind you need to use the macro `%RoleRoot%` to point to the root of where your files will be running from. Currently this is a very small drive called E: in the Azure server. Don't rely on it always being E:. It could change, and that is why the `%RoleRoot%` macro is provided.

The handler configuration needs to be added to the `handlers` section of the `system.webServer` part of the web.config file for your web project for IIS to setup the handler and start accepting requests.

7.2.3 Setting up HelloAzureWorld.php

This isn't really a book on PHP, so our sample is going to focus on simply getting some PHP to work in the cloud. We will leave what to do with PHP up to you, and perhaps another book. Our simple application is going to say "Hello Azure World" and display some server information, as you can see in figure 7.1.

Figure 7.1 - The result when we are done with our Hello Azure World PHP web application

To build our HelloAzureWorldPHP application we are going to create a new Cloud Service solution in Visual Studio. Instead of adding a web role to the solution, we are going to add a CGI web role. I named my HelloAzureWorldPHP, as seen in the fabulous figure 7.2. The CGI Web Role template includes the typical changes you will need to run a CGI based application in Azure.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 7.2 - Selecting a single web role for our PHP web application. To use FastCGI we should choose the CGI Web Role template, which comes prepackaged with the web.roleConfig file we will need.

The project will already contain a web.roleConfig file, with some notes on how to modify it properly. It will remind you to enable native code execution, and to change the path to the interpreter executable (php-cgi.exe in this case). You can download the right version of the PHP interpreter from windows.php.net. I recommend putting it in a subfolder of the project. Don't forget to mark any non-.NET files that you add as `Content` in the build action for that file.

The next step is to make some changes to the web.config file. All of these changes will be in the `system.webServer` section. Any lines in this section are used to configure IIS7, removing the need to use the IIS management tool, or to logon to the server. This is great, because now you can put the required web server configuration needed into the configuration for the web application and it will travel with your application to whichever web server you deploy it to.

The changes you need to make are in listing 7.4. There are basically two changes we need to make.

- The first is we want to add a default document section, so that new visitors come to the home page for our web site properly.
- The second is adding the configuration for the handler, to tell IIS want to do with any web requests that involve PHP.

Listing 7.4 - Required changes to the web.config file

```
<system.webServer>
      <defaultDocument>
        <files>
          <add value="index.php"/>
        </files>
      </defaultDocument>
.
.
.
 <handlers>
.
.
. <add name="FastGGI Handler"
            verb="*"
            path="*.php"
            scriptProcessor="%RoleRoot%\php\php-cgi.exe"
            modules="FastCgiModule"
            resourceType="Unspecified" />
 </handlers>
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
</sytem.webServer>
```

Our project is now setup, and we can start adding some real PHP code. Add a text file to your web project and name it `index.php`. We are going to add the code that you can see in listing 7.5. Listing 7.5 - The simple PHP code we will use in index.php

```
<html>
 <head>
  <title>PHP in Azure</title>
 </head>
 <body>
  <?php echo '<p>Hello Azure World</p>'; ?> <br />
  <?php phpinfo(); ?>
 </body>
</html>
```

This code does a few basic things. As required by the Union of Demo Code Developers, we have to somehow write to the screen some reference to "Hello, World." In this case we are just going to use `echo`, and write out `<p>Hello Azure World</p>`. Lower down we are going to use the famous `phpinfo()` function. This function will write out a bunch of diagnostic information on the version of PHP you are running, and information about your web server. This is a common function to run when you first install PHP on a server, to confirm that everything is working properly. If you deploy this to the cloud, and browse the results you will find some interesting facts about the server running your application. The user name you code is running under is a GUID, and your web site files will be stored on E:\.

Running PHP in Azure is pretty easy once you setup a few options. PHP on Azure has proven to very popular. Microsoft and WordPress announced they were working together at the PDC in 2009, and ICanHazCheeseburger announced they have been running some of their sites with WordPress running on PHP on Azure. Now that we have that under our belts, we can start looking at spawning processes in Azure.

7.3 Launching external processes in Windows Azure

Some enterprise applications rely on child processes to be run in an asynchronous way. As a way to show how to work with external processes we are going to build a small website that converts videos to different video formats. Since we don't want to write all of the video conversion code, we are going to leverage FFmpeg.

FFmpeg is an open source project that provides for a cross platform way to play, convert, and stream media files. You can download it at FFmpeg.org. We are using the Windows compatible binaries. In the package you will find an executable called FFmpeg.exe. This is the core executable that we will be using.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 7.x - A screen shot of the Simple Azure Video Converter, which uses the open source FFmpeg to convert media files. Our web role will call out to the FFmpeg application by launching it as an external process.

The application will provide a simple way to browse videos in your blob account, and choose to convert one by providing a destination name. The extension of the new file name will tell FFmpeg what format to convert to.

We have several versions of the Big Buck Bunny trailer already uploaded to our blob account. Big Buck Bunny is part of the open movie project, and is released under Creative Commons. You can download the whole movie at www.bigbuckbunny.org.

If you enter a destination filename, for example SmallMovie.mpg, the web site will create a new background process, execute FFmpeg on the big movie, and then place the new file, after it is converted into a mpg, back into the blob account.

We can't run this program in the process that is executing our .NET code, so we will spawn a new process, and have it execute the application for us. The SDK has some great sample code for how to spawn this process, and interact with it. I just use the sample as it is, with some tweaks based on what I am doing. You can find it as part of the Full Trust demo, in the ExecuteProcess.cs.aspx file. The method is called Run.

The Run method provides ways to capture any output from the process, so that you can use it in a way that makes sense for you. In the sample application for this section, we have chosen to capture some basic facts about what has happened, and pipe that to the Azure log. Our shortened Run command looks like the following. In our project we named it ExecuteCommand.

```
protected int ExecuteCommand(string cmdPath, string arguments)
{
     var process = new Process();
     var startInfo = process.StartInfo;
     startInfo.UseShellExecute = false;
     startInfo.CreateNoWindow = true;
     startInfo.FileName = Server.MapPath(cmdPath);
     startInfo.Arguments = arguments;
     startInfo.WorkingDirectory = [CA]Path.GetDirectoryName(startInfo.FileName);
     startInfo.RedirectStandardError = true;
     startInfo.RedirectStandardOutput = true;

     process.Start();
     process.BeginErrorReadLine();
     process.BeginOutputReadLine();
     process.WaitForExit();
     var elapasedTime = process.ExitTime - process.StartTime;
     RoleManager.WriteToLog("Information", String.Format("Command: {0} [CA]{1}", cmdPath,
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
arguments));
     RoleManager.WriteToLog("Information", String.Format("Exit Code: {0}",
[CA]process.ExitCode));
     RoleManager.WriteToLog("Information", String.Format("Elapsed Time: [CA]{0}",
elapasedTime));     return process.ExitCode;
}
```

In this method we are starting up a new process on the operating system, and telling it to run our executable. The path to the program to run is set to the `FileName` property, while any command line arguments you might want to pass are set to the `Arguments` property. The process isn't actually started until the `Start` method is called. At the end of our method we are capturing some run time information, and saving that to the Azure log.

When running in the cloud you can spawn both 32 bit and 64 bit processes. This makes it easy to move legacy code into the cloud. You can do this in the local development fabric as well, unless you are running a 32 bit machine. In that case you will only be able to spawn 32 bit processes when you are running locally.

You can start multiple processes in your role, but keep in mind that each new process takes up memory and CPU time. Start too many of them and your instance won't be able to get any real work done.

Like most applications you are likely to spawn, FFmpeg only works on files stored on the local file system. To accommodate the application we are going to copy the movie from blob storage to the local storage.

The `CopyBlobToLocalStorage` method works by getting a stream from the blob, and writing it out to a file stream that is pointed to local storage. The first few lines get a memory stream, `videoStream`, from the blob contents reference. We then use that to setup a byte array to use as a buffer from the memory stream to the file stream. Inside the using statement we create the file stream, and have it write out the bytes from the buffer.

Listing 7.6 – Copying a blob to local storage

```
private void CopyBlobToLocalStorage(string inputName, BlobContainer videoContainer)
        {
            if (videoContainer.DoesBlobExist(inputName))
            {
                MemoryStream videoStream = new MemoryStream();
                BlobContents videoContent = new BlobContents(videoStream);
                BlobProperties videoProperties = videoContainer.GetBlob(inputName,
videoContent, false);

                int length = Convert.ToInt32(videoStream.Length);
                int bytesRead = 0;
                byte[] buffer = new byte[length];

                using (FileStream fs = new FileStream(localDisk.RootPath + inputName,
FileMode.Create))
                {
                    while (bytesRead < length)
                    {
                        bytesRead = videoStream.Read(buffer, 0, length);
                        fs.Write(buffer, 0, bytesRead);
                    }
                }
            }
            else
                throw new Exception("input movie not found");
        }
```

Once the movie is copied down to the local storage, we can run the FFmpeg command. If we were running FFmpeg at the command line, we would use something like

```
FFmpeg.exe -i BigBuckBunny_Trailer_400p.ogg -y SmallerMovie.mpg
```

To execute this as a process, we call the `ExecuteCommand` method with the following code. This will spawn the process, execute the FFmpeg, pass in the parameters, and wait for it to finish executing.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
string VideoArgs = string.Format(@"-i {0} -y {1}",
            [CA]localInputFilename, localOutputFilename);
ExecuteCommand(@"ffmpeg\ffmpeg.exe", VideoArgs);
```

This command results in a new movie file, of the proper type, in the same local storage folder where the source movie was. Our next step is to copy it back into blob storage.

```
using (FileStream fsUp = new FileStream(localDisk.RootPath + outputName,
[CA]FileMode.OpenOrCreate))
{
     videoContainer.CreateBlob(new BlobProperties(outputName), new [CA]BlobContents(fsUp),
true);
}
```

This will create a new blob in our videos container with the same name as the user asked for in the web application. Since the `CreateBlob` method takes a stream as a parameter for the contents of the new blob, it is easy to pass in a `FileStream`. When the file is finally uploaded, we clean up after ourselves by deleting both local files.

Spawning processes is an important option to have, but you should focus on using it when you are migrating an application to Azure that relies on an external dependency. You shouldn't intentionally architect a new system to use this feature. In that situation you should probably use inter-role communication between different role instances. This gives your solution more flexibility when it is time to scale.

7.4 Calling Native Libraries with P/Invoke

We have looked at two ways to leverage native code in Azure, FastCGI and spawning processes. The third option at your disposal is to call into a native library with p/invoke. This allows you to call a native library directly, such as a Windows DLL. P/invoke is shorthand for 'platform invoke', or calling a platform API directly. If you work mostly in .NET, then you are rarely calling the platform API directly, and instead are using classes out of the .NET framework or the Base Class Library.

The limitation with p/invoke is that you can only call 64 bit native libraries while running in the cloud. If you happen to be developing on a 32 bit machine, you will be able to call a 32 bit library locally, but this will not work in the cloud.

A work around to this limitation is to spawn a 32 bit sub process, as outlined above, and call the p/invoke from there. We think this is too much work, and you should only stick with 64 bit libraries.

Calling native libraries is a special skill to begin with, so we aren't going to cover everything you need to know about doing that in .NET. Basically, doing so in the cloud is a lot like when you do it locally.

When working with native libraries, you need to first import the DLL into your namespace, and then provide a façade method into the native libraries method. This will involve a great deal of code that will map native data types to .NET CLR types.

Don't forget to allow native code execution in your cloud service definition file; otherwise you will receive a security exception.

The easiest way to implement the signatures you need in your code is to get them from http://www.pinvoke.net like we do. This website provides the signatures for just about any native call you could possibly want to make.

We are going to build a simple web application for Azure that will list out the files and directories in a given folder. You could easily build this with managed code, without p/invoke, but we thought it would make for a good example.

The first step is to add the interoperability services namespace to your project;

```
 using System.Runtime.InteropServices;
```

Then we are going to import the native methods that we want to call.

```
[DllImport("kernel32", CharSet = CharSet.Unicode)]
public static extern IntPtr FindFirstFile(string lpFileName, out WIN32_FIND_DATA
lpFindFileData);
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In this example code we are importing a method called `FindFirstFile` from the DLL called `kernel32`. This line will define a series of parameters you will need to provide to the method for it to work. The first, `lpFileName`, is the path of the folder we want to look in, and the second, `lpFindFileData`, is variable that the results will be stored in. This method will also return a handle to an object that will be our pointer into the file system. We will use this handle to iterate, by calling another imported method call `FindNextFile`, through the folder we are pointing at. You should always remember to close any handle objects you are using while working with native libraries. Otherwise they will be left open in memory, causing memory leaks.

It will often be the case that the method you are importing requires input and output parameters, and return values that use a type that is not supported in .NET. In these cases you will need to define these types so that they can be used. An example would be the structure that is used to define the data about the file the finder just found.

Llisting 7.7 – Defining a data type to work with a native library

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
        public struct WIN32_FIND_DATA
        {
            public FileAttributes dwFileAttributes;
            public FILETIME ftCreationTime;
            public FILETIME ftLastAccessTime;
            public FILETIME ftLastWriteTime;
            public int nFileSizeHigh;
            public int nFileSizeLow;
            public int dwReserved0;
            public int dwReserved1;
            [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 260)]
            public string cFileName;
            [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 14)]
            public string cAlternate;
        }
```

Once these types are defined you will be able to call into the method just fine. In our example we are going to make the initial `FindFirstFile` call, and perform a `Do` loop until `FindNextFile` comes back empty. With each iteration of the loop we will copy data from the return data structure to a class, `myFileData` that we define. We will then bind a collection of `myFileData` objects to a simple gridview on the webform.

Listing 7.8 – Looping over the files that are found

```
hFileFinder = FindFirstFile(currentPath, out foundFile);
   do
   {
      fileList.Add(new myFileData()
      { Filename = foundFile.cFileName,
        Filesize = foundFile.nFileSizeLow,
        isDirectory = (foundFile.dwFileAttributes &
[CA]FileAttributes.Directory) != 0 });
            }
            while (FindNextFile(hFileFinder, out foundFile));

         FindClose(hFileFinder);

         gvFileList.DataSource = fileList;
         gvFileList.DataBind();
```

Working with native libraries and p/invoke can be complicated. You should only look to using this when you can't possibly do what you need to do with a class found in the .NET library. The use of the native libraries can introduce brittleness into your solution by creating external dependcies.

7.5 Summary

Microsoft has provided broad support for running just about anything on Azure. They didn't take the easy way and limit cloud developers to just .NET code. They did this because they want it to be a usable platform with

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

broad adoption. The ability to run FastCGI, spawn processes, and call native libraries make it easier for us to port existing applications, and to support a broader array of applications.

With Azure's support for FastCGI we can leverage any CGI compatible module in Azure. This includes PHP, Ruby, and many other web platforms. With just a few simple steps we can deploy one of those platforms to your web roles in Azure.

Spawning external processes is important when you have an external dependency in a system you might be migrating to the cloud. You can also use it to manually parallelize your application.

Some applications that are built with unmanaged code need to access native libraries for system level access. Using p/invoke to access these libraries is available in Azure, but it can be complicated. You should only leverage this feature if you truly need it.

Azure isn't just for Web 2.0 web applications with a viral nature; it is also for serious enterprise applications. Those applications often come with a legacy aspect, whether it is calling into a home grown DLL that can model and calculate the air speed of a laden swallow, or being able to leverage a forum for your website that happens to run in PHP.

What we have covered in this chapter, especially the native calls and process spawning, are great tools, with great power. And with great power comes great responsibility. Make sure you use them wisely, or you will spend a lot of your weekends figuring out why your application isn't working properly.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

8

The Basics of BLOBs

In this big fat blob of a chapter we learn:

- How we share files today in our current retro systems
- How Windows Azure allows us to store files (woo hoo, go Azure)
- How to develop against the Blob Storage Service

In case you didn't bother reading the blurb at the beginning of Part 3, in this chapter (and the next couple of chapters), we will be looking at how we can store files in Windows Azure's highly scalable, fault tolerant, binary file storage system (otherwise known as the Blob Storage Service).

Binary Large Object

BLOB actually standards for Binary Large OBject. The term has been stolen from the world of relational databases where it used to describe the storage of binary data (such as an image or an MP3 file) in a single entity. I personally wished they used BinLob as the acronym. It more accurately describes what happens when a DBA discovers you stuck a terabyte of data in a single row of their precious database.

In this chapter we will be looking at:

- Why storing files in a typical scaled out system is hard?
- How the Blob Storage Service addresses typical scaling issues
- How the Blob Storage Service works
- How you can get your tools out and start developing against it
- How to store Blobs in the production system

So before we can appreciate the beauty of the Blob Service, we need to get a little insight on how we might solve the problem of storing files in a scalable fashion that can be accessed by multiple servers.

8.1 Storing files in a scaled out fashion is a pain in the NAS unless you have plenty of cash

No matter what hat I put on (Author, Presenter, Architect, Developer or Computer Scientist) I am embarrassed about the following statement. Sharing files across machines is errrghh, hard. Really, it is, it shouldn't be (it really shouldn't), but it is. Decoding the Genome and making robots climb stairs, that should be hard but not this, not sharing files that just shouldn't be hard.

To contextualize the problem, let's return to the podcast example that we introduced in Chapter 1. In this scenario we wish to provide a service where we allow our users to upload podcasts that will be converted from MP3 to WMA. In order to support the predicted demand, we have decided to load balance our website across 2 servers. Since our users can upload or download a podcast from any server, a shared storage solution is required.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 8.1 shows a logical representation of 2 load balanced web servers accessing a podcast from a shared storage mechanism.

Figure 8.1 – 2 load balanced servers that require access to common storage area to server up mp3 files

To be honest you don't need to be greatest architect in the world to draw the solution shown in Figure 8.1. It's pretty logical common sense stuff. 2 Web Servers access a common storage area.

Easy peasy, stoopid author, saying this is hard. Get me another book where it says it's easy. Well before you start reaching for Mavis Beacon teaches Windows Azure, let me fire some bullet points on why this is a little harder than it first seems:

- Do you have enough space to store all the files you need?
- How do I add more storage capacity?
- Is the data duplicated anywhere, what happens if a disk crashes, do I lose my data?
- Is the storage block load balanced?
- What if lose my connection to the block, is it redundant?
- At what point do I max out my disk in terms of reading and writing?
- How do I evenly distribute load across all disks?

The good news is that pretty much all of these problems have been answered and solved already. You can even implement these solutions in your traditional non cloud environments today (well there is probably longer lead time than a day). The bad news is that the cheap simple solutions are typically not scalable or fault tolerant. The solutions that are scalable and durable are typically very expensive. In the Windows Azure Blob Storage Service that all changes.

Before we look at how easy it is to store and access files (in a scalable durable fashion) across multiple servers in Windows Azure, let's look at some of the options outside of Windows Azure.

8.1.1 Traditional approaches to Blob Management

Over the next few sections we'll look at how you might provide a file storage facility in traditional ASP.NET web server farms using our podcasting example. We will specifically look at the following technologies:

- Network Share
- SQL Server
- Distributed File System
- Network Attached Storage (NAS)
- Storage Area Networks (SAN's)

So let's start of one with the typical developer solution to the problem, the database.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

SQL SERVER

Since web servers typically have access to a shared SQL Server database we could store our podcasts in a table. Although this is a common approach used in many solutions, this is probably not the best use of your very expensive database server. It's like racing a truck in a Grand Prix; there are cheaper, simpler, higher performance and more appropriate solutions for storing files.

Also, unless you are using a high availability technology (such as clustering, mirroring or replication) then your database server is likely to be a single point of failure in the system. In figure 8.1 SQL Server would be represented by the storage block (accessed over a typical network connection).

SHARED NETWORK SHARE

Another common approach to providing a shared file system across web servers is using a shared network drive that can be accessed by all instances of the website. This low cost solution is more lightweight than a database but it still introduces a single point of failure. This cheapo solution offers no redundancy and provides no ability to scale out. In figure 8.1 an application server with a network share would be represented by the storage block.

Now that we've looked at some of wacky solutions, let's take a look at some of the typical solutions that are used, starting with Distributed File Systems.

DISTRIBUTED FILE SYSTEM (DFS)

Windows Server 2003/2008 provides a technology known as DFS which allows you to create a Peer 2 Peer (P2P) file system on your network. This means that when a new podcast is uploaded, a copy of the file would be replicated to all other participating servers. Although this approach requires no new hardware, it is complicated to manage and adds extra performance overhead to all servers involved.

Figure 8.2 shows a DFS solution with a peer to peer network between 2 web servers.

Figure 8.2 Two Web Servers reading and writing files to a local shared disk. Files are replicated between each server.

Whenever a file is uploaded to a web server then it will be automatically replicated to all other servers in the farms. Using replication ensures that there are no single points of failure in this solution and the data is held on multiple machines. In Figure 8.2 Podcast01.mp3 is uploaded to web server 1 and then replicated to web server 2, and when Podcast02.mp3 is uploaded to web server 2, it is then replicated to web server 1.

In Figure 8.3 the web servers do not hold the files locally but use a replicated file store held in application servers.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 8.3 – 2 Web Servers reading and writing files from a set of replicated file servers.

In Figure 8.3 you can see that Podcast01.mp3 was uploaded to app server 1 via web server 1; the file was replicated to app server 2, and then served up to the client from app server 2 via web server 1.

With file replication, anytime a file is uploaded to a server there will be a small delay between the file being uploaded and it being replicated across all servers. It is therefore possible that the web user could be load balanced onto a server where the file is not available as it has not been replicated across. Although this issue can be alleviated by using sticky sessions, sticky sessions will not help if the original server keels over. Also be aware that the use of sticky sessions will mean that incoming requests will not be evenly distributed across all web servers.

Sticky Sessions

A sticky session is where a load balancer will forward all incoming requests to the same server for the period of the session.

Although Distributed File Systems are typically not used in Windows Environment, this is a popular type of solution in UNIX / Linux environments. Now that we've looked at some of the "hook some machines together" type solutions, we will now look at some of the dedicated disk array type solutions that are typically used in the market.

NETWORK ATTACHED STORAGE (NAS)

A Network Attached Storage device is a disk array that you can plug into your network and can be accessed via a network share. NAS devices are responsible for managing the device hardware, the file system, serving files and can provide varying levels of redundancy depending on the device and the number of disks in the array.

Although NAS devices reduce load from client operating systems by taking responsibility for file management it has no ability to scale beyond its own hardware. NAS devices can range from being pretty cheap to very expensive depending on the levels of scalability, performance and redundancy that you require from the device. In figure 8.1 the storage block would be represented by the NAS device (connected via the Ethernet).

NAS devices are typically used to provide file server type capabilities rather than being used as a disk management system in a high performance application solution. Let's us now take a look at storage solutions that are commonly used in application solutions

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

DIRECT ATTACHED STORAGE (DAS)

A Direct Attached Storage device is a disk array that you can plug directly into the back of your servers and can be accessed natively by the server. DAS devices are responsible for managing the device hardware, and can provide varying levels of redundancy depending on the device and the number of disks in the array.

Since DAS devices are directly connected to a server it's therefore treated as if it's a local disk meaning the server is responsible for the management of the file system. DAS devices can support large amounts of data (100TB or so), can be clustered (i.e. no single point of failure) and are usually very high performance systems. As such DAS devices are a common choice for high performance applications. The choice of device can range from being pretty cheap to very expensive depending on the levels of scalability, performance and redundancy that you require.

Although DAS devices are great they are limited by the physical hardware. Once you reach the physical limits of the hardware (which is quite substantial), you will be able to scale no further.

In figure 8.1 the storage block would be represented by the NAS device (connected directly to the servers).

STORAGE ATTACHED NETWORK (SAN'S)

Like DAS devices, SAN's are also separate hardware disk arrays and as such, they do not have their own operating system and file management is therefore performed by the client operating system.

SAN devices are represented on the client operating system as virtual local hard disks which are accessed over a fiber channel. Since we require our web servers to access shared data, the SAN would need to support a shared file system. In figure 8.1 the SAN device would be the storage block attached to the web servers via fiber channels.

SAN's are usually very expensive, require specialist knowledge and are rarely used outside of the enterprise domain. To give you a clue on how expensive, even Dell don't list the price on their website. As for installing and managing SAN's well that is purely in the domain of the long haired sandal wearing beards. We mere mortals have no chance of making those things workSAN Devices are highly scalable (scale much higher than DAS devices), fault tolerant, support replication, very high performance and very expensive. Due to the performance, price and scalability, this is the solution of choice in the Enterprise space. The rest of us can only dream.

So hopefully I might have justified my earlier premise that implementing a file storage solution today is not as easy as it first looks, as we have discovered all choices (beyond a certain size) require the following:

- IT Pro Knowledge / Skills
- IT Management
- Large amounts of cash or a tradeoff between capacity, redundancy, ability to scale or performance

Now that we have seen the solutions and issues to storing files in traditional on-premise solutions, let's now look at the Windows Azure Blob Storage Service and how it tackles these issues.

8.1.2 Blob Service approach to file management

As we discovered earlier, the Blob Storage Service is the Windows Azure solution to providing file storage. Let's take a look at how this is implemented.

API BASED SERVICE

Rather than building a native network share based solution, Microsoft has provided a set of REST based API's that allow you to interact with all the storage services over the http stack using a standard http request. As mentioned earlier not only can use these API's inside the data center but they can also be used outside of the data center.

Data Center Speed vs Internet Speed

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Although you can upload and download files outside the Data Center, you will be subject to Internet speed, i.e. it may take you a few hours to upload / download Gigabytes of data. Within the Data Center you can copy Gigabytes of data between Blob Storage and a Worker/Web Role in seconds. This massive speed difference is due to co-location of the Storage Service and the Roles.

SCALABILITY

Using http as the underlying transport layer meant that the Windows Azure could leverage the web role infrastructure within Windows Azure to host the storage services. By using the Web Role infrastructure to host their storage service (tens of thousands of instances), it can give you confidence that your application will be able scale to that level. Figure 8.4 shows the abstraction of web instances for the blob storage service.

Figure 8.4 Scaling of Blob Storage Services within Windows Azure

Since Blob Storage is built upon the Web Role infrastructure they too can get the advantages of utility computing. As the demand for the storage services increases, Microsoft can ramp up the number of instances just like any other web role meaning we never need to worry about the scalability of any of the storage services (unless Microsoft runs out of pennies).

DISK STORAGE

Just as there are thousands of racks of machines used to host the web and worker roles, there are just as many disk arrays storing our data! Microsoft can grow the storage required within the data center by adding more disks as and when required. This level of enterprise class storage means that we never need to worry about capacity or scale. We can think of the Blob Service as a giant virtual hard disk that will always scale up to meet our demands and never run out of space.

DATA CONSISTENCY WITH REPLICATION

Like the Distributed File System solution Windows Azure Blob Storage is also a replicated solution (to be honest you have to be to achieve massive scale). Now although the Blob Service is quite similar to the Amazon Simple Storage System (Amazon S3), replication is one of the areas that it differs.

With Amazon S3 there is no consistency of data throughout the data center. If you upload a file to Amazon S3 and then request that same file, it is likely that a different server will process that request. Due to network latency the file will probably not be available to the new server as the data will not have been replicated from the original server yet. Amazon S3 suffers from the same issues seen with Distributed File Storage earlier.

This issue of replication latency can never occur within Windows Azure Storage services as it guarantees a consistent view of your data across all instances that may serve your requests. Internally within Windows Azure

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Storage Services as soon as data is written to your storage account it is replicated throughout the data center. Every piece of data must be replicated at least 3 times as part of the commit process.

As your data is being replicated across the various disks within Windows Azure the fabric controller keeps track of what instances can access the latest version of your data. The Load Balancer will only route requests to an instance that can access the latest version ensuring that stale data is never served.

Even if a disk failure occurs straight after the upload no data loss will occur as other disks are guaranteed to receive a copy of that data.

So far we talked about how Blob Storage solves the problems of scalability and fault tolerance but we haven't talked about performance. Surely performance is going to suck; it is effectively a REST based web service after all.

PERFORMANCE

Sure, the performance of Blob Storage in comparison to SAN's or DAS's does suck. Ultimately that tradeoff between performance, fault tolerance and scalability meant that performance lost. However within the Data Center, it is generally good enough performance. Because the service is ultimately a load balanced Web Serve, you can expect a 50ms to 100ms latency between your role and the storage service. Although the latency is poor, the network connection is fast, so you can expect good enough performance. Sure, you wouldn't allow an application that needs to write to disk very quickly (e.g. SQL Server) to write directly to blob storage but not all applications need that speed.

If you do need that level of speed then you can always cache files locally on your role using Local Storage. This technique will typically give you much more acceptable performance for your application. In-fact this is exactly what the X-Drive feature uses to ensure performance.

> **So what is X-Drive?**
>
> Although the REST API is very flexible and gives us great scale, it's no substitute for a good old file system. So to make life a little easier for those bits of code that are used to talking to directories and files rather than web service, Microsoft have provided a new feature called X-Drive, which allows you to mount blob storage as an NTFS Drive. This allows you to access blob storage just like any other drive. Since this feature is implemented using a special OS driver that was developed specifically for Windows Azure, this feature is only available to your roles and not available outside of the data center.
>
> As cool as X-Drive is, it only allows one instance of a Role to read and write to the X-Drive (however all other drives can still read from it). In the next chapter we will look at this in a little more detail.

So now that we looked at how Blob Storage handles the issues that we face in traditional on-premise solutions, it's worth looking at it from a data management perspective.

MANAGEMENT

One of the most compelling arguments for the storage services is that IT pro management skills are not required. In traditional systems a large investment in IT management skills is usually needed to support storage.

Management of the storage arrays usually requires very expensive specialists that are capable of supporting the data such as SAN Experts, Network specialists, Technicians, Administrators and DBA's.

To plan such a system those experts will need to be able to design and implement the infrastructure taking disk management, fault tolerance, networking, lights out operation and data distribution into consideration.

The day to day running of the system will include hardware replacement, managing backups, optimizing infrastructure, health monitoring and data cleansing amongst others endless tasks.

With Windows Azure we let Microsoft manage the storage systems and we can concentrate on using the system via familiar developer API's allowing us to focus on our core skill set of building software.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

8.2 Taking a closer look at the Blob Storage Service

So now we have an idea how the Blob Storage service is hosted in Windows Azure let's look at how files are stored within the service. In this section we'll look at the 3 layers of Blob Storage namely:

- The Account
- The Container
- And The Blob

To help explain these concepts we will use figure 8.5 as a reference. Figure 8.5 shows how an MP3 file may be typically stored in Blob Storage.

Figure 8.5 Podcast01.mp3 is stored in the ChrisOriginals Container in the silverlightukstorage account

Before we get all technical about accounts, containers and blob's, it's probably worth remembering this. An account is simply your account, Dave will have an account, Jim will have an account and you have an account. An account is really about ownership. A container is just somewhere you can store your blobs. Containers are really about access control (public or private access) and some level of organization.

With that in mind let's look at some of the specifics.

8.2.1 Accessing the Blob (file)

In figure 8.5 we can see how files (otherwise known as blobs) are stored within blob storage. In figure 8.5 the blob Podcast01.wma resides in the container chrisconverted, which resides in the storage account silverlightukstorage. If you notice that a blob cannot directly reside within a storage *account* and must live in a storage container (if you do need to make available as if it's at top level of the account i.e. as if it doesn't have a container, you can store the blob in the root container, we will explain this in more detail in chapter 10).

Since storage services are a REST based architecture, we can retrieve a file from blob storage by performing an HTTP GET request to the URI for the blob. The following URI would allow us to retrieve Podcast01.wma from the ChrisConverted Container held in the silverlightukstorage storage account from the live blob storage service:

http://silverlightukstorage.blob.core.windows.net/ChrisConverted/Podcast01.wma

We could formalize the URI for the live storage account as follows:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

http://<storageaccount>.blob.core.windows.net/<Container>/<BlobName>

Let's now take a closer look at accounts, containers and blobs to get a clearer understanding of these components.

8.2.2 Setting up a storage Account

On signup for Windows Azure you are able to create a storage account within the Azure Portal. The storage account is the top level for all storage services (blobs, queues and tables) that reside under it.

On creation of your storage account you will be assigned a sub-domain for each storage service. The 3 domains for the storage services are listed below

- http://<storageaccountname>.blob.core.windows.net/
- http://<storageaccountname>.queue.core.windows.net/
- http://<storageaccountname>.table.core.windows.net/

In our previous example, the name of our storage account is silverlightukstorage, which means the top level Uri for each service in our account would be:

http://silverlightukstorage.blob.core.windows.net/
http:// silverlightukstorage.queue.core.windows.net/
http:// silverlightukstorage.table.core.windows.net/

For now we will just focus on the blob service but in later chapters we will return to the table service and the message queue service.

HOW DO I BREAK UP MY STORAGE ACCOUNTS?

I guess there are a couple of things to consider about your storage account. The major one being do you have one large account, or a separate account for each application? Although this is ultimately up to you, a good guide would be access control. If you are small shop, then one overall account is probably suitable; however a single account wouldn't work so well in say Microsoft or IBM. In these larger organizations separating by application is probably a more suitable approach.

If you don't like the beautiful subdomain assigned to you for blob storage (xxxxx.blob.core.windows.net) then you can always assign your own domain name.

8.2.3 Registering Custom Domain Names

What we will now do is step through the process of associating our own domain name with the blob storage service. This will mean that will be able to access our WMA file using the following Uri:

http://blobs.chrishayuk.com/ChrisConverted/Podcast01.wma

There is pretty much a 3 step process to registering a custom domain name with a blob storage account, which is:

- Register a suitable domain with your domain provider
- Setup a domain to point at Windows Azure
- Validate you own the domain
- Setup the subdomain to point at blob storage

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

To be honest, we are going to skip the registering a suitable domain step. If it really does confuse you, then I'm sure GoDaddy (or some other provider) will happily provide some instructions, so they can extract some lovely dollar bills (or British Pounds, or errrgh Euro Euros) from your pocket.

SETUP A SUITABLE DOMAIN AND VALIDATE THAT YOU OWN IT

Once you have registered your domain (e.g. chrishayuk.com) then you need to let Windows Azure know that you wish to point a suitable subdomain at it. To do that you will need to log in to the Azure Portal. Once you are logged in you should select your storage account (silverlightuk) and then select the manage domains button. On clicking the manage domains button you will be faced with the page shown in Figure 8.6.

Figure 8.6 – Validating that you are domain owner of the domain that you wish to point to blobs in the Azure Portal

Once we have entered the name of the domain (including subdomain) that we wish to point to the blob storage account, we need to validate the domain by clicking the Generate Key button. Once you have clicked the Generate Key button you will be presented with the screen shown in Figure 8.7

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Custom Domain for the Blob Endpoint

Domain Registration

In order to configure a custom domain for your storage account, you need to own a domain. If you do not already own a domain, you can register a new domain with an Internet Service Provider.

Domain Association

Associated domain name: blobs.chrishayuk.com

CName: fb160[redacted]-8f74-bb4[redacted]d2

Copy this key and create a CName record from **fb160c[redacted]-8f74-bb40[redacted]d2.blobs.chrishayuk.com** to **domainnameverification.windows.azure.com**. You will have to perform this configuration step on your domain registrar's website.

Once you have completed this step, please click on validate to continue with the ownership verification process.

Finally, once you have validated your domain, please create a CName record from **blobs.chrishayuk.com** to **silverlightukstorage.blob.core.windows.net** to complete this process.

Domain Ownership Validation

Once you have setup the DNS record on your domain registrar's web site, click on Validate to initiate the domain ownership validation process immediately, or click on Cancel to go back to the Storage Account property page. You will be able to validate your domain ownership at any time.

Delete | Validate | Cancel

Figure 8.7 – Receiving the domain validation CName guid

As you can see from Figure 8.7, we now have two actions to perform.

- Add a new cname for the guid (fb160....) pointing to domainverification.windowsazure.com
- Add a new cname for the subdomain (blobs.chrishayuk.com) pointing to your blob storage account (silverlightukstorage.blob.core.windows.net)

Whichever company you used to register your domain probably manages the DNS for your domain name. Using their web control panel you should be able to create the subdomain using a cname. Figure 8.8 shows the cnames for chrishayuk.com in the GoDaddy Domain Manager.

CNAMES (Aliases) — Reset to Default Settings | Add New CNAME Record

Host	Points To	TTL	Actions
blog	ghs.google.com	1 Hour	
fb160c4b-d159-41ba-8f74-bb40ce4b65d2	domainnameverification.windows.azure.com	1 Hour	
blobs.chrishayuk.com	silverlightukstorage.blob.core.windows.net	1 Hour	
www	@	1 Hour	

Figure 8.8 - the cname entries for chrishayuk.com, notice both the domain verification cname and the blob storage cname are listed

If you manage your own DNS server then you will already know how to set up a cname, if not then your system administrator will certainly be able to (although he might not be very pleased that you are looking to replace him with an automated system).

Once you have setup your cnames, you will need to return to the Windows Azure portal, a little later to validate the domain (clicking the validate button in Figure 8.7). Once the domain has been validated you will be able to use your custom domain name.

Why do I need to come back later?

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

> Well funnily enough this is all to do with replication. Once you have updated the DNS details on the server that is responsible for maintaining your domain records, this update will need to be replicated to all other DNS servers in the world. This replication delay is the reason that you will have to come back later (usually 10 minutes to an hour), as it will take a little time for the Windows Azure DNS servers to receive that update. Perhaps the world's DNS Servers should use Windows Azure instead ;)

Ok, so we have setup our custom domains and we understand containers; let's look at how we can use them to store blobs.

8.2.4 Using Containers to store blobs

In Blob Storage we cannot store blobs directly in a storage account as every blob must live in a container. A container is really a top level folder. Although we can set permissions directly on a blob, with a large amount of blobs this can be a pain. To alleviate that administrative headache you may wish to group similar blobs with similar access levels in the same container. This will then allow you to set permissions at the container level rather than the individual blob.

In Blob Storage there are two levels of access that can be set on a container: private and public.

PRIVATE CONTAINERS

Blobs in a private container are restricted to the owner of the account. If you need to list the contents or download a blob stored in a private container then you would need to make a request signed with your shared authentication key.

In figure 8.5 the container ChrisOriginals is a private container, so if we wanted to access the blob podcast01.mp3 then we would could make the following GET request (as long as the request is signed with either your Account Master Key or a pre-generated shared key, explained later):

http://silverlightukstorage.blob.core.windows.net/ChrisOriginals/Podcast01.mp3

FULL PUBLIC READ ACCESS AND PUBLIC READ ONLY ACCESS FOR BLOBS

If the container is set to Full Public Read Access then you can retrieve any blob held in the container over http without providing authentication credentials. Not only that you can list all the blobs in that container and query data about the container.

With Public Read Only Access for Blobs, anonymous requests will only be able to read a blob (you will not be able to read container data or list the blobs in the container).

In figure 8.8 the container "ChrisConverted" is a public container, so anyone on the internet would be able to download the file podcast01.wma by making an http GET request to the following URI:

http://silverlightukstorage.blob.core.windows.net/ChrisConverted/Podcast01.wma

If you need to perform any operations beyond the container permission level (e.g. if you need to upload or modify a blob) then you would need to provide authentication credentials (account owner or shared access) as these operations are restricted operations.

So far we have only really talked about the live blob storage service, we'll now take some time to look at how we can use develop against the blob storage service using a development version of the blob service in the Development Storage Service.

8.3 Getting started with Development Storage

Development Storage hosts all 3 storage services (blobs, queues and tables) and exposes local endpoints that implement the same API's as the live service. The production version of Storage Services and the

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

development version are two completely different animals. They may expose the same API's but the development version is greatly simplified and only suitable for local development.

Once we have finished developing our application against our local development storage we can easily switch to using the live environment by just changing configuration.

8.3.1 SQL Server Backing Store

Since our development environments and the data centers of Windows Azure are drastically different (we don't have replicated storage arrays on laptops), then the SDK can only really provide a simulation of the live storage environment. Although Development Storage and Blob Storage are API compatible the underlying implementations are understandably different.

In the Development Storage version of Blob Storage, SQL Server is used as the backing store.

Installation Issues

By default the Development Storage database is created in the SQLEXPRESS named instance of SQL Server on your development machine. This instance is normally installed as part of the Visual Studio installation which is why the SDK assumes that this instance is present. If you need or want to install the database onto a different SQL Server instance you can use a tool in the SDK called DSInit.exe. You may wish to do this if you work in a shared development environment or if you skipped installing the SQLEXPRESS instance during the Visual Studio installation.

If you want you can even run queries against the database to ensure your data is stored as you would expect. In figure 8.9 we can see all the tables representing the various Storage Services in the SQL Server implementation.

Figure 8.9 - Development Storage database in SQL Server

Although the Development Storage system uses SQL Server (as shown in Figure 8.9), the real Blob Storage System uses a much higher performance, more scalable, custom solution that makes best use of the Windows Azure infrastructure. You can be assured that your blobs are not stored in some SQL Server table in the live system ;)

8.3.2 Getting around in the Development Storage UI

The development storage service is automatically started up whenever you run a web or worker role project in Visual Studio. The startup of Development Storage occurs at the same time as the startup of the development

um:

fabric as described in Chapter 2. If we click on the Development Storage Icon in the status bar it will display the Development Storage UI as shown in Figure 8.10

Figure 8.10 Development Storage Command and Control Center. You can start, stop, and mess around with all of your storage services from here

The Development Storage UI allows us to see the current status of our services and stop and start them if required. Although Development Storage and the Development Storage UI are automatically launched when we run our application in Visual Studio, we can start them manually using the command line. This can be useful if we are interacting with the storage services from a non-cloud hosted application (i.e. a normal WPF application that just uses the Blob Storage Service).

STARTING AND SHUTTING DOWN DEVELOPMENT STORAGE MANUALLY

To start development storage manually you can use the following command:

```
C:\Program Files\Windows Azure SDK\v1.0\bin\devstore\dsservice.exe
```

To shut down the service you can use this command:

```
.\dsservice.exe /shutdown
```

With the basics about both blob storage and development storage under our belt, get ready! It's time to write our first application that talks to the blob service.

8.4 Developing against containers

Before we start writing some code against the containers, it's probably useful to discuss where would use this type of functionality.

If you are just needing to have a shared storage area where you can read blobs then you will probably eventually use the X-Drive functionality (chapter 10) rather than interacting with these API's directly (however it is still useful to understand how it works, as X-Drive interacts with these API's too).

If you need a scalable application where more than one role needs to write to a single shared storage area then you will need to use the storage client library or the REST API's directly.

Over the next few sections, we will be looking at what sort of operations you can perform against containers. The use of containers is particularly interesting when you need to dynamically create storage areas and assign permissions to different parties in a scalable fashion. Typical scenarios would include:

- File Hosting
- Enterprise Workflows
- Data Manipulation applications that need to access data

In the next chapter, we will look at how we can enable some of the above scenarios by generating dynamic keys.

For now, we will return to our podcast example. We need a method of creating public and private containers to store our original and converted podcasts in. In this sample application, we will create an ASP.NET Web Form page that will allow us to:

- Create a new container (private only for just now)
- List existing containers in our account
- Delete the container if necessary
- In the next chapter, we will look at how we can set permissions on containers

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Our final application should look like the screenshot displayed in figure 8.11

Figure 8.11 - Displaying a list of containers in our storage account using ASP.NET

Over the next few sections we will look at the following topics, which will help us build the web page displayed in Figure 8.11:

- Working with Storage Client Library SDK
- Accessing the Development Storage Account
- Creating a Container
- Listing Containers
- Deleting a Container

At the end of this section you will be able to create the above web page as well as having God like knowledge of containers.

Before we get started we need to create a new Web Role project in Visual Studio (as shown in chapter 1), you should call this web role PodcastSample.

8.4.1 Accessing the Storage Client Library

There are really two ways of the interacting with any of the Storage Services, you can either use the REST API directly or you can use the Storage Client Library assembly. In this book we will look at both methods.

One of the reasons that we will look at both methods is that the Storage Client Library is just a .NET wrapper for the REST API. For new features Microsoft will often release the REST API call before adding the feature to the Storage Client Library. By understanding both methods you will be able to utilize any new feature straight away (if you need to).

Another reason for looking at the REST API is that the underlying mechanism is heavily abstracted away from you. By understanding the underlying calls you can make the best decisions architecturally for your application (especially regarding performance).

If you are currently thinking, well buddy boy, if the REST API is so great, why are we using the Storage Client Library? The answer to that is quite simple, as flexible as the REST API is to use, it's a right royal pain (God bless Her Majesty ;). Using the REST API directly means we get to write unreadable code HttpWebRequest code with no Intellisense. By using the Storage Client Library wherever possible we get to write familiar .NET code (with Intellisense), increasing productivity meaning that we can ultimately spend more time doing more important things (like browsing the Internet or playing Halo).

Although the storage client library (Microsoft.WindowsAzure.StorageClient.dll) is automatically referenced in any new web or worker role projects that you create, if you need to add the reference manually, it can be found at the following location:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
C:\Program Files\Windows Azure SDK\v1.0\ref\
```

We won't need to add this assembly to our project, as it's already referenced. However when you are building your own code, you will probably want to split your code into proper layers. When doing that you will need to add the assembly to your own custom assembly.

Now that we know how to reference our assembly, let's look at how we configure the Storage Client Library to access our development storage account.

8.4.2 Accessing development storage

To use the Development Storage for storing blobs, you need to configure your application to use the development blob service in the same way as you would if it was the live system. As far as your application is concerned, the only things that are different between Development Storage and the live system are:

- Account Name
- Account Shared Key
- Endpoint

In Development Storage, there is no ability to create or host multiple storage accounts and there is no Azure Portal so you cannot easily generate a new key. The endpoints and URI structure also differs from the live system.

BLOB STORAGE ACCOUNT NAME

The only supported account name for Development Storage is by default:

```
devstoreaccount1
```

Although you can change the default account name in the DSService.exe.config file, no useful purpose is served by doing this however.

BLOB STORAGE ACCOUNT SHARED KEY

Just as the account name is effectively hardcoded, the account shared key is also hardcoded to:

```
Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPTOtr/KBHBeksoGM
Gw==
```

If required you can change this value by modifying the setting in DSService.exe.config, again this is not recommended.

BLOB STORAGE END POINT

In section 8.2.1 we showed you how to access a blob that was held on the live storage account in a public container using an http GET request. The URI used was:

http://silverlightukstorage.blob.core.windows.net/ChrisConverted/Podcast01.wma

If we wish to access the same file from the Development Storage Blob Service we could use the following URI:

http://127.0.0.1:10000/devstoreaccount1/ChrisConverted/Podcast01.wma

This could be formalized as:

http://127.0.0.1:10000/<StorageAccountName>/<Container>/<BlobName>

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The Blob Storage Endpoint for development storage blob services is by default 127.0.0.1:10000, like the account name this value is modifiable in DSService.exe.config but you should leave it to its default setting.

With that knowledge in-hand, let's look at how we can use this to access our account in code.

STORING ACCOUNT DETAILS IN THE SERVICE CONFIGURATION

To make things a little easier, the Windows Azure SDK provides a property that will spin up a CloudStorageAccount object with the default development storage settings extracted from your DSService.exe.config file via the following property:

```
CloudStorageAccount = CloudStorageAccount.DevelopmentStorageAccount;
```

Although this is probably the quickest method of getting started, it really isn't best practice. In this instance you are effectively hardcoding your application to your development account meaning you would have to modify and recompile your code before you could deploy your application to the live system. This can complicate your build process and introduce bugs.

The best practice for storing this information is to store the data in the Service Configuration giving you the option to change this information without redeploying the whole application. For example if your shared key is compromised then you would be able to generate a new shared key and modify your application to use the new key by simply changing the service configuration via the Azure portal.

To help you easily use the ServiceConfiguration file to store your account details, the storage client library provides a method that can extract the account name, shared key and endpoint from a configuration setting. The following call is used by the library to extract these values:

```
CloudStorageAccount =
   CloudStorageAccount.FromConfigurationSetting("DataConnectionString");
```

In the above example your account details will extracted from a configuration setting named DataConnectionString.

Although this code is specific to the storage client library you should still store the account details in the service configuration even if you are using the REST API directly. This will simplify and standardize your code and allow you to easily use both the storage client and the REST API directly within your application (you really don't want to modify 2 different settings to access your account).

DEFINING CONFIGURATION SETTINGS

Now that we have seen how easy it is to extract an account from your configuration, let's look at how we would define that configuration setting.

As explained in Chapter 4 you will first need to define your configuration settings in the Service Definition file before you can configure them. The following setting is the standard method of defining your storage account in your Service Definition file.

```
<ConfigurationSettings>
   <Setting name="DataConnectionString"/>
</ConfigurationSettings>
```

Notice in the code above that DataConnectionString is the same name we passed to the FromConfigurationSetting method, earlier.

COMMUNICATING WITH DEVELOPMENT STORAGE

Once we have defined the configuration settings, we can set the runtime values in our service configuration file. The following configuration settings are the defaults used to talk to Development Storage.

```
<ConfigurationSettings>
  <Setting name="DataConnectionString"
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
                value="UseDevelopmentStorage=true" />
      </ConfigurationSettings>
```

By setting the value of the DataConnectionString to **UseDevelopmentStorage=true**, we are effectively telling the storage client to extract our settings from the DSService.exe.config, which gives us the same result as using the DevelopmentStorageAccount property.

The advantage of using the FromConfigurationSetting method over the DevelopmentStorageAccount property, is that we can modify the service configuration file to use the live account details (shown later in this chapter) without having to recompile or redeploy our application.

Now that our application is configured to use Development Storage via the storage client library we can continue on and create our web page.

8.4.3 Creating a Container

We will now create the web page shown in figure 8.11. Within our podcast sample web role project, create a new ASPX page called containers.aspx.

At this stage we only want to write the code that will create our container, we don't need to see the list of containers. At present our UI only needs to display the following elements:

- The new container name textbox
- the create container button

Listing 8.1 shows the ASPX required for the create container section of the page.

Listing 8.1 ASPX for creating a container

```
<div>
   <div>
      Name: <asp:TextBox  ID="txtContainerName" runat="server" />         #A
   </div>
   <div>
      <asp:Button ID="btnCreate" runat="server"
                  Text="Create" OnClick="btnCreate_Click" />              #B
   </div>
</div>
```

#A - Name of the container to create
#B - Create Button

Now that we have defined our UI we need to create the code that handles the button click. Listing 8.2 contains the code-behind for the create button click event.

Listing 8.2 Creating a new container

```
   protected void btnCreate_Click(object sender, EventArgs e)
   {
      CloudStorageAccount account =
         CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

      CloudBlobClient blobClient =
         account.CreateCloudBlobClient();                                  #A

      CloudBlobContainer container =
         blobClient.GetContainerReference(txtContainerName.Text);          #B

      container.Create();                                                  #C
   }
```

#A – Create the Blob Client

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

#B – Get a reference to the container (even though it doesn't exist yet)

#C – Create the container

So before we explain the code, we need to remind ourselves of its purpose. The user will type in the new container name and then click on the button to create the new container. Let's look at the code.

STORAGE ACCOUNT

The first thing we need to do is retrieve an object that allows us to work with the blobs storage account. Using the CloudStorageAccount that we used earlier to extract our credentials, we can now instantiate a CloudBlobClient object that will allow us to mess with things at an account level. This is achieved with the following call:

```
CloudBlobClient blobClient =
      account.CreateCloudBlobClient();
```

Once we have retrieved the CloudBlobClient object, we can perform the following operations at an account level on blob storage:

- Return a list of all containers in the account (ListContainers)
- Get a specific container (GetContainerReference)
- List Blobs (ListBlobsWithPrefix)
- Get a specific Blob (GetBlobReference)

As well as performing the above operations, you can also set some general policies such as:

- Block Sizes (see later)
- Retry Policy
- Timeout
- Number of Parallel Threads

In our example, since we are creating a new container, we need to grab a reference to the container that we wish to create therefore we will use the GetContainerReference method passing in our new containers name:

```
CloudBlobContainer container =
      blobClient.GetContainerReference(txtContainerName.Text.ToLower());
```

As you can see from the code above, in this example we are setting the container name to whatever the user types in the text box.

> **LOWER CASE CONTAINER NAMES**
>
> The name of the container is converted to lowercase as the blob storage service does not allow uppercase characters in the container name.

At this stage we have not made any communication with the storage service. We have just setup the container we wish to create. The CloudBlobContainer object that has been returned by GetContainerReference can perform the following operations:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

- Create a Container (Create)
- Delete a Container (Delete)
- Get & Set any custom metadata you may wish to associate with the container
- Get & properties associated with the container i.e. ETag and Last Modified Time
- Get & Set Container permissions
- List Blobs (ListBlobs)
- Get a specific Blob

Finally, we will now make a call to create the container.

```
container.Create();
```

As soon as we call the create method, the storage client will generate an http request to the BlobStorage service requesting that the container is created.

Default Permissions

In our Create container call, we did not specify any permissions on the container to be created. By default a container will be created as private access only, meaning that only the account owner can access the container or any of the blobs contained within it. In the next chapter, we will look at how we can set permissions on containers and blobs.

You should now be able to run your web role and create some containers in your development storage account. At this stage you will not be able to see the containers that you have created within your web page but you could check they are there by running a SQL query against the Development Storage database.

8.4.4 Listing Containers

Now that we can create a container from our web page, we need to modify our page so we can display all the containers in our storage account. This will now complete our containers.aspx page so it looks like the web page in figure 8.11.

In the figure 8.11, there was a grid that contained a list of all the containers in the account. To create that grid, we need to is update our asp.net page to include an ASP.NET GridView component (we will then bind this grid to a list of containers later on). You should place the code in Listing 8.3 before the code in Listing 8.1 in the containers.aspx page.

Listing 8.3 Listing Blobs with a grid view

```
<asp:GridView ID="gvContainers" runat="server"
              AutoGenerateColumns="true"                          #1
              OnRowDataBound="gvContainers_RowDataBound"
              onrowcommand="gvContainers_RowCommand"
              onrowdeleting="gvContainers_RowDeleting">
   <Columns>
       <asp:TemplateField>                                        #2
          <ItemTemplate>
          <asp:LinkButton ID="btnDelete" runat="server"
                          Text="Delete"
                          CommandName="Delete"
                          CommandArgument='<%#Eval("ContainerName")%>'/>
          </ItemTemplate>
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
            </asp:TemplateField>
        <asp:HyperLinkField Text="View"
                DataNavigateUrlFields="ContainerName"
                DataNavigateUrlFormatString="Blobs.aspx?Container={0}" />#3
      </Columns>
   </asp:GridView>
```

#1 - Autogenerate the columns in the grid from the bound object
#2 - Delete the container
#3 - We'll use this hyperlink later to navigate to a page that lists the blobs

The code in Listing 8.3 is the ASP.Net markup for Figure 8.11. Notice that we are allowing the grid to autogenerate all the columns (except the delete button and the view hyperlink) based on the properties of the object bound to the grid. Listing 8.4 shows the code-behind for our web page that gets a list of containers from the account and binds it to the grid.

Listing 8.4 Binding the Grid

```
protected void Page_Load(object sender, EventArgs e)
{
   if (!IsPostBack)
      BindGrid();
}

private void BindGrid()
{
   CloudStorageAccount account =
      CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

   CloudBlobClient blobClient =
      account.CreateCloudBlobClient();

   gvContainers.DataSource =
      blobClient.ListContainers();                                        #A

   gvContainers.DataBind();                                               #B

}
```

#A – Set the grids data source as the list of containers in the account
#B – Bind the data grid to its data source

We should now have a web page that will display all the containers in our storage account and allow you to create new private containers. To complete our sample we just need to implement the delete functionality.

8.4.5 Deleting a Container

From our web page if we click on the delete button on a particular row then it should delete the underlying container. To achieve this, we need to hook in our Delete button. Listing 8.5 shows the code-behind for our delete functionality.

Listing 8.5 Deleting the container

```
   protected void gvContainers_RowCommand
                        (object sender, GridViewCommandEventArgs e)
   {
      if (e.CommandName == "Delete")
      {
          DeleteContainer(e.CommandArgument.ToString());                  #A
      }

      BindGrid();
   }

   private void DeleteContainer(string containerName)
   {
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
      CloudStorageAccount account =
         CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

      CloudBlobClient blobClient =
         account.CreateCloudBlobClient();

      CloudBlobContainer container =
         blobClient.GetContainerReference(containerName);                  #B

      container.Delete();                                                  #C
   }

   protected void gvContainers_RowDeleting(object sender, GridViewDeleteEventArgs e)
   #D
   {

   }
```

#A – Call the DeleteContainer method on receiving the Delete Command
#B – Get the container from the blob account
#C – Delete the container
#D – We need this otherwise the grid will throw a wobbly

With the Delete button code hooked in, if we now run our application we should be able to view all blob containers in Development Storage, add a container and then delete it from our web page.

Wow, we have done pretty great. We've just completed our first Windows Azure Blob Storage application. I guess all that's left is to make this baby work against the live blob storage service.

8.5 Configuring your application to work against the live service

To switch our application from the development storage to the live storage account, we need to perform 3 items:

- Create a Live Storage
- Switch our configuration the live storage account

In this section, we won't go through the process of creating a storage account, as it's pretty simple and the information you require is pretty much available in this chapter. So let's get on and configure our application to work against our live storage account.

Always set affinity for your storage account

During the process of creating your storage account, always set affinity. If you did not set affinity you could have a situation where your web role was hosted in Washington State but your storage account was hosted in Chicago. The latency caused by cross data center communication would harm the performance of your application. To gain maximum performance you should always set affinity on your web roles, worker roles and storage accounts to the same data center ensuring you achieve the best possible network latency.

8.5.1 Switching to the live storage account

To make our application work against the live system, all we need to do is modify the value of our configuration setting in the Service Configuration file. That's it, the end, nothing else to do ☺. If you remember earlier we set our storage account configuration setting to the following:

```
<Setting name="DataConnectionString"
               value="UseDevelopmentStorage=true" />
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Although this is great for the development storage, it doesn't really give us a clue to the structure of the setting for when we want to use the live system. The following setting shows how the string should be structured.

```
<Setting name="DataConnectionString"
              value="DefaultEndpointsProtocol=protocol;
              AccountName=storageaccountname;
              AccountKey=storageaccountkey" />
```

To make this run against the live system, we just need to plug in the appropriate values e.g.

```
<Setting name="DataConnectionString"
         value="DefaultEndpointsProtocol=http;
         AccountName=silverlightukstorage;

AccountKey=Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPTOtr/KBHBeks
oGMGw==
" />
```

With the live settings configured, we can now use the live blob storage system either from the development fabric or the live production fabric.

8.5.2 Configuring the Access Key

If you are unsure where you get the account key for the storage account, you can always refer back to the Azure portal. On creation of a new storage account in Windows Azure a primary and secondary access key is generated for you to secure your API requests. The access key for our live account is used by all storage services and is available from the storage account section of the portal. In figure 8.12 we see where we can retrieve our access key. This key would replace the development key held in the service configuration file.

Figure 8.12 Access Key in the Azure Portal

> **Regenerating the Key**
>
> If your access key is compromised at any point then you can regenerate a new key by clicking the regenerate button as show in figure 8.12. Once a new key has been generated you will need to update the key in the service configuration for your application.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

8.6 Summary

In this chapter we took a quick overview of the sort of problems that you would normally face when trying to provide a shared storage solution in a traditional web farm. We have seen that sharing files between multiple servers is not easy and that Windows Azure provides a pretty neat mechanism that allows us to forget about those worries. After a brief introduction to storage services we showed you how blob storage fits into the overall architecture.

Then we got straight into developing our first application within the development environment, first using the storage client library supplied in the SDK, and then changing the configuration so it could work against the live production system. Hopefully you now have an appreciation of how easy it is to get started with blob storage.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

9

Uploading and Downloading BLOBs

This chapter covers:

- Uploading Files
- Downloading Files
- Serving files from Blob Storage using an http handler
- Improving performance using local storage
- Using custom metadata
- Splitting large files into smaller blocks

In the previous chapter we showed you how to get started with Blob Storage development using the storage client library focusing on the managing accounts and containers. In this chapter we will:

- Look at the underlying REST API for the Blob Service
- Managing Blobs using the Storage Client

9.1 Using the REST API

So far we have only used the Storage Client sample library in the SDK and have ignored the REST API. The reason for this is that as a developer you are unlikely to be writing code directly against the REST API. In general you will prefer to use a more object oriented structure that uses familiar looking .NET classes.

The storage client library is very useful but it is only a wrapper implementation of the REST API (which is the only official API). So although you will mainly be working against the storage client library there are some instances where you may need to use the REST API directly.

Windows Azure is an evolving platform and the Windows Azure team will typically release new features exposed via the REST API first. At a later date they may then provide an update to the SDK. If there is a new feature that you badly need to use then you may not have the luxury of waiting for the SDK update.

Another reason that you may need to the REST API directly is that not all features are implemented (or implemented in the way you may expect) in the SDK, so you may need to drop down to the REST API to use that feature. Rather than showing you showing you every last feature with the REST API, we will try and show you the important parts:

- How to List Blobs in a Public Container using the REST API
- Authenticating Private Requests using REST

9.1.1 Listing Blobs in a public container using REST

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In this example we are going create a small console application that will return a list of all the blobs in a public container using the REST API. To do that we will return to the funky little podcasting conversion sample that we were developing in the previous chapter. In that application, we can assume that we have converted a bunch of MP3s to WMA, and we now wish to list all the converted podcasts. In the console application that we will develop we will return all the blobs stored in the ChrisConverted public container (which holds the WMA files) from the silverlightsukstorage blob service account in the live production system. Figure 9.1 shows the result returned from the request within our console application.

Figure 9.1 - Console Application that returns a list of blobs from a public container using REST

If we look at the xml output returned above that we can see that this container contains a single wma file called mi2limpbiskit.wma.

To create the code that returns this output, create a new Console Application in Visual Studio and replace the existing static main method with the code in Listing 9.1

Listing 9.1 Listing the blobs in a container via REST

```
static void Main(string[] args)
{
   HttpWebRequest hwr =
      CreateHttpRequest(new
Uri(@"http://silverlightukstorage.blob.core.windows.net/chrisconverted?comp=list"), "GET", new
TimeSpan(0, 0, 30));                                          #1

   using (StreamReader sr =
      new  StreamReader(hwr.GetResponse().GetResponseStream()))              #2
   {
      XDocument myDocument = XDocument.Parse(sr.ReadToEnd());                #3

      Console.Write(myDocument.ToString());
   }

   Console.ReadKey();
}


private static HttpWebRequest CreateHttpRequest(Uri uri, string httpMethod, TimeSpan timeout)
{
   HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(uri);     #4
   request.Timeout = (int)timeout.TotalMilliseconds;
   request.ReadWriteTimeout = (int)timeout.TotalMilliseconds;
   request.Method = httpMethod;                                            #5
   request.ContentLength = 0;
   request.Headers.Add("x-ms-date",
      DateTime.UtcNow.ToString("R", CultureInfo.InvariantCulture));        #6
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
   request.Headers.Add("x-ms-version", "2009-09-19");                    #7
              return request;
}
```

Wow, so that's quite a bit of code. All it really does is list the blobs in a public container and output to it to the console (as shown in Figure 9.1). Unfortunately whenever use the REST API directly your code will get more complex, I guess you can see why we prefer to use the storage client library.

It is worth remembering that the very same http requests that we generated above, are the very same requests that the storage client library will generate on our behalf.

Let's now take a greater look at the code in Listing 9.1, as it will give us a better understanding of the communication between our clients and the storage accounts.

THE REQUEST

If we look back at listing 9.1, you should be able to see that we create our request at #1 and then execute the request at #2.

THE VERB

At #1 we can see that we are making a GET request, this verb tells us that we are wanting some data to be returned from the server rather than performing an action that will update the data such as a create, update or delete.

THE URI

If we look at the URI that we are calling at #1, we can extract some interesting information about the request being made. The URI is:

```
http://silverlightukstorage.blob.core.windows.net/chrisconverted?comp=list
```

From the domain we can see that we are using the live blob storage service (`blob.core.windows.net`) and the request is being made against our storage account "silverlightukstorage". Looking at the request we can also derive that we would like a list of whatever is in the container chrisconverted (`chrisconverted?comp=list`), which is we know are blobs (in-fact the WMA files that we converted from MP3).

Windows Azure follows a standard naming convention for performing requests so once you are familiar with some of the API calls it is easy infer what other calls may look like. For example if you required a list of whatever resides in a storage account (i.e. containers), you could use the following URI:

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

It is important to note that you would need sign the request with your access key as it is not a public operation. Listing blobs in a public container can be performed without an authorization key since an authorization key is only required for private containers.

THE REQUEST HEADERS

If we look at the code for our standard CreateHttpRequest in Listing 9.1, we can see that 2 headers in particular are set:

- x-ms-version
- x-ms-date

The x-ms-version header (at #7) is an optional header that should be treated as a required header. The storage service versioning policy is that when a new version of an API is released any existing APIs will continue to be supported. By providing the correct x-ms-version header then you are stating which API you want your request to work against. This policy allows Microsoft to release new functionality and change existing API's but allowing your existing services to continue to work against the previous API.

Storage Version

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

> You should always check the version of the REST API that you are using to support a particular feature. At #7 we are using the 19th of September 2009 version of the API. If a new feature is released and it isn't working there is a good chance that you forgot to update the version. The good news is that whenever you download the latest version of the storage client library, it will already be using the correct version.

The x-ms-date header is a required header that states the time of the client request. We set this value at #6 in Listing 9.1. The value set in the request header is a representation of the current time in UTC e.g. "Sat, 27 Jun 2009 23:37:31 GMT". This request header serves two purposes:

- It allows the server to generate the same authorization hash as the client (next section)
- It prevents replay attacks by denying old requests

> **Server Time Synchronization**
>
> If you suddenly start getting errors whenever you call the Storage Service it may be worth checking the time on your machine. If the time of the request is out of synchronization (i.e. older than 15 minutes) with the server time in the data centers then the request will be rejected with a 403 response code.

We have looked at how at how to make non-authenticated requests against a public container and we have looked at making requests to the storage accounts via the SDK. We will now look at how to make authenticated requests via the REST API and give you an understanding of how the REST API calls are authenticated.

9.1.2 Authenticating private requests

In the previous section we developed a console application to return a list of the files that reside in our public container (chrisconverted). We will now modify this code to return all the containers in our development storage account. This sample is the direct REST API equivalent of the storage client calls that we performed in the previous chapter.

Since we only have 2 containers in our development storage account (ChrisOriginals and ChrisConvered) we would expect the following xml output from our console application:

```
<EnumerationResults AccountName="http://128.0.0.1:10000/devstoreaccount1/">
  <Containers>
    <Container>
      <Name>chrisconverted</Name>
      <Url>http://128.0.0.1:10000/devstoreaccount1/chrisconverted</Url>
      <LastModified>Sat, 27 Jun 2009 23:37:31 GMT</LastModified>
      <Etag>0x8CBC5975FB7A0D0</Etag>
    </Container>
    <Container>
      <Name>chrisoriginals</Name>
      <Url>http://128.0.0.1:10000/devstoreaccount1/chrisoriginals</Url>
      <LastModified>Sat, 27 Jun 2009 23:14:23 GMT</LastModified>
      <Etag>0x8CBC594247EFB60</Etag>
    </Container>
  </Containers>
  <NextMarker />
</EnumerationResults>
```

Listing 9.2 contains the code that we used to make this request via the REST API. Before you can run this sample you will need to reference the storage client as we will use some of its library calls to sign the http request with the shared access key. You will also need the following using statement at the top of your class.

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

Listing 9.2 Listing the containers in the development storage account via REST

```
static void Main(string[] args)
{
   HttpWebRequest hwr = CreateHttpRequest(
      new Uri(@"http://127.0.0.1:10000/devstoreaccount1?comp=list"),
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
      "GET", new TimeSpan(0, 0, 30));                                    #A

   var account =
      CloudStorageAccount.FromConfigurationSetting("DataConnectionString");#B

   account.Credentials.SignRequest(hwr);                                 #C

   using (StreamReader sr =
             new StreamReader(hwr.GetResponse().GetResponseStream()))    #D
   {
      XDocument myDocument = XDocument.Parse(sr.ReadToEnd());
                  Console.Write(myDocument.ToString());
   }

      Console.ReadKey();
   }
```

#A – Create the Http Request as before
#B – Get our account credentials using our faithful storage client library
#C – Sign the request using our favorite little storage client library
#D – Execute the request and get the response

This code in Listing 9.2 is very similar to the code in Listing 9.1 except we are using a different URI (as we are listing containers rather than blobs) and signing the request. The URI for listing containers in the development storage account is:

http://127.0.0.1:10000/devstoreaccount1/?comp=list

As explained in section 9.1 this URI will return a list of the contents of the storage account (i.e. a list of containers).

Apart from the URI the only difference between the 2 listings is that we sign the http web request with our shared key credentials. The process of signing the request is quite complicated so it is best to use the storage client code as we have done in Listing 9.2.

In this case we simply use the same code that we wrote earlier to extract the configuration setting.

```
var account =
   CloudStorageAccount.FromConfigurationSetting("DataConnectionString");
```

APP SETTINGS

Please note that CloudStorageAccount.FromConfigurationSetting will work with both Windows Azure ConfigurationSettings and regular.net App Settings.

We can then sign the request using the request signing functionality built into the storage client library as shown below:

```
account.Credentials.SignRequest(hwr);
```

Even if you do plan to only the REST API, it is still worth referencing the storage client library just to use that bit of code.

SIGNING THE REQUEST

Although the request signing code is already implemented for us, it is worth looking at the overall process of signing a request.

If you were to look at the http request that we generated in Listing 9.2 (you could stick a breakpoint after the request has been signed) you would see that the request contains an Authorization header (generated by #2 and #3). The authorization header for the request in Listing 9.2 is:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

SharedKey devstoreaccount1:J5xkbSz7/7Xf8sCNY3RJIzyUEfnj1SJ3ccIBNpDzsq4=

The signature in the header (the long string after devstoreaccount1) is generated by canonicalizing the request, canonicalizing is a defined process that converts the request into a predictable format. The canonicalized request is hashed using a SHA256 algorithm and then encoded as the signature using Base64 encoding.

Further details of the canonicalization process can be found in the online documentation at:

http://msdn.microsoft.com/en-us/library/dd179428.aspx

PROCESSING THE REQUEST

Once the request is received by the server, the server can take the incoming request and perform the same canonicalization and hashing process with the shared key. If the signature generated matches the Authorization header it can perform the request. Figure 9.2 shows the validation process between the client and the blob storage service. Notice that the authorization key generated by the server matches the original client request.

Figure 9.2 - Validation of the authorization key

If the signature generated by the server is different to the authorization header then the server will not process the request and return a 400 response code (Bad Request). This method of being able to generate the same authorization key both client and server side means users who do not have the shared key will be prevented from performing unauthorized requests against the account. As the shared key is never sent between the client and the blob storage service, the risk of the key being compromised is substantially reduced.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The authorization key is generated from both the contents of the request and the shared key, generating the hash with both pieces of information means that the user cannot tamper with the request to perform another operation. If for example a hacker were to intercept our previous request then the generated authorization header server side would change and the request would be denied. Figure 9.3 shows the validation process between the client and the blob storage service when the original request from figure 9.2 has been tampered with.

Figure 9.3 Tampered Request

In figure 9.3 we have tampered with the original request for figure 8.2 to return any containers from devstoreaccount2. Notice that the server generated a different authorization key from the tampered request and therefore the server returns a 400 response code (Bad Request).

Now that we understand how the REST API authentication process works, it's probably worth stopping for a quick beer. Just think if you had to implement that code yourself rather than Microsoft providing it, you'd probably prefer to have a long beer and drown your sorrows away.

9.2 Managing Blobs using the Storage Client

Now that you've finished your beer and you're sober enough to read this, let's look at how we can use the storage client library to list blobs in a container rather than using the REST API directly.

In chapter 8 we developed a sample management application that would allow us to upload podcasts in an mp3 format that were ready to be converted to wma. We developed an asp.net page that displayed a grid of all the containers in our storage account. We included in the grid a hyperlink that would redirect you to another page called blobs.aspx passing in the name of the selected container in the querystring. We will now extend that example to develop the page blobs.aspx. This page will allow you to:

- List all blobs in the selected container
- Upload a new file
- Delete an existing blob
- Download an existing blob

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 9.4 shows how our final web page will look.

Figure 9.4 – An ASP.NET Web page that displays all the blobs within the selected container in a grid control

As we can see from the figure 9.4 that the web page we will be developing is very similar to the page that we developed for listing containers in chapter 8.

9.2.1 Listing Blobs using storage client

In this section we will look at how to generate the grid that displays the list of blobs. We will look at uploading, deleting and viewing blobs later in the chapter. Listing 9.3 contains the markup required for the blobs.aspx page shown in figure 9.3.

Listing 9.3 ASPX Page for listing the blobs in a container

```
<div>
   <asp:GridView ID="gvBlobs" runat="server"
                 AutoGenerateColumns="true"
                 onrowcommand="gvBlobs_RowCommand"
                 onrowdeleting="gvBlobs_RowDeleting">                    #1
      <Columns>
         <asp:TemplateField>
            <ItemTemplate>
               <asp:LinkButton ID="btnDelete" runat="server"
                               Text="Delete"
                               CommandName="Delete"
                               CommandArgument='<%#Eval("Name")%>'/>
            </ItemTemplate>
         </asp:TemplateField>
         <asp:TemplateField>
            <ItemTemplate>
               <asp:LinkButton ID="btnView" runat="server"
                              Text="View"
                              CommandName="View"
                              CommandArgument='<%#Eval("Name")%>'/>        #2
             </ItemTemplate>
          </asp:TemplateField>
       </Columns>
   </asp:GridView>
   <div>
      <div style="padding: 10px;">
         <div>
            Upload: <asp:FileUpload ID="fu" runat="server" />             #3
         </div>
         <div>
            <asp:Button ID="btnUpload" runat="server"
                        Text="Upload"
                       OnClick="btnUpload_Click" />                      #4
         </div>
      </div>
   </div>
</div>
```

#1 Grid of blobs
#2 View Blob Button
#3 File upload control

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

#4 Upload Button

Cueballs #1-3 in code and text

The ASPX code for displaying the blobs is similar to the ASPX we used in chapter 8 to list containers, but there are a few differences to note.

At #1 we display a GridView that displays all the blobs in its data source, again notice that we are choosing to auto generate the columns. At #2 we have defined a hyperlink button that we will use to download the blob, and at #3 we will use the standard ASP.NET FileUpload control to upload our file.

THE CODE BEHIND FOR THE WEB PAGE

Now that we have defined how our web page will look, we need to bind our grid to the data source on page load. Listing 9.4 shows the code behind required to display a list of blobs for the selected container in our grid.

Listing 9.4 Show the list of blobs in the grid

```
protected void Page_Load(object sender, EventArgs e)
{
   if (!IsPostBack)
      BindGrid();
}

private void BindGrid()
{
   CloudStorageAccount account =
      CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

   CloudBlobClient blobClient =
      account.CreateCloudBlobClient();

   CloudBlobContainer container =
      blobClient.GetContainerReference(Request["container"] as string);           #1

   gvBlobs.DataSource = container.ListBlobs();                          #2
   gvBlobs.DataBind();
```

#1 Get the container from the container passed in the request query string
#2 List the blobs in the container

Cueballs #1-2 in code and text

The code shown in listing 9.4 is again very similar to the list containers example in chapter 8. The code displayed in listing 9.4 is actually the storage client equivalent to the code we used to display the list of blobs with the REST API directly in listing 9.1

At #1 we retrieve an instance of our Blob Container by calling the GetContainerReference method off of our blob client, notice we are passing in the container name from the request querystring. Finally we call retrieve a list of the files held in the container by calling the ListBlobs method and binding the result to our GridView at #2.

Now that we can list and display all of the blobs in the selected container, we need to extend our web page so that we can upload new files into the container.

9.2.2 Uploading Blobs

To upload files from our web page we have chosen to use the built-in ASP.NET upload control at #3 in listing 9.4. On click of the upload button at #4 we wish to capture the uploaded file and then upload the captured file to blob storage. Listing 9.5 contains the code-behind for the upload button click event.

Listing 9.5 Posting the uploaded file to blob storage

```
protected void btnUpload_Click(object sender, EventArgs e)
{
   CloudStorageAccount account =
      CloudStorageAccount.FromConfigurationSetting("DataConnectionString");
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
    CloudBlobClient blobClient =
       account.CreateCloudBlobClient();

    CloudBlobContainer container =
       blobClient.GetContainerReference(Request["container"] as string);

    var blob =
       container.GetBlobReference(fu.PostedFile.FileName);                    #1

   blob.UploadByteArray(fu.FileContent);                                      #2


    BindGrid();
}
```

#1 – Get a reference to our new blob
#2 – Upload the file contants to the blob

Cueball #1, 2 in code and text

Since we have seen all the other code in previous examples, we only need to explain the code at #1 and #2.

At #1 we get a reference to the blob that we are about to create. In this case we will give the blob the original name of the uploaded file (retrieved from the UploadFile control).

FILE NAMES

You will notice that we are extracting the file name and the file contents directly from the asp.net file upload control. We can if wish give the file a different name from its original.

Setting the Max Request Length

By default asp.net is configured to allow a maximum upload of 4Mb if you provide a web role front-end to the blob storage as we have done in this sample then you may need to increase maximum request length.

To increase the default value to a larger value, you need to add the following line under the system.web element in the web.config:

```
<httpRuntime executionTimeout="300" maxRequestLength="51200"/>
```

The maximum upload size in the above example is 50Mb.

In the previous example we used the UploadByteArray method to upload our blob. There are 3 other methods provided in the storage client library that we can use:

- UploadFile
- UploadText
- UploadFromStream

Depending on your situation, one of these methods may be easier to use than UploadByteArray (e.g. UploadFile if you have a local file on disk that you wish to store in Blob Storage).

Splitting Blobs into Blocks

The Maximum size of a blob is 1TB, however if a file is larger than 64MB then under the covers the storage client library will split the file into smaller blocks of 4MB. One of the advantages of the storage client library is that you don't need to worry about this. If you are using the REST API then you will need to implement the splitting of blobs into blocks, the committal of blocks and retry logic associated with re-uploading blocks on failure (yet another good reason to use the storage client library.

If you are a sick and twisted individual who wants to mess around with blocks then feel free to look in more detail at the online documentation:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

http://msdn.microsoft.com/en-us/library/ee691964.aspx

Now that we have spent all that time and effort adding our file to blob storage, let's delete it, yay.

9.2.3 Deleting Blobs

Deleting a blob is very similar to uploading a file except we delete the blob instead of uploading it ;). Just like the upload file example in Listing 9.5, we would get the reference to the blob, and the delete the blob by calling

```
blob.Delete();
```

Listing 9.6 shows the code to delete the Blob in our web page.

Listing 9.6 Deleting the Blob

```
protected void gvBlobs_RowCommand
                      (object sender, GridViewCommandEventArgs e)
{
   if (e.CommandName == "Delete")
   {
       DeleteBlob(e.CommandArgument.ToString());
   }

   BindGrid();
}

private void DeleteBlob(string blobName)
{

   CloudStorageAccount account =
      CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

   CloudBlobClient blobClient =
      account.CreateCloudBlobClient();

   CloudBlobContainer container =
      blobClient.GetContainerReference(Request["container"] as string);

   var blob =
     container.GetBlobReference(blobName);

   blob.Delete();                                                    #A
}

protected void gvBlobs_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
}
```

#A – Delete the Blob

As you can see deleting a blob is pretty simple. Now that we can list, upload and delete blobs from our storage account using our asp.net management website hosted in our Windows Azure web role, we will now complete the management page example by looking at how we can download blobs.

9.3 Downloading Blobs

In this section we will look at how we can at how we can download blobs in the following situations:

- Downloading blobs from a public container
- Downloading blobs from a private container

To take things nice and gently we will look at how we can download blobs stored in a public container first.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

9.3.1 Downloading Blobs from a Public Container

If our blob is hosted in a public container then we can present the URI of our blob directly to the user and they will be able to download the file in their browser directly. In our podcasting sample, the following URI will download podcast01.wma from the chrisconverted public container in our development storage account:

http://127.0.0.1:10000/devstoreaccount1/ChrisConverted/Podcast01.wma

Since the blob is held in a public container, the user will not need to provide any credentials to access the blob.

> **Mime Types**
>
> If we have correctly set the mime type of our podcast then when the URI is pasted into our browser then the podcast would automatically start playing in Windows Media Player.

In our management web page example however, we do not wish to expose our podcasts to the world and wish to restrict access to our own credentials. Let's look at how we can do that using our new best friend, the storage client.

9.3.2 Downloading Blobs from a private container using storage client

We would now like to modify our management web page so that if we clicked on the view button for the selected podcast, as show in figure 9.5, then we would like to be prompted to download the file as also shown in figure 9.5.

Although we could have just stored the blob in a public container to achieve the same result, in this example we will first download the blob to our web role and then serve the blob from our web role rather than from the Blob storage directly.

In Listing 9.3, we defined the ASPX for our view button; listing 9.7 is where we hook up the code that will download the file when this button is clicked.

Figure 9.5 - Clicking the view button on our blobs page

In Listing 9.3, we defined the ASPX for our view button; listing 9.7 is where we hook up the code that will download the file when this button is clicked.

Listing 9.7 Downloading Blobs from the grid using storage client

```
protected void gvBlobs_RowCommand(object sender, GridViewCommandEventArgs e)
{
   if (e.CommandName == "Delete")
   {
      DeleteBlob(e.CommandArgument.ToString());
   }

   if (e.CommandName == "View")
   {
       DownloadBlob(e.CommandArgument.ToString());
   }
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
      BindGrid();
   }

   private void DownloadBlob(string blobName)
   {
      CloudStorageAccount account =
         CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

      CloudBlobClient blobClient =
         account.CreateCloudBlobClient();

      CloudBlobContainer container =
         blobClient.GetContainerReference(Request["container"] as string);

      var blob =
        container.GetBlobReference(blobName);


      using (var ms = new MemoryStream());                                #1
      {
         blob.DownloadToStream(ms);                                       #2
         Response.ContentType = blob.Properties.ContentType;              #3
         Response.AddHeader("Content-Disposition",
                              "attachment; filename=" blobName);      #4
         Response.BinaryWrite(ms.ToArray());                                  #5
      }
    }
```

#1 Create Memory Stream
#2 Download blob to Memory Stream
#3 Set the Mime Type
#4 Make it flash up as an attachment
#5 Write the file to the response stream

Cueballs #1-5 in code and text

The code in Listing 9.7 will download the selected blob and prompt the user with a save dialog box as shown in figure 9.5.

At #1 we create a new memory stream where will write the contents of the blob downloaded from blob storage at #2. At #3, we set the mime type of the file so that we can allow the browser to perform the correct action based on that type (e.g. launch Windows Media Player, Word etc). At #4 we add the content-disposition header so the browser knows to offer a save file dialog box to the user and finally we make the downloaded blob available to the client by writing the file to response stream at #5.

Other Download Options

In the above example we used the DownloadToStream option, however the Storage Client does offer other methods of downloading files from blob storage including:

DownloadText

DownloadByteArray

DownloadToFile

So far we have seen how to download files that are in public or private containers. What if we something a little more granular, i.e. we wish to control access to our blobs or containers, well in those case we can use a shared access signature. Later on in, we will look at this feature in a little more detail.

Now that we know how to upload and download blobs, let's look at how we can integrate blob storage with our existing ASP.NET Websites.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

9.4 Integrating Blobs with your ASP.NET Websites

In typical ASP.NET Websites, we will typically distribute our assets with our website. Although this strategy works great for small websites, this is pretty much unmanageable when dealing with larger websites. Do we really want to redeploy our entire website just because we have new Hawaiian shirt in our product range and we need to add an image of it?

In this section we will look at how you can:

- Integrate public assets with your ASP.NET Website
- Integrate private assets (e.g. purchased MP3 files) with your ASP.NET Website

Let's return to the Hawaiian shirt shop sample from chapter 2, to show how we can integrate our public assets.

9.4.1 Integrating ASP.NET Websites with table driven blob content

Currently the shirt shop website displays a list of all the products that we have for sale but does not display a picture of the shirt. As the data is currently hardcoded we could just store the image on the website directly as any changes to the product line would require a new version of the website to be deployed.

In future chapters we will drive the data from an external data source such as Table Storage or SQL Azure, so we need a strategy that allows us to update static content on the website when the product line is changed without redeploying the images to the website.

If we stored the picture of the shirt in blob storage then we could store the URI of the blob in our external data source. This means that as we add new items to our data source, we can add the associated blob at the same time. So, in the Hawaiian shirt shop example, we could store an image of the shirt in blob storage and as well as adding a new shirt to the table. In the table, we would store all the details of the shirt (name of the shirt, price, description) and its associated URI. Figure 9.6 shows how this might be represented.

Figure 9.6 Storing URI's in an external data source

As we are happy for the images of our Hawaiian shirts to be in the public domain we could store the shirt images in a public container and set the URI of our html image tag directly to the public URI. The code below shows how this image tag might look

```
<img src="http://silverlightukstorage.blob.core.windows.net/shirtimages/greenshirt.jpg" />
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Using a public container is fine for storing content that we do not mind making available to the world, but it is not a suitable solution if we wish to serve content that we wish to keep protected.

Content Delivery Networks

Storing your website assets in Blob Storage is a great performance optimization to make as it really will take load away from your web server. In the next chapter we will look at how you can take your assets stored in Blob Storage and expose them via the Windows Azure Content Delivery Network to make even more performance improvements.

Returning to our podcast sample, suppose we decided that we only wanted paid members of our website to be able to download our mp3. In that case, we would not store the file in a public container; we would store it in a private container instead.

9.4.2 Integrating protected, private content

Earlier in this chapter in our blob management web page we had a download link that allowed us to serve files stored in a private container to our users on a public website. We will now expand that technique to a more integrated, reusable solution by:

- Creating an HTTP Handler that serves MP3 files from Blob Storage
- Registering the HTTP Handler
- Protecting our handler so only authorized users can use it

HTTP Handlers

Http Handlers allow us to handle configured types of http requests to be handles by some custom code. In this case we will build a handler that will intercept requests for MP3 files and return the MP3 file from Blob Storage instead of the local file system.

CREATING THE HTTP HANDLER

We will now build a small sample that will intercept any incoming requests for mp3 files using an http handler. Rather than attempting to serve the mp3 file from our websites file system, the handler will retrieve the file from our private blob storage container and serve it to the user that requested it. We will build on the techniques that we used in Listing 9.7. We can either serve the file directly (useful for images) or initiate a save dialog (as we did in figure 9.5). In this example, we will download the file directly. So for example when the user requests the following URI, they will be served the file from blob storage:

```
www.mypodcastwebsite.com/podcast01.mp3
```

To implement an http handler, we need to add a new httphandler file to our ASP.NET Web Project. Our handler is called MP3Handler.ashx. Listing 9.8 contains the implementation for MP3Handler.ashx.

Listing 9.8 An MP3 Http Handler that serves mp3s from blob storage

```
public class MP3Handler : IHttpHandler
{
   public void ProcessRequest(HttpContext context)              #1
   {
      string blobName = context.Request.Path.Trim('/');         #2
      var blobData = GetBlob("chrisoriginals", blobName);#3

      context.Response.ContentType = "audio/mpeg";               #4
      context.Response.BinaryWrite(blobData);                    #5
   }

   public bool IsReusable
   {
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
      get {return false;}
   }

   private byte[] GetBlob(string containerName, string blobName) #6
   {
      CloudStorageAccount account =
         CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

      CloudBlobClient blobClient =
         account.CreateCloudBlobClient();

      CloudBlobContainer container =
         blobClient.GetContainerReference(Request["container"] as string);

      var blob =
        container.GetBlobReference(blobName);


      using (var ms = new MemoryStream());
      {
         return ms.ToArray());
      }
   }
```

#1 Incoming request
#2 Get the blob name
#3 Get the blob
#4 Set the mime type as MP3
#5 Write the file to the response
#6 Get the blob from blob storage

Cueballs #1-7 in code and text

The http handler in Listing 9.8 will be called whenever MP3 file requests are intercepted at #1. Once the request has been intercepted we will extract the name of the mp3 file that should be downloaded from blob storage from the requested path at #2. Our handler will then download the requested blob from the chrisoriginals private container at #3 by calling the GetBlob method at #6. Once we have retrieved the file from blob storage we will set the mime type as MP3 at #4 (we have hardcoded the mime type but in a more generic sample you could retrieve it from the blob properties) and then write the file back to the client at #5.

REGISTERING THE HTTP HANDLER

To register the http handler with our website we need to add the following line to our web.config.

```
      <add name="MP3Handler" path="*.mp3" verb="GET"
type="AzurePlayAreaWeb_WebRole.MP3Handler, AzurePlayAreaWeb_WebRole"
resourceType="Unspecified"/>
```

AUTHORIZATION

If we wish to restrict our MP3 files to logged in users, we could use the built-in ASP.NET Authorization and Authentication functionality:

```
<authorization>
  <deny users="?"/>
</authorization>
```

We now have an integrated http handler that can serve protected blob files from blob storage to authorized users as if it was part of our website. If our website was serving the same file to hundreds of users every day then continually retrieving the same file from blob storage would be inefficient. Let's now look at how we can use blob storage and local storage together, to cache requests, so we don't need to continually request the same blob from blob storage.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Shared Access Signatures

This technique is really useful for showing you how to integrate local storage and blob storage together. It is also useful as an option for integrating content. You can however use shared access signatures to provide the same result. Shared Access Signatures is the preferred approach for such a scenario as it's cheaper and takes load off your servers.

9.5 Using Local Storage with Blob Storage

In the previous section we used a file handler to intercept mp3 requests from our website and serve them from blob storage rather than our website. A more efficient approach would be to check if we have the file already on our local file system. If the file exists then we can just serve the file straight up. If the file doesn't exist we can retrieve it from blob storage, store it on the local file system and then serve the file. This means that all future requests for the file will not need to continually retrieve the file from blob storage.

Local Cache

When we define our Windows Azure web roles, we can allocate a portion of the local file system for use as a temporary cache. This local storage area allows us to store semi-persistent data that we may frequently use without having to continually re-request or re-calculate the data for every call.

You must be aware that this local storage area is not shared across multiple instances and the current instance of your web role is the only one that can access that data, Figure 9.7 shows the distribution of blobs in Local storage across multiple instances. Since the load balancer evenly distributes requests across instances a user will not always be served by the same web role instance. Any data that you need persisted across multiple requests (such as shopping cart or session data) should not be stored in local storage and must be stored in a data store that all instances can access.

Figure 9.7 – 3 Web Roles Instances all with different local copies of blobs in their local storage

9.5.1 Defining and accessing local storage

In our service definition file we can define how much space we require on our local file system. The fabric controller will use this information to assign our web role to a host with enough disk space. The following code is how we would define that we need 100Mb of space to cache our MP3 files:

```
<WebRole name="ServiceRuntimeWebsite">
   <LocalResources>
      <LocalStorage name="mp3Cache"
                    cleanOnRoleRecycle="true"
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
                    sizeInMB="100" />
   </LocalResources>
</WebRole>
```

To access any of the files held in local storage, we can use the following code to retrieve the location of the file:

```
     LocalResource localCache =
       RoleEnvironment.GetLocalResource("mp3Cache");

    string localCacheRootDirectory = localCache.RootPath;
```

Once we have retrieved the root directory of local storage (RootPath), we can then use standard .NET file system calls to modify, create, read or delete files held in local storage.

MORE ON LOCAL STORAGE

For more details on local storage, please see chapter 4

Now that we have configured our local storage area, we now need to modify our http handler to use our local storage area where possible.

9.5.2 Updating our Http Handler to use local storage

We will now modify our http handler to check in local storage if the requested file exists already. If the file doesn't exist, then we will retrieve the MP3 file from blob storage and store it in local storage. Finally we will write the file back to the client from our local file system. The following code shows how we would modify the ProcessRequest method in Listing 9.8 to use Local Storage.

```
   public void ProcessRequest(HttpContext context)
   {
      string blobName = context.Request.Path.Trim('/');

      var mp3Cache = RoleEnvironment.GetLocalResource("mp3Cache");
      string localFilePath = mp3Cache.RootPath + blobName;

     if (!File.Exists(localFilePath))
     {
        var blobData = GetBlob("chrisoriginals", blobName);
        File.WriteAllBytes(localFilePath, blobData);
     }

     context.Response.ContentType = "audio/mpeg";
     context.Response.WriteFile(localFilePath);
   }
```

Using the local storage mechanism in our http handler really improves our performance by serving the requested file from local storage rather than always have to retrieve the file from blob storage first. Although this improves performance it does mean that if the file is changed in blob storage then the file that we will serve will be out of date. Let's look at how we can keep our performance improvement but serve the latest content if the file has changed in blob storage.

Applicable whenever synchronizing local storage and blob storage

Although this sample is focused on web http handlers, this can easily be used in worker roles also. I currently use this technique with a Windows Azure Map Reduce solution that I've built.

9.5.3 Checking properties of a blob without downloading it (checking the HEAD)

If we were to use the http HEAD verb instead of the GET verb, then we could check the properties of the file

without downloading the file. Figure 9.8 shows the output of the HEAD request.

Figure 9.8 Showing the output of a HEAD Request

In figure 9.8, the last-modified tag shows us the last time the file was update in blob storage. By comparing the header value to the local value in our file properties we know if the file has changed.

x-ms-request-id

In figure 9.8, you will notice there is a tag called x-ms-request-id. Every request made is assigned an unique identifier (GUID) that is returned in the response. Every request and response is logged by Microsoft meaning that if you are experiencing issues then you can always pass this ID with a support request. This allows Microsoft to easily investigate any issues you may have.

Listing 9.9 shows the code for the console application shown in figure 9.8.

Listing 9.9 Showing the output of a HEAD request

```
static void Main(string[] args)
{
   HttpWebRequest hwr = CreateHttpRequest(
      new Uri(@"http://127.0.0.1:10000/devstoreaccount1/chrisoriginals/podcast01.mp3"),
      "HEAD", new TimeSpan(0, 0, 30));                                                  #1

   var account =
      CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

   account.Credentials.SignRequest(hwr);

   var response = hwr.GetResponse();

   foreach (string header in response.Headers)                                          #2
   {
      Console.WriteLine("{0} : {1}", header, response.Headers[header]);
   }
}
```

#1 HEAD request rather than GET
#2 Iterate through the returned headers

Cueballs #1-2 in code and text

In the code example shown in Listing 8.9 we make the same request as we have done before but instead of making a GET request we make a HEAD request for the file podcast01.mp3 in the chrisoriginals container in our development storage account at #1. At #2 we loop through all the returned headers outputting the header key and value to the console screen.

USING THE STORAGE CLIENT LIBRARY

In Listing 9.9 we retrieved the last modified time using the REST API directly, although we can of course use the storage client library. The following code performs a HEAD request returning the last modified time.

```
blob.FetchAttributes();
var lastModifiedTime = blob.Properties.LastModifiedUtc;
```

FetchAttributes is the storage client library equivalent of HEAD and will return all the properties and custom metadata (next section) of the blob (without downloading the actual file.

Blob Properties

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Property Name	Description
LastModifiedTimeUtc	Last time the blob was modified
Etag	Unique identifier of the request (changes every time a blob is modified)
Length	Size of the blob in bytes
ContentType	The mime type of the blob
BlobType	PageBlob, BlockBlob, Unspecified (will explain later)
ContentEncoding	Encoding of the header
ContentLanguage	Language Header of the Blob
ContentMD5	MD5 hash of the content header
LeaseStatus	Used with Page Blocks (explained later). Locked, Unlocked, Unspecified
CacheControl	Allows you to instruct the browser on how to cache the blob

Now that we have an overview of Blob Properties and how to retrieve them without downloading the file, let's look at how we can use this to improve the performance of our handler.

9.5.4 Improving our handler to check the last modified time

Now that we know how to retrieve the last-modified time, we can modify our handler to check if the file has changed since we last downloaded it. If it has changed, we can download another copy of the file. Listing 8.10 shows the modified code for our handler.

Listing 9.10 Improving our handler to check the last modified date

```
blob.FetchAttributes();
var lastModifiedTime = blob.Properties.LastModifiedUtc;                 #1

if (!File.Exists(localFilePath) ||
       File.GetLastAccessTimeUtc(localFilePath) < lastModified)         #2
{
    var blobData = GetBlob("chrisoriginals", blobName);
    File.WriteAllBytes(localFilePath, blobData);
    File.SetLastWriteTimeUtc(localFilePath, lastModified);              #3
}
```

#1 – Get the Blob Properties and Metadata
#2 – Check if blob has changed
#3 – Set Last Write time

Cueballs #1-3 in code and text

At #1 we get the Last Modified time of the blob. This method is the same code as we used in section 9.5.3, you can either use the storage client library or the REST API directly to retrieve this data.

Once we have the last modified time of the blob then we check at #2 if we already have the file locally and that the local file is not older than the server file. The final change that we need to make is that we set the "last write time" of the local file to the last-modified time of the blob at #3.

So far we have only looked at the properties of a Blob, let's now look at the other part of the blob data that is returned in a HEAD request, Custom Metadata.

9.5.5 Adding and returning custom metadata

In listing 9.9, we showed you how to view all the headers associated with a blob using a console application. In that example we returned all the standard headers associated with a blob (last-modified, x-ms-request-id, and

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

so on). If we want to associate some custom metadata with a file then this would be displayed in the response headers also.

DISPLAYING CUSTOM METADATA

We will now modify the blob podcast01.mp3 to include the author of the file (me) as custom metadata. Figure 9.9 shows the HEAD response in the same console application using the same code that we developed in listing 9.10.

Figure 9.9 - Head Response with custom metadata

Note that with the file podcast01.mp3 we have a new header returned in the response, called "x-ms-meta-author" with the value of "chris hay". In this example, we returned the metadata from the HEAD request, but it is also available in GET request.

SETTING CUSTOM METADATA

If we need to set some custom metadata, we can easily do this with the storage client library. The following code sets the metadata for the file podcast01.mp3:

```
blob.Metadata = new System.Collections.Specialized.NameValueCollection();
blob.Metadata.Add("Äuthor", "Chris Hay");
blob.SetMetadata();
```

This code calls the `UpdateBlobMetadata` method on the container, passing in a `NameValueCollection` of metadata that we wish to store against the blob.

x-ms-meta

The storage client library automatically prefixes all metadata with the tag `x-ms-meta-`. We actually set the key as `author` in the metadata collection, but the response header returned `x-ms-meta-author`. As a matter of fact, the blob storage service will only allow you to set metadata with the prefix x-ms-meta- and it ignores any attempt to modify any other header associated with a blob. Unfortunately, this means you cannot modify any standard http header that is not set by the blob storage service.

SCENARIOS TO USE CUSTOM METADATA

The metadata support for blobs allows you to have self-describing blobs. If you need to associate extra information with a blob (e.g. podcast author, recording location etc.) then this would be a suitable place to store that data without having to resort to an external data source.

Any custom attributes associated with a file (such as the author of a word document) could also be stored in the metadata.

PERFORMANCE TIP

If you need to be able to search for metadata across multiple blobs, then you should consider using an external data source (e.g. a database or the table storage service), rather than searching across the blobs.

Now that we have explored blob properties and metadata.

Now that we've seen how the upload and download operations operate under the covers, let's return to the final part of the uploading and downloading puzzle, namely copying blobs.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

9.6 - Copying Blobs

In this chapter we have so far uploaded files and downloaded files, in and out of our account. However we don't always want to necessarily always transfer outside of our account, sometimes we may wish to take a copy of an existing file within the account.

Let's return to our podcasting example, for an example on why we may wish to do this. Let's say during the conversion of a WMA file to MP3, we decide that we don't wish to make the converted file immediately available. In this case our converted MP3 file would reside in a private container that is not available to the general public. At a later date, we decide to make the file available to the public by moving it into our public downloads container. In this case we wish to copy our converted file from one container to another. Figure 9.10 shows a file being copied from one container to another.

Figure 9.10 – Copying Podcast03.mp3 in the MP3Conversions container to the PublicDownloads container

As we can see from Figure 9.10, we don't want to upload or download the blob, we just wish to copy the blob Podcast01.mp3 from the MP3Conversions container and place the copy in the PublicDownloads container.

Although we could do this using a download followed by an upload, this would be much slower than just doing an internal copy within the data center. Likewise, performing an upload followed by a download would be incredibly slow and wasteful of network resources if the calling client was based outside of the Windows Azure data center.

Listing 9.11 shows the code used to copy a very large file name Podcast03.mp3 to the publicdownloads container from the mp3conversions container via the REST API.

Listing 9.11 Copying a blob between containers

```
      HttpWebRequest hwr = CreateHttpRequest(
         new Uri(@"http://127.0.0.1:10000/devstoreaccount1/publicdownloads/podcast03.mp3"),
         "PUT", new TimeSpan(0, 0, 30));                                                  #1

      hwr.Headers.Add("x-ms-copy-source",
         "/silverlightukstorage/mp3conversions/podcast03.mp3");                           #2

      var account =
         CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

      account.Credentials.SignRequest(hwr);

   hwr.GetResponse();
```

#1 PUT Request to destination container
#2 Source location of the file

Cueballs #1-2 in code and text

As you can see from Listing 9.11 the basics of making the HTTP request is pretty much the same as any other REST request we have made.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In this case we set the destination container at #1 as the URI for the request, and this time we set the request to be a PUT (rather than a GET, HEAD or POST).

At #2 we set the location of the source blob to be copied. The header "x-ms-copy-source" is where we define the location of the file; notice that we include the storage account name (silverlightukstorage), the container (mp3conversions) and the blob name (podcast03.mp3) in the header value.

COPYING FILES VIA THE STORAGE CLIENT LIBRARY

Listing 9.11 uses the REST API directly to the copy the blob, but we can of course make this call by making the much simpler storage client library call:

```
var sourceBlob = sourceContainer.GetBlobReference("podcast01.mp3");
var targetBlob = targeCcontainer.GetBlobReference("podcast01.mp3");
targetBlob.CopyFromBlob(sourceBlob);
```

This code makes the same call as in Listing 9.11, this time using the storage client library instead of using the REST API directly.

NOTE OF SANITY

Although you can use the REST API, using the Storage Client Library is much easier. Save yourself a lot of heartache and only use the REST API when necessary and try and stick to using the lovely storage client library.

Snapshotting Blobs

The problem of copying blobs is that you will need to pay storage costs to the Microsoft Bean Counters for keeping duplicate copies of the data. If you are only copying the blob to maintain some sort of version control, you may wish to consider snapshotting instead. A snapshot will pin a version of your blob at the date and time you created the snapshot. The snapshot is read only (i.e. cannot be modified) and can be used to revert to earlier versions of a blob. Only changes made between versions of snapshots are chargeable.

To create a snapshot, you can make the following call:

var snapshotBlob = blob.CreateSnapshot();

To retrieve a snapshot via the REST API, you can use the following URI

http://accountname.blob.core.windows.net/containername/blobname?snapshot=<DateTime>

or via the storage client

var snapshotBlob = container.GetBlobReference("blobname?snapshot=<DateTime>");

For more details on snapshotting see:

http://msdn.microsoft.com/en-us/library/ee691971.aspx

9.7 Setting shared access permissions

So far in this chapter we have seen that access to a blob is controlled by the container that it lives in. If the blob lives in a public container then it's available to the world. If the blob lives in a private container then you can only access with your private authentication key.

WARNING

Do not distribute your private authentication key, it's a sure fire way to have some pillock trash your data.

These level of access are a little too extreme, we really need a more granular way of controlling access to our blobs, namely shared access signatures. Shared access allow you to basically set a policy on a private container

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

(or blob), and anyone who makes a request with the correct signature can perform the appropriate action on the blob (e.g. download the blob).

> **Container or Blob Level Access**
>
> Although you can assign permissions on an individual blob level, this a pain to maintain. It's easier to maintain permissions at a container level. You can always have a container that consists of a single blob ;)

Let's now return to the Podcast example and look at how we can control download access to one of our podcasts.

9.7.1 Setting shared access permissions on a container (container level access policy)

Let's say our podcasting business has gone well and we have decided to start selling some our podcasts to the general public. In this scenario, once our rich fool has purchased the pocast, we need a method of providing a way for them to download the podcast without making it public (we obviously don't want to give them our owner authentication key). To achieve this we will store our podcast (Podcast03.mp3) in its own private container (Podcast03) that is not available to the general public.

Once our customer has purchased the podcast, we will generate a shared access signature that will give that customer permission to read any blobs (i.e. Podcast03.mp3) in the Podcast03 container, for a period of 24 hours. After the 24 hour period has expired, the customer will no longer be able to download the podcast.

The first thing we need to do is generate a shared access policy that will restrict the download period to the next 24 hours, this is achieved with the following code:

```
 var oneDayDownloadpolicy = new SharedAccessPolicy();
oneDayDownloadpolicy.SharedAccessStartTime = DateTime.Now;
oneDayDownloadpolicy.SharedAccessExpiryTime = DateTime.Now.AddDays(1);
```

As you can see from the code above, we can specify both a start time and an expiry time for the policy. Once we have specified this policy, we need to apply it to the container.

```
var permissions = new BlobContainerPermissions();
permissions.PublicAccess = BlobContainerPublicAccessType.Blob;
permissions.SharedAccessPolicies.Clear();
permissions.SharedAccessPolicies.Add("CustomerA", oneDayDownloadPolicy);
container.SetPermissions(permissions);
```

In the code above we have generated a new set of permissions and applied to the container. The set of permissions state that the customer can read any blobs in the container (`permissions.PublicAccess = BlobContainerPublicAccessType.Blob`) but not list the contents of the container (you can do this if you want by setting PublicAccess to BlobContainerPublicAccessType.Container.

Finally we can generate a URI that customer will be able to use to download the blob using the following code:

```
string sharedAccessSignature = blob.GetSharedAccessSignature(mypolicy);
string uri = blob.Uri.AbsoluteUri + sharedAccessSignature;
```

The generated URI will look something like this:

https://chrishayuk.blob.core.windows.net/podcast03/podcast03.mp3?st=2010-01-04T12%3A08%3A00Z&se=2010-01-05T12%3A08%3A00Z&sr=b&sp=r&sig=ByfV3a1SXOXT04G4GF%2FNQo%2B9cxx4vrRE45kYxbhFhJk%3D

And that's really it; you now have the ability to dynamically assign permission to read blobs in containers.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Assigning other types of permissions

If you want to be able to assign permissions at a blob level (rather than a container level) or if you want to provide more than just read permissions. You can actually generate shared access signatures that gives users permissions to write to certain blobs in your container. This scenario is probably a little detailed for what we would like to show in this book, but feel free to visit the online documentation for more details

http://msdn.microsoft.com/en-us/library/ee395415.aspx

9.8 Summary

And….breathe, it's fairly safe to say we have covered in great depth on how you can manage blobs stored in blob storage.

In this chapter we built upon the knowledge we learned in chapter 8 and looked at the storage client library translates to REST API calls. We've looked at how we can upload and download files using the storage client library and realized how much it does for us. I guess the real take away from that section is that you shouldn't use the REST API and you should stick to the storage client library, otherwise you have to worry about blocks and retries.

Not only did we look at how to use Blob Storage but we also learned how we can integrate with our ASP.NET Website solutions. In those sections we also learned that blob storage is the ideal place to store our ASP.NET Website assets and that Blob Storage is perfectly suited to providing controlled access to content we wish to protect such as paid MP3 files.

In this example we used 2 different methods of serving up content, the http handler and the container level access policy. Although we should probably use the container level access policy whenever possible (it's cheaper for us), the techniques that we learned building the http handler are invaluable for synchronizing access between local storage and blob storage.

In the next chapter, we will take a look at how we can use Blob Storage without necessarily having a web role or worker role. I think you'll agree it opens up some pretty interesting scenarios.

In the previous chapter we focused on managing accounts and containers, in this chapter we showed you how to work with the actual blobs. Not only did we look at how to develop against the blob service but we also showed you how you could use the service with your ASP.NET Websites.

In the next chapter we will look at how the blob storage service could be used without a web role or server and run as a standalone service.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

10

When the BLOB Stands Alone

This chapter covers

- Hosting static website in Blob Storage
- Hosting Silverlight applications in Blob Storage
- Hosting Website Assets in Blob Storage
- Using Blob Storage to Progressively Download Video
- Blob Storage Shared Access

Although Blob Storage is generally used as a durable storage area used by web and worker roles, it can also be used as a standalone service (i.e. you can use the service for your applications without needing to host a worker or web role).

So far in the book we have focused on using web roles to host our websites, we'll now look at how we can use Blob Storage to host a static html website without requiring a web role.

10.1 Hosting Static Html Websites

We learned in chapter 9 that blobs held within a public container are accessible to the outside via a public URI over an HTTP connection. This means that those files can be accessed with standard web browsers such as Internet Explorer. Blob Storage also gives the ability to configure the MIME types associated with contained files meaning that the web browser can correctly handle the served content. These capabilities make Blob Storage more than just a networked hard disk but rather a full-fledged web server farm (as shown in Figure 10.1)

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 10.1 - Blob Storage acting as a Web Farm (look, no Web Roles)

Blob Storage can serve up all modern content types (including standard html pages, JavaScript files, CSS files, images, movies, Word documents, PDF documents, Silverlight applications, Flash applications etc.) making Blob Storage a real consideration for hosting static websites.

Why would I want to use Blob Storage rather than a web role?

Although web roles are the recommended solution for most scenarios, in some instances it may be more cost-effective to use Blob Storage to host your website. If your website is a standard static html website that does not perform any server side processing, then this would be a suitable candidate for hosting in Blob Storage.

For example, informational sites such as local business websites, static corporate websites and Silverlight (or Flash) games are typical candidates for blob storage hosting.

Quite simply if you want a cheap method of hosting websites in a scalable fashion and you don't need to dynamically generate pages, you can save yourself the cost of webroles and just pay per request.

In this section, we'll show you how blob storage hosting works by creating and then publishing a very simple static html website. We'll also look at a couple of issues related to using blob storage as a host. Now let's get to it and get creating a static website that we can host in Blob Storage.

10.1.1 Creating our Static Html Website

The website that we are about to build is a very simple html page with a JavaScript calculator, as shown in Figure 10.2. Actually when we say calculator, it really only adds 2 numbers together, Abacus++ is probably more accurate.

Figure 10.2 - A simple static html website hosted in Blob Storage

After the user enters two values and clicks on the equals button, the result appears (with a gratuitous smiley face graphic) as shown in Figure 10.3.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 10.3 - On click of the equals button, the result is displayed (as well as a badly drawn smiley face).

Although this webpage is a very simple example, it does consist of all the major parts of a typical website i.e. html, CSS, JavaScript and assets (such as images). Table 10.1 gives a quick overview of the contents of our site.

Table 10.1 - A breakdown of all the files in our very complicated website

Website file	Purpose
default.htm	Our html page with buttons, textboxes and stuff
standard.css	All good web designers use CSS for their markup
calculator.js	The complicated JavaScript that adds 2 numbers together
happy.jpg	The smiley face that appears in Figure 9.1

Now that we know what makes up our website let's take a quick look at some of the code. Listing 10.1 contains the html for our page.

Listing 10.1 - Calculator Html

```
<!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>
    <title>Chapter 10 - Static Html Website hosted in Blob Storage</title>
    <link rel="stylesheet" type="text/css" href="standard.css" />       #A
    <script type="text/javascript" src="calculator.js"></script>        #B
    <script type="text/javascript">
        function calculate() {                                          #1
            document.getElementById('result').value =
                add(
                     document.getElementById('number1').value,
                     document.getElementById('number2').value
                    );

            document.getElementById('piccy').className = "visible";
        }
    </script>
</head>
<body>
    <div class="standarddiv">
        A very simple calculator
    </div>
    <div class="standarddiv">
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
        <input id="number1" type="text"
               value="5" class="standardtextbox"/>

        <span>+</span>
        <input id="number2" type="text"
               value="10" class="standardtextbox"/>

        <input type="submit" value="="
               class="standardbutton"
               onclick="calculate();"/>                                  #2

        <input id="result" type="text"
               class="standardtextbox"
               readonly="readonly"/>

        <div id="piccy" class="hidden">                                  #3
            <img src="happy.jpg"/>
        </div>
    </div></body>
</html>
```

#A - Reference to CSS File
#B - Reference to JavaScript File
#1 - Calculates the value
#2 - Calculate Button
#3 - Happy Picture

Cueballs #1-6 in code and text

I'm not gonna explain the code above too much. It's fairly simple html and this is chapter 10, if you've got this far you've probably figured out html already. However the important bits are, on click of the "=" button at #2 in Listing 10.1, the JavaScript function at #1 is invoked. This function takes the contents of the two textboxes (number1 and number2), calls the add function and displays the result in the result textbox. Finally, once the calculation is complete, the style at #3 is changed from "hidden" to "visible" displaying the smiley face graphic as seen in Figure 10.3.

VIEW THE SAMPLE IN THE CLOUD

http://www.azureinaction.com/samples/chapter10/

Now that we've have pushed technology to its absolute limit by building a web page that can add numbers together (woo, go us), we should look at how we can publish our work of art to blob storage for hosting.

10.1.2 Publishing our Website to Blob Services

In previous examples we have programmatically added our Blobs to the Storage Service, in this example we will make use of a tool called "Chris Hay's Azure Blob Browser" (can you guess from the title who created it), to upload the files to Blob Storage. Figure 10.4 shows the addition of the calculator.js file to Blob Storage via the tool.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 10.4 - Adding calculator.js to Blob Storage via Chris Hay's Blob Browser

Using the blob browser, we can add each file of the website (default.htm, calculator.js, standard.css and happy.jpg) by selecting the file to upload, setting the mime type (which is automatically predicted for you but is also editable) and clicking the add button.

THAT'S A COOL BLOB BROWSER

"Chris Hays' Azure Blob Browser", its source, the example website, and a working example of the code are available at http://www.azureinaction.com/

Once all the files are uploaded into appropriate public container, you can access the website with a standard web browser using the following URI structure.

Error! Hyperlink reference not valid.

Please note that you must include the filename (default.htm) as Blob Storage does not support default documents.

Custom Domain Names

If we are hosting our website or web application in blob storage, we are unlikely to want our clients to use the blob.core.windows.net domain. We really want to host our website on our own custom domain e.g. http://www.noddyjavascriptcalculator.com/

For details on how to do this, see Chapter 8.

SETTING THE CORRECT MIME TYPE

If the correct MIME Type is not set then the web browser will not be able to render the downloaded file. Some browsers such as Internet Explorer will still render the content in some situations even if the MIME type is incorrect, other browsers such as Firefox require the MIME type to be set explicitly.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 10.5 shows how Firefox handles html documents that do not have the correct mime type (text/html) but instead use the default Azure Blob Storage Services MIME type (application/octet-stream). As you can see from the screenshot Firefox cannot display the page and requires further instructions from the user. If the MIME type is set correctly Firefox will show the same web page as shown in Figure 10.2

Figure 10.5 - Accessing a file with the incorrect MIME type in Firefox

Now that we have our first static html website running in Blob Storage, let's look at how we can move beyond flat directory structures and store and retrieve files within hierarchies.

HANDLING DIRECTORY STRUCTURE

In our calculator example, we used the following flat directory structure for our files.

- Default.htm
- Calculator.js
- Standard.css
- Happy.jpg

Although this is useful for small websites, it is very limiting for websites with a large hierarchy. Blob Storage does not technically support directories within a container but this can be simulated by using a "/" as a separator in the blob name. The following list shows how we can represent our files in Blob Storage (with hierarchy).

- Chapter5/Default.htm
- Chapter5/javascript/calculator.js
- Chapter5/css/standard.css
- Chapter5/images/happy.jpg

Using the "/" separator in the blob name means that the Default.htm page would be accessed using the following URI structure:

Error! Hyperlink reference not valid.

CloudBlobDirectory

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The CloudBlobDirectory class in the Storage Client Library abstracts some of this complexity by providing a set of methods that allows you to pretend there is a real directory structure in a container.

Now that we can host our static html websites in Blob Storage, we need to look at the some of the issues involved with using Blob Storage as a web server.

Analytics

One issue with the Blob Storage service is that there is no logging of blob requests (as of version 1). This level of logging is vital for analyzing who is visiting your website, where they have come from and what parts of your site are popular. If you are used to running your own website you will typically have access to your IIS logs, which contain information such as Page Requested, IP Address, and Referrer etc. Since the Blob Storage service does not provide such information you would need to use an external analytics service such as Google Analytics.

In this section, we showed you how to display static html websites within Blob Storage services. Next, we will look at hosting Silverlight applications.

10.2 Hosting Silverlight Applications in Blob Storage

Although you will typically host your Silverlight applications in a standard Windows Azure web role, you can host your Silverlight applications in Blob Storage if required.

Depending on your web application you can effectively utilize Blob Storage to achieve massive scale for minimal cost. For example, in our podcasting example from the previous chapters it may make financial sense to have your website hosted in a web role, but have a Silverlight media player hosted within Blob Storage.

Another example could be, if you run a foreign exchange website, you may wish to host the main site (which could offer an ability to purchase currency) within a web role, but host an exchange rate calculator within Blob Storage.

Other RIA Solutions

In this chapter, we are focusing on Silverlight as this is more familiar to .NET Developers. Although we are looking exclusively at Silverlight, these techniques are applicable to other RIA solutions (e.g. Flash, Flex, Air etc).

In this section we will look at how we can enable such scenarios by showing you how you can use blob storage to:

- Host a standalone Silverlight Application contained in a static html page in Blob Storage
- Host a standalone Silverlight Application contained in a ASP.NET Web Page where the Silverlight Application is hosted in Blob Storage
- Have your Blob Storage hosted Silverlight Application communicate with external web services
- Have an external Silverlight Application access files stored in Blob Storage

We'll now get started looking at how we can host a Silverlight Application in Blob Storage

10.2.1 Hosting the Silverlight Spectrum Emulator

If you have developed a standalone Silverlight application that requires no interaction with any backend services, then Blob Storage is a cost effective candidate for hosting your Silverlight application. These types of applications typically include games, tax calculators and other widgets.

In this next example we will host a small Silverlight Application in Blob Storage. The application that we will host is a Silverlight ZX Spectrum Emulator (an 8-bit home computer of the 1980's, very like a Commodore 64 (actually that's not true it was much worse but it had spirit)). Figure 10.6 shows the emulator running from Blob Storage.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 10.6 - Silverlight application running in Blob Storage

The spectrum emulator not only allows you to play games from the 1980's but as you can see from Figure 10.6, you can even write BASIC programs in it. Now I don't want you to lose focus on this book, as soon as you load this thing up. As cool as Jet Set Willy is, it won't help you delivery your amazing cloud based application,

CAN I GET THE SOURCE FOR THE EMULATOR?

You can download this Silverlight Application (ZXSilverlight.xap) and its source code from

http://www.azureinaction.com

To store the application within Blob Storage, you can upload ZXSilverlight.xap using the same procedure as we used in the previous section. The blob must be set with the correct MIME type otherwise the browser will not be able to launch the application (Figure 10.4 shows the handling of an incorrect MIME type in Firefox).

THE MIME TYPE FOR A SILVERLIGHT APPLICATION

application/x-silverlight-app

In the ZX Spectrum Emulator example we also store the html page that hosts the Silverlight Application within Blob Storage. Listing 10.2 shows the html used to run our ZX Spectrum Emulator Silverlight application (ZXSilverlight.app).

Listing 10.2 - Silverlight Html

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>Silverlight Project Test Page </title>

    <style type="text/css">
        html, body {height: 100%; overflow: auto;}
        #silverlightControlHost {height: 100%;}
    </style>
    <script type="text/javascript" src="Silverlight.js"></script>
</head>
<body>
     <div id="silverlightControlHost">
            <object data="data:application/x-silverlight,"              #A
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
                        type="application/x-silverlight-2"                  #B
                        width="100%" height="100%">
                       <param name="source" value="ZXSilverlight.xap"/>     #C
                       <param name="background" value="white" />
                 </object>
        </div>
    </body>
    </html>
```

#A - Specifies the Silverlight plug-in
#B - Specifies the Silverlight 2 application
#C - Specifies the location of the Silverlight application

As we can see from Listing 10.2, the html page doesn't really have any logic in it. Its job is just to host the Silverlight Application. In this case where our web page is just a host for a Silverlight Application then you can see using a fully-fledged web server is a little over the top (and expensive) where a static html page hosted in Blob Storage does the job perfectly.

Accessing Blob Storage hosted Silverlight Applications in non-Blob Storage hosted Web Pages

If you have an existing web page that you host outside of Blob Storage (e.g. Web Role or existing Web Hosting Provider) then you can still host your Silverlight Application in Blob Storage but keep your site with your existing host. In this case you just need to change the source parameter of the Silverlight Plugin (listing 10.2) to point to your Blob Storage URI.

This standalone Silverlight application doesn't require access any backend web services but what if you want to host an application that does? You can still host your application in blob services, but you'll need to understand how a cross domain policy works.

10.2.2 Communicating with third party sites

Suppose your Silverlight application requires communication with an external Web Site or Web Service (WCF, ASMX, REST, POX, HTTP); your Silverlight Application (and its hosting html page) could still be hosted in Blob Storage. An applicable scenario could be that you want to host a Silverlight exchange rate calculator within Blob Storage, but you want your calculator to communicate with a live exchange rate web service that is hosted in a web role or by an external site. Figure 10.7 shows a Silverlight Application communicating with an external web service.

Figure 10.7 - Silverlight Application communicating with external web service

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In order for the Silverlight application to communicate with a third-party domain, the external site must host a suitable cross domain policy. Listing 10.3 shows a typical Cross Domain Policy file (ClientAccessPolicy.xml) used to give permissions to a Silverlight Application.

Cross Domain Policy

For security purposes, Silverlight Applications (and Flash Applications) can by default only communicate with domain that the container web page is hosted on. If your Silverlight Application needs to communicate with a third party domain then the external website will need to host a cross domain policy file (ClientAccessPolicy or CrossDomain.Xml), to give your application permission to communicate with it.

Listing 10.3 - ClientAccessPolicy.xml

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="*">
      <domain uri="http://silverlightukstorage.blob.core.windows.net/"/>#1
      </allow-from>
      <grant-to>
        <resource path="/" include-subpaths="true"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

#1 URI of blob storage domain

Cueball #1 in code and text

The clientaccesspolicy.xml file displayed in listing 10.3 states at #1 any Silverlight application hosted at http://silverlightukstorage.blob.core.windows.net/ (the blob storage account of our Silverlight application) can access the third party web service. Our application would also be able to access the service if the domain Uri at #1 was set to * (i.e. any Silverlight application hosted at any website can access it).

ClientAccessPolicy.xml is a cross domain policy file that is used purely by Silverlight Applications. Silverlight Applications can also access web services that host a crossdomain.xml file (this is a format is supported by both Flash and Silverlight). Listing 10.4 shows a crossdomain.xml file that our Blob Storage hosted application could communicate with.

Listing 10.4 - CrossDomain.xml

```
<?xml version="1.0" ?>
<!DOCTYPE cross-domain-policy  SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-
policy.dtd">
<cross-domain-policy>
    <allow-access-from domain="*" />                                    #1
</cross-domain-policy>
```

#1 Allow Access to all Domains

Cueball #1 in code and text

The CrossDomain.xml file displayed in Listing 10.4 states at #1 that any website hosting a browser based application (Silverlight or Flash) can communicate with this service. If the owner of the Web Service wished to restrict access to your application only then it would replace the * at #1 with silverlightukstorage.blob.core.windows.net (your blob storage account).

$root Container

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

If you need to allow third party Silverlight / Flash applications to access assets held in your Blob Storage Account, you should store your crossdomain.xml or clientaccesspolicy.xml file in a public container named $root

This special container allows you to store any file in the root of your URI, e.g.

http://chrishayuk.blob.core.windows.net/crossdomain.xml

Let's now build a Silverlight Web Search Application that will use Yahoo's Search API's to search the Internet. Figure 10.8 shows the Silverlight Application that we will build.

Figure 10.8 - A Silverlight Search Application hosted in Blob Storage that communicates with the Yahoo Search API.

The Html page for the application that we will build (as shown in Figure 10.8) is no different from the one in Listing 10.2, the only difference is that the Source Parameter changes to reference our new Silverlight Application (.XAP file).

To create the application, you would create a new Silverlight Application as normal, and change the XAML to the XAML shown in Listing 10.3

Listing 10.3 - XAML for Silverlight Search Application

```
<Grid x:Name="LayoutRoot" Background="White">
        <StackPanel>
            <StackPanel Orientation="Horizontal">
            <TextBlock  Text="Search Yahoo"
                        VerticalAlignment="Top"
                        Margin="5"/>
            <TextBox x:Name="txtSearch"
                                VerticalAlignment="Top"
                                Height="25" Width="200"
                                Margin="5"/>
            <Button x:Name="btnSearch"
                    Content="Search"
                    VerticalAlignment="Top"
                    Height="25" Width="50"
                    Margin="5"
                    Click="btnSearch_Click"/>
        </StackPanel>
            <ItemsControl x:Name="itemsResults">
                <ItemsControl.ItemTemplate>
                    <DataTemplate>
                        <Grid Margin="10">
                            <Grid.RowDefinitions>
                                <RowDefinition Height="25"/>
                                <RowDefinition Height="*"/>
                            </Grid.RowDefinitions>
                            <HyperlinkButton
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
                                Grid.Row="0"
                                Content="{Binding Title}"
                                NavigateUri="{Binding Url}"/>
                            <TextBlock
                                Grid.Row="1"
                                Text="{Binding Summary}"
                                TextWrapping="Wrap">
                            </TextBlock>
                        </Grid>
                    </DataTemplate>
                </ItemsControl.ItemTemplate>
            </ItemsControl>
        </StackPanel>
    </Grid>
```

Once you have pasted the XAML in Listing 10.3 into your Silverlight page, you can now hook up the codebehind for the page as shown in Listing 10.4

Listing 10.4 - Codebehind for Yahoo Search XAML page

```
public partial class YahooSearch : UserControl
{
        public YahooSearch()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Search yahoo
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnSearch_Click(object sender, RoutedEventArgs e)
        {
            // Initialise
            Search mySearch = new Search();
            mySearch.SearchResultsReturned += new
Search.SearchResultReturnedDelegate(mySearch_SearchResultsReturned);
            mySearch.Execute(txtSearch.Text);                          #1
        }

        void mySearch_SearchResultsReturned(SearchResultSet results)
        {
            itemsResults.ItemsSource = results.Results;                #2
        }
}
```

In the codebehind for the Silverlight page shown in Listing 10.3, we can see that when the Search button is clicked, we will invoke execute method of our Search class at #1. This will then send our search term to the Yahoo Search API, and then return the results asynchronously. On receiving the search results, the results are bound to the datagrid as shown at #2.

Yahoo Search

Yahoo used to be a very popular search engine many years ago; this was at a time before we Googled for things ;). Here at Azure in Action, we like to support the little guy, come on Yahoo, you can be big again.

Actually the reason we use Yahoo, is that it has a very nice, simple REST based API that we can easily use from Silverlight.

Listing 10.5 shows the code for our Yahoo Search class.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Listing 10.5 - Yahoo Search Class

```
namespace SLAssaultCourseSamples.NetworkingSamples
{
    public class Search
    {
        public delegate void SearchResultReturnedDelegate(SearchResultSet results);
        public event SearchResultReturnedDelegate SearchResultsReturned;

        private const string _baseUri =
"http://search.yahooapis.com/WebSearchService/V1/webSearch?appid={0}&query={1}&start={2}&res
ults={3}";
        private const string _applicationId =
@"Jk.kKH_V34FIj5WGgtm6iimK.37hiKF_0A5EVnJkoltzGoydU.Z0notpjqa0DxTiJULlbg--";

            /// <summary>
            /// Searches yahoo for the keyword
            /// </summary>
            /// <param name="keyword">keyword to search</param>
            /// <returns></returns>
            public void Execute(string keyword)
            {
                // Construct the search uri
                Uri searchUri = new Uri(string.Format(_baseUri, _applicationId,
keyword,1,5));

                // Call the search
                Execute(searchUri);
            }

            /// <summary>
            /// Performs the search with the given uri
            /// </summary>
            /// <param name="address"></param>
            private void Execute(Uri address)
            {
                // Initialise
                WebClient searchClient = new WebClient();

              //  searchClient.
                // Setup the event handler
                searchClient.DownloadStringCompleted += new
DownloadStringCompletedEventHandler(searchClient_DownloadStringCompleted);

                // Download the data as a string
                searchClient.DownloadStringAsync(address);
            }

            /// <summary>
            /// On completion of the search results
            /// </summary>
            /// <param name="sender"></param>
            /// <param name="e"></param>
            void searchClient_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
            {
                // Check if we have someone hooked up to the event
                if (SearchResultsReturned != null)
                {
                    // Parse the Xml
                    XElement xeResult = XElement.Parse(e.Result);

                    // Initialise
                    SearchResultSet resultSet = new SearchResultSet(xeResult);

                    // return the results
                    SearchResultsReturned(resultSet);
                }
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
        }
    }

    public class SearchResultItem
    {
        public string Title { get; set; }
        public string Summary { get; set; }
        public Uri Url { get; set; }
    }

    public class SearchResultSet
    {
        /// <summary>
        /// Constructor
        /// </summary>
        public SearchResultSet()
        {
            // Initialise the list
            Results = new List<SearchResultItem>();
        }

        /// <summary>
        /// Constructor
        /// </summary>
        public SearchResultSet(XElement resultsXml)
        {
            // Initialise the list
            Results = new List<SearchResultItem>();

            // Set the namespace
            XNamespace ns = "urn:yahoo:srch";

            // Get the results
            var xeResults = from xeResult in resultsXml.Elements(ns + "Result")
                            select new SearchResultItem
                            {
                                Title = xeResult.Element(ns + "Title").Value,
                                Summary = xeResult.Element(ns + "Summary").Value,
                                Url = new Uri(xeResult.Element(ns + "Url").Value)
                            };


            // Add the results to the list
            Results.AddRange(xeResults);
        }

        public List<SearchResultItem> Results { get; set; }

    }
```

I'm not really gonna spend any time explaining the code in Listing 10.5, except to say that it essentially makes an http request to the Yahoo Search API and returns a set of results that you can bind to your Silverlight Datagrid.

Isn't Yahoo an third party service?

Yes, but because the domain that hosts the Yahoo Search API (search.yahooapis.com), hosts a crossdomain.xml file (http://search.yahoo.apis.com/crossdomain.xml), our Silverlight Application can communicate with it ☺

Now that we have an idea of how we can store static html websites and Silverlight Applications in Blob Storage. Let's now look at how we can use Blob Storage as a media server.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

10.3 Building a progressive downloading video podcast player

In this section, we will return to our podcasting sample. Previously we have used blob storage as a place to store our video and audio podcasts; we now wish to use Blob Storage as a mechanism for serving our videos to our customers. Let's look at how this is done.

VIDEO DOWNLOADS

Since Blob Storage provides us with a URI for any files held in public containers, we could just make this link available to our customers to download the files offline.

```
http://storageaccount.blobs.core.windows.net/container/mypodcast.wmv
```

Although you can allow your customers to download and view the file on their local machine this is probably not the greatest user experience in the world.

STREAMING

An alternative to downloading an entire media file is the use of streaming. When media is streamed, the streaming server will start sending a byte stream of the video to the client. The client media player will create a buffer of the downloaded bytes, and start playing the video once the buffer is sufficiently full. Whilst the user is watching the video from the buffer, the client will continue to download the data in the background.

Media Streaming means that the user can start watching the video within seconds rather than having to wait for the entire movie to download. If the user decides only to watch the first few seconds of the movie then the service provider will only have served up some of the movie, meaning cheaper bandwidth bills.

STREAMING NOT SUPPORTED IN WINDOWS AZURE BLOB STORAGE

Unfortunately in Windows Azure and Windows Azure Blob Storage Service streaming is not currently available as an option. If you wished to use such a solution, you would need to use a third party streaming service.

PROGRESSIVE DOWNLOADING

Progressive downloading is where a file is downloaded in smaller chunks and is stitched together by the client application. Once a few chunks are downloaded, the client application can start playing the movie while the rest are downloaded in the background. Progressive Downloading has the same performance advantages as streaming and provides a similar user experience.

The main difference between progressive downloading and streaming is that the file being streamed will never physically reside on disk and will only ever remain in memory.

10.3.1 A Silverlight / WPF Video Player

By default Silverlight supports the ability to progressively download files. In this example we will build a small Silverlight Video Player that will allow you to play blob storage hosted movies on your web page. Figure 10.9 shows a small Video player that we will build that is playing a video served directly from Blob Storage using the public URI.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 10.9 - WPF Video Player showing a video featuring a giant bunny rabbit (Big Buck Bunny)

Let's look at how we can create the video player shown in Figure 10.9.

CREATING THE PLAYER

A WPF / Silverlight media player (as shown in Figure 10.9), can be built by creating a new or WPF or Silverlight application and using the following XAML.

```
<MediaElement x:Name="myVideo"
Source="http://storageaccount.blobs.core.windows.net/container/videopodcast01.wmv"
/>
```

Wow, is that really it? Yup, that's how easy it is to create a progressive downloading video player that shows videos hosted in Blob Storage.

Although Silverlight does progressively download the Video Player, it is performed in a linear fashion downloading from a single file, from a single website. If you have very large files (gigabytes) you may wish to use a slight different technique. In these cases, we can download our video much faster if we use split up and chunk our files manually (and eventually split across multiple servers.

> **Adaptive Streaming**
>
> The following technique is what most CDN use to delivery high performance video. This is also same basic technique used in IIS Adaptive Streaming (although we won't deliver at multiple bitrates)

10.3.2 A WPF Based Adaptive Streaming like Video Player

We will now look at how we can build a WPF based Media Player that can progressively download our video. Our media player will look exactly like the WPF media player displayed in Figure 10.9.

> **WARNING - NOT PRODUCTION CODE**
>
> The following code is really just an example on how you can perform this technique. This is not production quality code (by a long shot) ☺

THE XAML

The following XAML is used to play our video in WPF:

```
<MediaElement x:Name="myVideo" Source="videopodcast01.wmv"
              LoadedBehavior="Manual"/>
```

MediaElement is a built-in control that allows you to play movies in your WPF applications. In our example we will be downloading our movie (mymovie.wmv) to the same folder that the WPF movie player application resides in. The source attribute states where the MediaElement should look for our movie.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

By default the MediaElement will automatically start playing the movie on startup. Since our movie will have not yet been downloaded, we need to prevent this automatic behavior which is achieved by setting the attribute LoadedBehavior to "Manual".

THE CHUNKING METHODOLOGY

In our movie player we will download our movie in chunks of 100Kb. Only one chunk will be downloaded at a time and each newly downloaded chunk will be appended to the previously downloaded chunk. As you can see from Listing 10.6, as soon as the movie player is loaded, it will start downloading the chunks and then after 10 seconds the movie will start playing.

Listing 10.6 - On load of the WPF Movie Player

```
private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
   movieStream =
      File.Open("videopodcast01.wmv",
         FileMode.Append, FileAccess.Write, FileShare.ReadWrite);     #A

   GetNextChunk();                                                     #B
   Thread.Sleep(10000);                                                #C
   myVideo.Play();                                          #D
}
```

#A - Create a new empty movie file
#B - Start downloading the video from Blob Storage
#C - Wait for 10 seconds
#D - Start Playing the video

USING THE RANGE HEADER

Before we look at the code used to progressively download our video, it is worth looking at the range header.

By default when making a GET request using either the storage client or via an HTTP request the entire file will be downloaded. The following code will download "videopodcast01.wmv" from the public container "podcasts".

Listing 10.7 - Using the storage client to download a video file

```
  CloudStorageAccount account =
     CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

   CloudBlobClient blobClient =
      account.CreateCloudBlobClient();

   CloudBlobContainer container =
      blobClient.GetContainerReference("podcasts");                     #A

   container.GetBlobReference("videopodcast01.wmv");                    #B
   container.DownloadFile("videopodcast01.wmv");                        #C
```

#A - Get the podcasts container
#B - Get a reference to the blob videopodcast01.wmv
#C - Download videopodcast01.wmv blob from the podcasts container

The code in Listing 10.7 will download the entire movie, in this example we really want to split the movie up into manageable chunks. At present the storage client sample code doesn't give the ability to download a specified portion of the file even though the underlying REST API does support this. The following code will allow us to download the entire podcast using HttpWebRequest.

Listing 10.8 - Using HttpWebRequest to download a video file

```
   string baseUri = @"http://silverlightukstorage.blob.core.windows.net/";
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
HttpWebRequest hwr =                                                    #1
     CreateHttpRequest(new Uri(baseUri +"podcasts/videopodcast01.wmv"),
                            "GET", new TimeSpan(0, 0, 30));
// TODO: Range Header goes here                                         #2
DownloadFile(hwr," videopodcast01.wmv");                                #3
```

At #1 we will generate the HttpWebRequest with the standard headers required using the CreateHttpRequest method that we used in Chapter 5. At #2 we use the DownloadFile method to invoke the request and download the file to disk. The implementation of the DownloadFile method is available in the online samples.

If we do not want to download the entire video file, we can use the "range" header to specify the range of bytes that we wish to download. The following code would restrict the download to the first 100,000 bytes of the file, and can be used in Listing 10.8 at #2

```
hwr.AddRange(0, 100000);
```

DOWNLOADING CHUNKS OF DATA

On loading of our WPF movie player, we made a call to GetNextChunk()in Listing 10.6. In Listing 10.9 we now show you how GetNextChunk is implemented, and how we use the range header to progressively download the movie.

Listing 10.9 - Downloading the data in chunks

```
private void GetNextChunk()                                             #1
{
string baseUri = @"http://silverlightukstorage.blob.core.windows.net/";
HttpWebRequest hwr =                                                    #2
     CreateHttpRequest(new Uri(baseUri +"podcasts/videopodcast01.wmv"),
                            "GET", new TimeSpan(0, 0, 30));

hwr.AddRange(nextRange, nextRange + size);                              #3
nextRange += (size + 1);
hwr.BeginGetResponse(new AsyncCallback(webRequest_Callback), hwr);      #4
}

private void webRequest_Callback(IAsyncResult asynchronousResult)       #5
{
   try
   {
      HttpWebRequest request =
         (HttpWebRequest)asynchronousResult.AsyncState;

      HttpWebResponse response =
         (HttpWebResponse)request.EndGetResponse(asynchronousResult);

      SaveChunk(response.GetResponseStream());                          #6
      response.Close();

      GetNextChunk();                                                   #7
   }
   catch { }
}

private void SaveChunk(Stream incomingStream)                           #8
{
   int READ_CHUNK = 1024 * 1024;
int WRITE_CHUNK = 1000 * 1024;
byte[] buffer = new byte[READ_CHUNK];

Stream stream = incomingStream;

while (true)
{
   int read = stream.Read(buffer, 0, READ_CHUNK);
   if (read <= 0)
      break;
   int to_write = read;
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
      while (to_write > 0)
      {
         movieStream.Write(buffer, 0, Math.Min(to_write, WRITE_CHUNK));
         to_write -= Math.Min(to_write, WRITE_CHUNK);
      }
   }
   }
```

The GetNextChunk method shown at #1 will download 100Kb of data from the Blob Storage service asynchronously. This method will be called every time we need to download a new chunk of our video; it is first called in the loading of our application at #B in listing 10.6.

At #2 we create our standard http web request; this is the same method that we used to create requests earlier. The video (videopodcast01.wmv) that we will be downloading resides in a public container called "podcasts" held in our storage account. Once we have created the request, at #3 we will add our range header to restrict our download to the next 100Kb chunk.

At #4 we will make our asynchronous http web request to the blob storage service, with a callback to #6. On the request callback at #5, we will retrieve the downloaded data and at #6 we will call the SaveChunk method (implemented at #8), which will append the downloaded chunk to the "videopodcast01.wmv" file on the local disk. Finally at #7 we will call GetNextChunk at #1 to Get the next chunk of data.

Time Based Buffering

In our very simple example we used time based buffering to delay the playback of the movie. Although is this is suitable for our example, in a production scenario you should start playing the movie after a certain portion of the movie has been downloaded. In our example we also do not make any provision for variable download speeds so you may wish to extend the sample to handle situations where the playback speed is faster than the download speed.

So we now have a working version of a WPF progressive downloading media player. This shows that we can use Blob Storage as a storage service for desktop clients and with some tweaks build media players that can give similar performance benefits to a streaming service.

10.3.3 A Silverlight based chunking media player

In the previous section we showed you how to build a desktop progressive downloading media player for video podcasts. Although the desktop application is great for that richer client experience, we also want to provide a web based cross platform Silverlight experience.

THE XAML

Firstly we need go back to the Silverlight Player that we created earlier and then modify the XAML to the following.

```
<MediaElement x:Name="myVideo" AutoPlay="False"/>
```

Although the XAML is quite similar to the WPF version there are some subtle differences. In the Silverlight version we are using AutoPlay="false" to specify that we do not want the video to play automatically, in the WPF version we set LoadedBehavior="Manual". We have also not set a source for the MediaElement as we will do this programmatically in the code-behind.

Listing 10.10 shows the on load event of the Silverlight Media Player.

Listing 10.10 - On load of the Silverlight Media Player

```
private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
            DeleteFile();
            GetNextChunk();                                    #1
            Thread.Sleep(10000);                               #2
            using (IsolatedStorageFile isf =                   #3
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
                IsolatedStorageFile.GetUserStoreForApplication())
                 {
                     var iosf = isf.OpenFile("videopodcast01.wmv", FileMode.Open,
FileAccess.Read, FileShare.ReadWrite);
                     myVideo.SetSource(iosf);
                     myVideo.Play();
                 }
}
```

At #1 we call GetNextChunk to start downloading the chunks of data and at #2 we delay playing the video for 10 seconds by sending the thread to sleep. At #3 we start playing back the video from Isolated Storage.

WARNING

Sending the thread to sleep is bad production practice as you should never send the UI thread to sleep. In production you should use a background timer that monitors the download progress, and start playing the video back at a suitable threshold.

THE CHUNKING METHODOLOGY

To chunk the data we will be using the same methodology that we used in the WPF example i.e. split the data into manageable chunks of data and then stitch the chunks back together client side.

Restricted Headers

The browser plug-in model prevents the use of certain request headers as they are considered reserved and are only available to the browser. The restricted headers that are used by Blob Storage APIs and are unavailable to use in Silverlight include:

- Authorization (see Shared Access Section)
- Date (can use x-ms-date as a workaround)
- Range (can use x-ms-range as a workaround)

Although we could split the files using the x-ms-range header (instead of the range header) as we did in the WPF version. It might be kind of interesting to presplit the files into chunks and store the chunks in Blob Storage.

To perform this splitting we could use the same the code in the WPF example but instead of stitching the file back together we can save a separate file for each chunk. For example VideoPodcast01.wmv would be split into the following:

- VideoPodcast01_1.wmv
- VideoPodcast01_2.wmv
- ..
- VideoPodcast01_12.wmv

Finally we would need to upload each chunk to Blob Storage using a tool such as "Chris Hay's Blob Explorer".

What advantages does pre-splitting give me?

By presplitting the files, we could potentially distribute the files across a greater number of servers (or even domains). This would be very useful if we decided to use a CDN (see later) in combination with blob storage. Finally we could instruct the browser to cache the chunks (see previous chapters), meaning that if the file hasn't been fully downloaded, the next download will be much quicker.

To use the presplit version of the chunks our Silverlight Application would therefore use the code in Listing 10.11 to replace the WPF version of the GetNextChunk method.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Listing 10.11 - Silverlight Version of GetNextChunk

```
private void GetNextChunk()
{
string baseUri = @"http://silverlightukstorage.blob.core.windows.net/";
string videoUri = baseUri+"podcasts/videopodcast01_"+nextRange+".wmv";  #1
nextRange++;
HttpWebRequest hwr =
     CreateHttpRequest(new Uri(videoUri), "GET", new TimeSpan(0, 0, 30));
hwr.BeginGetResponse(new AsyncCallback(webRequest_Callback), hwr);
}
```

In Listing 10.11 at #1 we are manually requesting the chunk stored in our container by continually incrementing the chunk in the URI for each download request.

Adaptive Streaming

The major difference between this example and adaptive streaming, is that we don't have multiple encodings of our video at different bitrates. You could easily modify these samples to have videos at different bit rates and detect the best bitrate for the client.

SAVING TO ISOLATED STORAGE

Silverlight can only directly access the file system via a user initiated action. Silverlight applications can however write data directly to an Isolated Storage area without user initiation.

Saving to the actual file system

If you don't mind user initiated actions then you can always use the SaveFileDialog rather than Isolated Storage.

Be-aware that if you are downloading files of any real size, you will probably need a user initiated action to increase the isolated storage default limits. In this case you are probably as well to use the SaveFileDialog.

In the Silverlight version of the media player we will use the Isolated Storage area to save the video locally. Although the video podcast has been presplit into chunks, we still have to combine the chunks into a single file so our media player can playback the video. The method of saving and combining the chunks is similar to the WPF version except we use Isolated Storage. Listing 10.12 shows the Silverlight version of SaveChunk.

Listing 10.12 - Save Chunk

```
private void SaveChunk(Stream incomingStream)
{
int READ_CHUNK = 1024 * 1024;
            int WRITE_CHUNK = 1000 * 1024;
            byte[] buffer = new byte[READ_CHUNK];
            using (IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication())                        #A
            {
                using (IsolatedStorageFileStream isostream = new
IsolatedStorageFileStream("videopodcast01.wmv", FileMode.Append, FileAccess.Write,
FileShare.ReadWrite, isf))                        #B
                {
                    Stream stream = incomingStream;
                    stream.Position = 0;

                    while (true)
                    {
                        int read = stream.Read(buffer, 0, READ_CHUNK);
                        if (read <= 0)
                            break;
                        int to_write = read;

                        while (to_write > 0)
                        {
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
                                isostream.Write(buffer, 0, Math.Min(to_write, WRITE_CHUNK));
                                to_write -= Math.Min(to_write, WRITE_CHUNK);
                            }
                        }
                        isostream.Close();
                    }
                }
    }
```

#A - Get the isolated storage area for the Silverlight Application
#B - Append the chunk to the video in isolate storage

Although we have had to do some slight workarounds we now have a working version of the media player in Silverlight.

SOURCE CODE AND WORKING EXAMPLE

The full source code and a running sample of the Silverlight Media Player can be accessed from http://www.azureinaction.com/

Now that we have built our Silverlight Video Player let's look at how we can improve the performance of the video delivery by looking at Content Delivery Networks.

10.4 Content Delivery Networks (CDN's)

If your website customers are geographically dispersed then making use of a Content Delivery Network (CDN) can significantly improve the user experience. In this section we will discuss Content Delivery Networks and how they can be used in conjunction with the Windows Azure Blob Storage Service.

10.4.1 What is a CDN?

A Content Delivery Network is a large set of web servers that are distributed across the world. Those Web Servers typically sit very close to the Internet Backbone and can serve up very large files, very quickly. When a user makes a request to the CDN for a file, the CDN will work out which Data Center in the CDN is closest to your location, and serve up the content from that Data Center.

In figure 10.10, you can see that if you are based in Edinburgh then your files would be served from the Dublin Data Center rather than one in Hong Kong.

Figure 10.11 - A Content Delivery Network will deliver the files using the server closest to you

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

EDGE SERVERS

If we return to Figure 10.11, we can see that the CDN network has the follow Data Centers

- Los Angeles
- New York
- Dublin
- London
- Dubai
- Hong Kong
- Tokyo

These servers represented by circles in Figure 10.11 are known as Edge Servers. An Edge Server is simply the name for one of the geographically dispersed web servers that are responsible for serving your content.

ORIGIN SERVER

The origin server is the web server that contains the original version of the content being distributed. In figure 10.11 this represented by the triangle and the origin server resides in Washington State. For Windows Azure the origin server could either be your Web Role, or a public container in your blob storage account. The content held on your origin server would never be accessed directly by your end user as the origin server will only serve content to the Edge servers.

When a request is made for a file held in a CDN, the request will be redirected to the nearest Edge server. If the Edge server does not have a local copy of your file then it will request it from the origin server and cache it locally.

What type of files can I distribute on a CDN?

A CDN Web Server is not unlike one of the Blob Storage Web Servers as it can serve up any static file with the correct MIME type but it has no backend server processing capabilities. A CDN is therefore suitable for serving up static content such as (html, JavaScript, CSS, Silverlight Applications, Flash Applications, PDF documents, Audio Files, Videos etc).

10.4.2 CDN Performance Advantages

Use of a CDN network is a simple method of improving performance on your web servers without significantly changing your architecture or code.

REDUCING LOAD FROM YOUR WEB APPLICATION SERVERS

If your static content is offloaded from your web application servers then your web servers will have more capacity to handle incoming requests. The reduced load will have the following effect:

- Reduced Active Connections
- Reduction of CPU usage
- Reduction of network traffic on the server

INCREASED CLIENT SIDE PERFORMANCE

The largest bottleneck on Web Applications tends to be the time taken to download static content from the web server, not the time taken to serve the html page. In figure 10.12 we can see the number of requests made to www.manning.com, and the length of time it has taken for the content to be served.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 10.12 - Requests to www.mannning.com

If we look at Figure 10.12, we can see that it only takes 1.72 seconds to serve up the html page but a further 3.5 seconds to serve that the static content. This is because most internet browsers (including Firefox and Internet Explorer) can only download 2 files simultaneously from the same domain. If more than 2 files have been requested then all other requests from the same domain will be queued. In the graph in Figure 10.12 we can see that the .gif files from manning.com are only being downloaded 2 files at a time; however the files being served from google-analytics.com can be downloaded at the same time as the manning.com gifs.

A CDN means that static content will be moved to another sub-domains allowing the browser to simultaneously download more files and therefore reducing the time taken to render content on the client browser.

With the reduced network latency, faster internet connections and the increased number of distributed web servers, any static content served from the CDN will be delivered quicker than a standard web server.

Using CDN's means your Web Server does less work

Any request that you can push off to the CDN (Images, Movies, CSS, Javascript) means the less files that the Web Servers hosting your Web Application needs to server.

Ultimately this means to meet the demands of your web traffic you will need less Web Role Instances meaning you will save money.

CDN's are good, CDN's save money, CDN's save the planet, CDN's save your soul

OK, we've got the message, how do we use CDN's with Windows Azure.

10.4.3 Blob Storage and CDN's

In this section we will take a brief look at how we can use CDN's with Windows Azure.

Other CDN's do exist

Although we are showing you how to use the Windows Azure CDN (c'mon it is a Windows Azure book, give us a break), you can use other CDN providers. Akamai and Amazon Cloud Front are examples of other CDN providers.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

As you are about to discover using the Windows Azure CDN is probably the easiest option (no idea if it's the cheapest, you'll need to use our JavaScript calculator to work that one out).

ORIGIN SERVERS

Although Web Roles can be used as Origin Servers with non-Windows Azure CDN's, this is a more expensive option than using Blob Storage. Since Edge servers can only serve static content therefore there is no need for the capability of generating dynamic server side content. An efficient cost-effective solution would be to utilize your storage account as an origin server rather than using a Web Role.

WINDOWS AZURE CDN ONLY ALLOWS YOU TO USE BLOB STORAGE AS AN ORIGIN SERVER

In our movie player example, the original version of our Big Buck Bunny Movie, or even the original pre-split chunk files would all be stored in Blob Storage, and Blob Storage would act as the origin server.

EDGE SERVERS

To enable the Windows Azure CDN, you just need to open the storage account in the Windows Azure Developer Portal, and click on the enable CDN button as shown in Figure 10.13

Figure 10.13 - Enabling a CDN via the Windows Azure Developer Portal

Once you have clicked on the enable CDN button, any blobs stored in a public container will be available on the CDN after about an hour. Figure 10.14 shows the Developer Portal after your CDN account has been enabled.

Figure 10.14 - An enabled CDN account in the Windows Azure Developer Portal

As you can see from Figure 10.14, a new custom domain is assigned to your CDN enabled blobs. In Figure 10.14 my CDN domain is (http://az1903.vo.msecdn.net/). All I need to do now is replace my previous blob URI's with this domain rather than http://chrishayuk.blob.core.windows.net/

Finally, if you don't like the assigned sub-domain, you can assign your own custom domain to the CDN domain, for details on how to do that, see chapter 8.

PERFORMANCE TIP

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

If you want to improve the amount of parallel downloads on your site, assign half your assets to the assigned CDN domain, and the other half to a custom domain name ;)

10.6 Summary

Over the course of the last few chapters, we have explored Blob Storage pretty thoroughly and you should now be able to use this storage service effectively in your applications.

In this chapter we have discovered that Blob Storage does need to be just a hard disk for your Web and Worker roles. Blob Storage is a very powerful storage mechanism that can be used to:

- Host static websites
- Host Rich Internet Applications (RIA) such as Silverlight
- Act as a Media Server
- Host assets for your existing websites (azure or non-azure hosted)
- Act as a content delivery network

We're now going to move away from Blob Storage and build upon our knowledge gained with this Storage Service and look at another part of the Storage Services puzzle, namely the Table Service.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

11

The Table Service, a whole different entity

This chapter covers:

- What the Table Service Is
- How to get started developing with the Table Service
- Using the Table Service in the live production environment

As we discovered in the previous chapters if your web applications runs in a web farm (i.e. can be load balanced to run on more than one server) then you need a shared storage service that all instances of your application can read and write to.

In the previous 3 chapters we focused on how to retrieve and storage binary data using the blob storage service. In the next few chapters we will look at how we can store table based data using the Table Service.

In typical web applications we would normally store our data in a relational database such as SQL Server. Although SQL Server is great at representing relational data and is a suitable data store for many situations, it is very difficult to design very scalable databases at low cost. Even SQL Azure is not really a scalable database (although this is likely to change in the future) and if you decide to use SQL Azure as your data store then you will need to design and implement the scalability yourself (it's not provided out of the box).

Problems of Scale in Relational Databases

In sites with large number of users as the web server load increases then you may need to scale out the number of servers to cope with the increase in demand. What do you if you need increase the capacity of your data store? Unfortunately most databases are designed not to scale beyond a single server which means the only way to cope with the increase in demand is to either scale up your hardware or to redesign your application.

Although it is possible to design scalable federated databases with SQL server the licensing costs, the design complexity, the cost of development and the operational costs of running such a system makes it very difficult to justify.

To get around the problems of scalability, Windows Azure provides its own table based storage mechanism called the Table Service (catchy name isn't it, they couldn't call it PurpleTableLight or something, could they?).

11.1 What is the Table Service?

The Table Service component of the Windows Azure Storage Services (Blobs, Table and Queues) is a very simple, highly scalable, cost-effective solution that can be used in many scenarios to replace traditional SQL Server based designs.

Built upon Windows Azure Data Center Infrastructure

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Like all other storage services the Table Service is hosted within the Windows Azure Data Centers leveraging the Web Role infrastructure. Access to the service is provided through an HTTP based REST API. For more details on the infrastructure of the service see the Blobs Chapters.

The Table Service provides us the ability to create very simple tables that we can use to store serialized versions of our entities in. Figure 11.1 shows how entities are stored in the Table Service.

Figure 11.1 - In this example our Hawaiian Shirts (which are represented as entities) are stored in the products table. Tables are very similar to Blob Containers (but instead hold entities rather than blobs). Like Blob Containers, Tables are housed within your

storage account. In the diagram the products and shoppingcart tables live in the silverlighukstorage account,.

In figure 11.1 we can see that we have created 2 tables in our storage account (products and shopping cart). The products table could represent the product list for the Hawaiian Shirt Shop Website that we introduced in chapter 2, and each entity stored in the products table (Red Shirt, Blue Shirt and Blue Frilly Shirt) would represent each different type of shirt that we sell.

NOT A RELATIONAL DATABASE

It is important to point out at this stage although the Table Service offers the ability to store data in tables it is not a relational database.

Since it is not a relational database it is does not offer the sort of functionality that you may be used to, such as the ability to:

- create foreign key relationships between tables
- perform server side joins between tables
- create custom indexes on the tables

If you do require relational database storage then you can look at SQL Azure which is a Windows Azure Platform hosted SQL Server database.

Surely I always need a relational database, you pillock?

Over the past decade or so, there has been a government conspiracy to make us always use relational databases to track our data ;). Every database vendor has some secret code in their product that means the government can query our data whenever they want ☺

Okay, so the above statement is a bunch of baloney but there is a serious point there (no really there is). We are a little conditioned to store our data in a relational form, even when it's not strictly necessary.

If you can expand your mind and except that there are other ways of storing data then you can use the Table Service to store your data in a highly scalable (and cheaper fashion).

As we will see later in this book, many applications (including Shopping Carts, Blogs, Content Management Systems etc) could potentially use the Simple Table Service rather than a relational database.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Now that we understand the Table Service is not a relational database but an entity storage mechanism. We now really want to look at how entities are stored in the Table Service. Before we do that, let's take a look at how we normally represent our data in non-Windows Azure environments.

11.2 How we normally represent an entity outside of Azure?

To keep things pretty simple we will return to the Hawaiian Shirt Website that we introduced in Chapter 2. Over the next few sections we will look at how we typically store our shirts in a database for Non-Cloud Solutions by looking at the following:

- How would we represent a shirt in C#?
- How would we store shirt data in SQL Server?
- How do we map and transfer data between the two platforms?

By understanding how we represent our shirt data in typical solutions, we can then see how this translates to the world of the Table Service.

11.2.1 How we would normally represent an entity in C#

In chapter 2 we defined a Hawaiian Shirt using the following Data Transfer Object (DTO) entity.

```
public class Product
{
   public int Id { get; set; }
   public string Name { get; set; }
   public string Description { get; set; }
}
```

The product entity class that we are using to represent our Hawaiian Shirt contains 3 properties to that we are interested in (Product Id, Name and Description).

In chapter 2 we chose to keep our sample simple by hard-coding our list of products rather than retrieving it from a data store such as Table Service or SQL Server. The following code is a hardcoded list of the 3 shirt entities displayed in figure 11.1 (Red Shirt, Blue Shirt and Blue Frilly Shirt).

```
var products =
   new List<Product>
   {
     new Product
     {
        Id = 1,
        Name = "Red Shirt",
        Description = "Red"
     },
     new Product
     {
        Id = 2
        Name = "Blue Shirt",
        Description = "A Blue Shirt"
     },
     new Product
     {
        Id = 3
        Name = "Blue Frilly Shirt",
        Description = "A Frilly Blue Shirt"
     },

};
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In the code above, we simply defined our list of products as a hardcoded list. Obviously this is not a very scalable pattern, we really don't want to redeploy our application every time our shop offers a new product, so let's look at how we can store that data using a non-Windows Azure environment such as SQL Server.

11.2.2 How we would normally store an entity in SQL Server

To be able to store our entity in SQL Server we first need to define a table where we can store our entity data. Figure 11.2 shows how the products table could be structured in SQL Server.

Figure 11.2 - A typical representation of how you would store our Hawaiian shirts data in SQL Server.

In figure 11.2 we can see that we have table called "Products" with 3 columns (ProductId, ProductName and Description). In this table ProductId would be our primary key and would uniquely identify our shirt in the table. Table 11.1 shows how our shirt data would be represented in SQL Server.

ProductId	ProductName	Description
1	Red Shirt	A Red Shirt
2	Blue Shirt	A Blue Shirt
3	Blue Frilly Shirt	A Frilly Blue Shirt

Table 11.1 - Logical representation of our products table in SQL Server

In Table 11.1 you can see that we have enforced a fixed schema in our SQL Server representation of our Hawaiian Shirt. If we wanted to store extra information about the product (e.g. Thumbnail URI) then we would need to add an extra column to the products table and a new property to the Product entity.

Now that we can represent our Hawaiian shirt product as both an entity and as a table in SQL Server we will need to map the entity to the table

11.2.3 Mapping our Entity to our SQL Server Database

Although you can manually map entities to SQL Server data you would typically use a data access layer framework that provides mapping capabilities. Typical frameworks include:

- ADO.NET Entity Framework
- Linq to SQL
- NHibernate

The following code maps our products table returned from SQL Server as dataset to our product entity class using LINQ to DataSet.

```
var products = ds.Tables["Products"].AsEnumerable().Select
   (
      row => new Product
      {
        Id = row.Field<int>("ProductId"),
        Name = row.Field<string>("ProductName"),
        Description = row.Field<string>("Description")
      }
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
    );
```

In this example we convert our dataset to an enumerable list of data rows and then reshape the data to return a list of product entities. For each property in the product entity (Id, Name and Description) we map the corresponding columns (ProductId, ProductName and Description) from the returned data row.

At this stage we have seen how we would normally define our entities in C#, represent our entities in SQL Server and map the entity layer to the database. We will now look at how this differs when using the Table Storage Service.

11.3 Modify our entity to work with Table Service

Before we look at how we can start coding against the Table Service. We need to gain an understanding of how our data is stored in the Table Service (and how that differs from the SQL based solution that we showed earlier). In the next couple of sections, we will look at the following items:

- How we modify our entity so it can be store in the Table Service?
- How our entity is stored in the Table Service?

As the list above suggests, before we can store our shirt data in Table Storage we need to do a little bit of jiggery pokery with our entity definition first. Let's look at what we need to do.

11.3.1 Modifying our entity definition to work with the table service

To be able to store our C# entity in the Table Service each entity must have the following properties:

- Timestamp
- PartitionKey
- RowKey

Therefore, to store our product entity in Windows Azure table storage we would have to modify the previous definition of our product entity to look something like this:

```
[DataServiceKey("PartitionKey", "RowKey")]
public class Product
{
   public string Timestamp{ get; set; }
   public string PartitionKey { get; set; }
   public string RowKey { get; set; }
   public string Name { get; set; }
   public string Description { get; set; }
}
```

In the above code we have modified the original Product entity to include those properties required for Table Storage (Timestamp, PartitionKey and RowKey) (don't worry we will explain what these properties mean slightly later on).

To generate a hardcoded list of shirts using the new version of product entity, we would need to change the code to something like this:

```
var products =
   new List<Product>
   {
     new Product
     {
        PartitionKey = "Shirts",
        RowKey= "1",
        Name = "Red Shirt",
        Description = "Red"
     },
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
        new Product
        {
           PartitionKey = "Shirts",
           RowKey = "2",
           Name = "Blue Shirt",
           Description = "A Blue Shirt"
        },
        new Product
        {
           PartitionKey = "Shirts",
           RowKey = "3",
           Name = "Frilly Blue Shirt",
           Description = "A Frilly Blue Shirt"
        },

   };
```

As you can see from the above code, nothing has really changed from before except, we are now setting a couple of extra properties (i.e. PartitionKey and RowKey)

Look no Timestamp

Notice in the code above that we do not set the Timestamp property. The reason we do not set this property is that it is generated server-side and is only available to us as a read-only property.

The Timestamp property holds the date and time that the entity was inserted in the table, if we did set this property then the Table Storage service would just ignore the value.

The Timestamp property is typically used to handle concurrency. Prior to updating an entity in the table you could check that the timestamp for the local version of the entity is the same as the server version. If the timestamps have changed then you would know that another process has modified the data since you last retrieved your local version of the entity.

Now that we've seen how to modify our entities, so that we can store them in the Table Service, let's take a look at how this would be stored in a Table Service table.

11.3.2 Table Service Representation of our Products

In Table 11.1 we saw how we would normally store our list of Hawaiian shirt product entities in SQL Server and in Table 11.2 we can see how those same entities would logically be stored in the Windows Azure Table Service.

Timestamp	PartitionKey	RowKey	PropertyBag
2009-07-01T16:20:32	Shirts	1	**Name:** Red Shirt
			Description: A Red Shirt
2009-07-01T16:20:33	Shirts	2	**Name:** Blue Shirt
			Description: A Blue Shirt
2009-07-01T16:20:33	Shirts	3	**Name:** Frilly Blue Shirt
			Description: A Frilly Blue Shirt

Table 11.2 - Logical representation of our products table in Windows Azure

As you can see from Table 11.2, entities are represented in the Table Service in a way that is quite a bit different from how it would be stored in SQL Server.

Minimal Schema

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In the SQL Server version of our products table we maintained a fixed schema where each property of our entity was represented by a column in the table. In Table 11.2 you can see that the table service maintains does not rigidly fix the schema and maintains a much more minimal schema. The only properties that the table service requires and therefore logically represented by their own columns are Timestamp, PartitionKey and RowKey; all other properties are lumped together in a property bag.

EXTENDING OUR ENTITY DEFINITON

Since all tables that are created in the table storage service have the same minimal fixed schema (Timestamp, PartitionKey, RowKey and PropertyBag) we do not need to define our entity structure to the table service in advance.

This flexibility means that we can also change our entity class definition at any time. If we wanted to show a picture of a Hawaiian Shirt on our website we may change our Product Entity to include a Thumbnail Uri property as shown below.

```
[DataServiceKey("PartitionKey", "RowKey")]
public class Product
{
   public string Timestamp{ get; set; }
   public string PartitionKey { get; set; }
   public string RowKey { get; set; }
   public string Name { get; set; }
   public string Description { get; set; }
   public string ThumbnailUri { get; set; }
}
```

Now that we have modified our entity to include a thumbnail URI we can now store that entity directly in our existing entities table without either modifying the table structure or the existing data. Table 11.3 shows a list of shirts that include our new property.

Timestamp	PartitionKey	RowKey	PropertyBag
2009-07-01T16:20:32	Shirts	1	**Name:** Red Shirt
			Description: A Red Shirt
2009-07-01T16:20:33	Shirts	2	**Name:** Blue Shirt
			Description: A Blue Shirt
2009-07-01T16:20:33	Shirts	3	**Name:** Frilly Blue Shirt
			Description: A Frilly Blue Shirt
2009-07-05T10:30:21	Shirts	4	**Name:** Frilly Pink Shirt
			Description: A Frilly Pink Shirt
			ThumbnailUri: frillypinkshirt.png

Table 11.3 - Although we've modified our entity definition to include a new property, our new entity can happily co-exist with older entities without the new property.

In our list of shirts stored in Table 11.3 we can see that existing shirts (Red Shirt, Blue Shirt and Frilly Blue Shirt) have the same data that was stored in table 11.2 (i.e. does not contain the new ThumbnailUri property) and the data for our new shirt (Frilly Pink Shirt) does have the new ThumbnailUri property.

11.3.3 Storing Completely Different Entities

Due to the flexible nature of the table service we could even store entities of completely different types in the same table. For example we could even store our product entity in the same table as a completely different entity such as the customer entity below:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
[DataServiceKey("PartitionKey", "RowKey")]
public class Customer
{
   public string Timestamp{ get; set; }
   public string PartitionKey { get; set; }
   public string RowKey { get; set; }
   public string Firstname { get; set; }
   public string Surname { get; set; }
}
```

As we can see from the Customer entity that although the entity must contain the standard properties (Timestamp, PartitionKey and RowKey) no other properties are shared between the Customer and the Product entities (they even have different class names). Even though these entities have very different definitions they could be stored in the live service as shown in table 11.4

Timestamp	PartitionKey	RowKey	PropertyBag
2009-07-01T16:20:32	Shirts	1	**Name:** Red Shirt
			Description: A Red Shirt
2009-07-01T16:20:33	Shirts	2	**Name:** Blue Shirt
			Description: A Blue Shirt
2009-07-01T16:20:33	Shirts	FredJones	**Firstname:** Fred
			Surname: Jones
2009-07-05T10:30:21	Shirts	4	**Name:** Frilly Pink Shirt
			Description: A Frilly Pink Shirt
			ThumbnailUri: frillypinkshirt.png

Table 11.4 - Storing completely different entities in the same table.

Challenges of Storing different Entity Types

Although the table service is flexible enough to store entities of different types in the same table as shown in Table 11.4, you should be very careful if you are considering such an approach. If every entity that you retrieve has a different schema then you will need to write some custom code that will serialize the data to the correct object type. Following this approach will lead to more complex code that will be difficult to maintain. This code is likely to be more error prone and complex to debug so we encourage you to only store entities of different types when absolutely necessary.

Challenges of Extending Entities

On a similar note if you need to modify the definition of existing entities then you should be very careful that your existing entities do not break your application after the upgrade.

There are a few rules that you should consider that should prevent you from running into too much trouble.

- Treat entity definitions as data contracts, breaking the contract will have a serious effect on your application so do not treat it lightly.
- Code new properties as additional rather than required. This strategy means any existing data will be able to serialize to the new data structure. If your code does require existing entities to contain data for the new property then you should migrate your existing data to the new structure.
- Continue to support existing property names for existing data. If you do need to change a property name you should either support both the old and new name in your new entity or support two versions of your entity (old definition and new definition). If you only want to support one entity definition then you will need to migrate any existing data to the new structure.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Now that we understand how entities are stored within the Table Service, let's look how what makes this scalable.

11.4 Partitioning data across lots of servers (i.e. the scalable bit)

In the last couple of sections, we've sort of happily skipped past a few topics namely (Accounts, Partition Keys and Row Keys). We'll now return to these topics ans explain how the Windows Azure Table Service is such a scalable storage mechanism.

In this section we will look at how the Table Service scales using partitioning at the following levels:

- Partitioning the Storage Account
- Partitioning the Tables

To achieve a highly scalable service, the Table Storage service will split your data into more manageable partitions that can be apportioned out to multiple servers. As a developer we can control how this data is partitioned to maximize the performance of our application. Let's look at how this done first as the storage account layer.

11.4.1 Partitioning the Storage Account

In this section we will look at how data is partitioned but we will leave performance optimization to a later section.

TABLES

In figure 11.1 we had two tables within our storage account (ShoppingCart and Products). As the table service is not a relational database there is no facility to join these 2 tables server side. Since there is no physical dependency between any two tables in the table service, Windows Azure can scale data storage beyond a single server and store tables on separate physical servers. Figure 11.3 shows how these tables could be split across the Windows Azure data center.

Figure 11.3 Tables within a storage account split across multiple servers

If we look at how the servers are split in Figure 11.3 you will notice that the products table lives on servers 1, 2 and 4, whilst the shopping cart table resides on servers 1, 3 and 4. In the Windows Azure data center we have no control on where our tables will be stored. The tables could reside on the same server (as in the case of

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

server 1 and 4) but they could easily live on completely separate servers (servers 2 and 3). In most situations we can assume that each table will physically reside on different servers from each other.

Data Replication

In order to protect you from data loss, Windows Azure guarantees to replicate your data to at least 3 different servers as part of the transaction. This data replication guarantee means that if there is a hardware failure after the data has been committed then another server will have a copy of your data.

Not only is your data replicated within the same data center as your storage account. In the PDC 2009 release of Windows Azure your data center will be replicated across multiple data centers. Just in-case a big scary monster destroys the data center.

Guaranteed Read Post Transaction Commit

Once the transaction is committed (and your data has been therefore replicated at least 3 times), the Table Service is guaranteed to serve the new data and will never serve older versions. This guarantee means that if you insert a new Hawaiian shirt entity on server 1, then you can only be load balanced onto one of the servers that has the latest version of your data. If server 2 was not part of the replication process and contains stale data you will not be load balanced onto that server. Therefore you can safely perform a read of your data straight after a write knowing that you will receive the latest copy of the data.

The Amazon SimpleDB database (which has roughly the same architecture as the Windows Azure Table Service) does not have this replication guarantee. Due to replication latency it is not uncommon in SimpleDB for newly written data not to exist or to be stale when a read is performed straight after a write. This situation can never occur with the Windows Azure Table Service.

Now have seen how that different tables within a single account will be spread across multiple servers to achieve scalability, it's worth looking how we can partition data a little more granularly and split data within a single table across multiple servers.

11.4.2 Partitioning Tables

One of the major issues with traditional SQL server based databases is that individual tables can grow too large, slowing down all operations against the table. Although the Windows Azure Table Service is highly efficient if you store too much data in a single table then your data access performance will degrade also.

The Table Service allows you to specify how your table could be split into smaller partitions by requiring each entity to contain a partition key. The table service can then scale out by storing each partition of data onto separate physical servers. Any entities with the same partition key must reside together on the same physical server. In table 11.2 we stored all our data in the same partition (shirts) meaning that all 3 shirts would always have to reside together on the same server as shown in figure 11.3. Table 11.4 shows how we could split our data into multiple partitions.

Timestamp	PartitionKey	RowKey	PropertyBag
2009-07-01T16:20:32	Red	1	**Name:** Red Shirt
			Description: A Red Shirt
2009-07-01T16:20:33	Blue	1	**Name:** Blue Shirt
			Description: A Blue Shirt
2009-07-01T16:20:33	Blue	2	**Name:** Frilly Blue Shirt
			Description: A Frilly Blue Shirt
2009-07-05T10:30:21	Red	2	**Name:** Frilly Pink Shirt

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Description: A Frilly Pink Shirt

ThumbnailUri: frillypinkshirt.png

Table 11.4 - Splitting Partitions by Partition Key

In Table 11.4 the Red Shirt and the Frilly Pink Shirt now reside in the Red partition whilst the Blue Shirt and the Frilly Blue shirt are now stored in the Blue partition. Figure 11.4 shows our shirt data from table 11.4 split across multiple servers.

Figure 11.4 - Splitting Partitions across multiple servers

In figure 11.4 the Red Partition data (Red Shirt and Pink Frilly Shirt) lives on Server A whilst the Blue Partition data (Blue Shirt and Frilly Blue Shirt) is stored on server B. Although the partitions have been separated out to different physical servers all entities within the same partition always reside together on the same physical server.

ROW KEY

The final property to explain is the Row Key. The Row Key uniquely identifies an entity within a partition meaning that no 2 entities in the same partition can have the same row key however any 2 entities that are stored in different partitions can share the same key. If we look at the data stored in Table 11.4 we can see that Row Key is unique within each partition but not unique outside of the partition. For example Red Shirt and Blue shirt both have the same Row Key but live in different partitions (Red and Blue).

The partition key and the row key combined make up the composite primary key of the table i.e. the partition key and row key combined uniquely to identify an entity.

INDEXES

Now that we have a basic understanding of how data is logically stored within the data service it is worth talking briefly about the indexing of the data.

There are a few simple golden rules to remember that we will explore in more detail in future chapter.

- Retrieving an entity with a unique partition key is super-fast
- Retrieving an entity using the Partition Key and Row Key is very fast (all index)
- Retrieving an entity using the partition key and no row key is slower (need to read all properties for each entity in the partition)
- Retrieving an entity using no partition key and no row key is dog slow (need to read all properties for all entities across all partitions which can span separate physical servers)

Load Balancing of Requests

Since data is partition and replicated across multiple servers all requests via the REST API can be load balanced. This combination of data replication, data partitioning and a large web server farm provides you with a highly scalable storage solution that can evenly distribute data and requests across the data center. This level of horsepower and data distribution means that you should not need to worry about overloading server resources.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Now that we understand the theory of table storage, it's time to put it in practice, crank open Visual Studio, and start storing some data.

11.5 Developing with the Table Service

Now that we have an understanding of how data is stored in the Table Service it is time to get started developing a web application that can use it. In the previous section, we defined an entity for storing our Hawaiian Shirt Product and we showed you how it would be stored in the Table Service.

Before we get started developing our management web page, we need to get comfortable with using the Table Service in our development environment.

11.5.1 Creating the project

Rather than returning to the solution that we built in Chapter 2 we will develop our new product management web page in a new web application project in a fresh solution. You should now create a new cloud service web role project called ShirtManagement, if you need a refresher on how to setup your development environment or how create a web role project then you should refer back to chapter 2.

Like the other storage services communication with the Table Service occurs through the REST API (more details in the next chapter). Although you can use this API directly you are likely to be more productive using the Storage Client Library provided in the Windows Azure SDK.

Whenever you create a new Cloud Service Project this library will be automatically referenced. If however you are building a brand new class library or migrating an existing project then you can always reference the following storage client assembly manually:

- Microsoft.WindowsAzure.StorageClient

As well as referencing the Storage Client Library, you will also need to reference the ADO.NET Data Services assemblies

- System.Data.Services
- System.Data.Services.Client

ADO.NET Data Services

The table service exposes its HTTP REST API's through its implementation of the ADO.NET Data Services framework. The use of this framework means we can utilize the ADO.NET Data Services Client assemblies to communicate with the service rather than having to develop or use a custom REST wrapper.

Since ADO.NET Data Services is used by the Storage Client SDK then you will need to reference those assemblies too.

Now that we have setup our project, let's look at how we can add our product entity class to the project.

11.5.2 Defining the entity

Before we create our management web page as shown in Figure 11.5 we need to create our entity in the web project. At this stage we will just add our entity directly to our web page (however in the next chapter we will show you how you can architecturally separate your class into an entity layer).

To keep our example simple we will just store the Shirt Name and Description as before. You should now add a new class to your web project named Product.cs and define the class as shown in Listing 11.1

Listing 11.1 Product Entity

```
public class Product : TableServiceEntity                          #1
{
   public string Name { get; set; }
   public string Description { get; set; }
}
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In Listing 11.1 at #1 the product class inherits from the TableServiceEntity base class (Microsoft.WindowsAzure.TableService.TableServiceEntity). This base class contains the 3 properties required by all table storage entities:

- Timestamp
- PartitionKey
- RowKey

Do not inherit from TableServiceEntity in a world of purity

Although the TableServiceEntity base class makes your entities look a little simpler, in the real world this means you have just created a dependency between your entity layer and the storage client library.

To keep your architectural layers clean, you should either create your own TableServiceEntity base class in your entity layer, or include all the required properties in your entity definition as shown earlier in this chapter.

Now that we have setup our project and defined our entity, we need to be able to create a table to store our entity in. The same method can be used in both the development and live environment.

11.5.3 Creating the Table

The simplest method of creating a table is to either use to create a PowerShell script or to use one of the many storage clients that are available (my favorite being Azure Storage Explorer)

YOU CAN DOWNLOAD AZURE STORAGE EXPLORER FROM CODEPLEX

http://azurestorageexplorer.codeplex.com/

In this section we will show you how to:

- Create a Table via Azure Storage Explorer
- Create a Table in Code

CREATING A TABLE USING AZURE STORAGE EXPLORER

Once you have downloaded and fired up Azure Storage Explorer, the tool will automatically connect you to your development storage account.

To create a new table in your storage account all you need to is to click on Table->New Table and enter the name of your table (e.g. Product). Figure 11.5 shows our newly created Product table in Azure Storage Explorer.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 11.5 - Azure Storage Explorer showing our newly created Product table

Although using a tool (such as Azure Storage Explorer) is probably the easiest method of creating a new table, you may wish to do this manually in C#.

CREATING A TABLE IN CODE

In this example we will manually create a console application that when run will create the new table in our storage account. Although we have used a console application in this case, we could easily use Web Application, Windows Forms Application or WPF Application. The deployment application does not need to be a cloud application (web or worker role) and can just be a standard application that you can run locally.

To create the test application, you perform the following steps:

1. Create the Console Application
2. Add a reference to System.Data.Services
3. Add a reference to System.Data.Services.Client
4. Add a reference to Microsoft.WindowsAzure.StorageClient
5. Add an app.config / web.config with your live / development storage account

   appSettings for your storage account and endpoint (see blobs chapter)
6. Add the following code to create the table (replacing the table name with your table name)

```
var storageAccount =
    CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

CloudTableClient tableClient =
    storageAccount.CreateCloudTableClient();

tableClient.CreateTableIfNotExist("Product");
```

In the code above we retrieve our storage account information from the app.config as before and then call the CreateTableIfNotExist method from our Cloud Table Client object passing in the name of the table to create (Product).

> **Deploying to Live**
>
> The code used to create a new table will not only work on your development storage account but will also work against the live system.
>
> All you need to do to make this code work against the Live System is change your DataConnectionString appSetting to your live account (see the Blobs Chapter for more details on how to do this).

Finally, now that we know how to create our table both in the Live system and the Development Storage it's worth taking a quick peek at how this is implemented in the Development Storage backing store. Figure 11.6 shows how tables are represented in the Development Storage SQL Server database.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 11.6 - How tables are represented in the Development Storage SQL Server Db

As we can see from Figure 11.6, the SQL Server Database that stores our entities is pretty flexible. The TableContainer table keeps a list of all the tables stored in our development storage account. Since we can create tables dynamically, any new table created will contain a new entry in this table.

If we were to look at the TableRow table shown in Figure 11.6, each row in this table stores a serialized version of our entity. As you can see from this table definition the only fixed data that is stored in this table are: AccountName, TableName, PartitionKey, RowKey and TimeStamp. All other properties are stored in the Data column. As you can see the actual development storage schema relates to the logical representation that we showed in Table 11.4.

Now that we have defined our entity, created the table that our entity will be stored in, and looked at how this represented in the Development Storage. Let's now look at how we can start working with our entities.

11.6 Doing Cruddy Stuff with the Table Service

In this section, we will build a new product management web page that will allow us to manage the Hawaiian shirt product list stored in the Table Service for our shirt shop. Figure 11.7 shows what the management web page will look like.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 11.7 Product List Management Web Page

The management web page shown in Figure 11.5 will allow us to:

- Add New Shirts
- List all shirts
- Edit Existing Shirts
- Delete Shirts

Now that we have setup our entity and registered the entity table in the development storage we can start to develop the product management web shown in figure 11.7.
Now that we have created our product entity class, we must also create a context class that we will use to access our data service.

11.6.1 Creating a context class

In order to able to work with entities (query, add, insert, delete) using ADO.NET Data Services, we first need to setup a context object as shown in Listing 11.2

Listing 11.2 Product Context Class

```
public class ProductContext : TableServiceContext                        #A
{
   private static CloudStorageAccount storageAccount =                   #B
      CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

   public ProductContext()
      : base(storageAccount.BlobEndpoint.ToString(),
               storageAccount.Credentials)                               #C
   {

   }

   public DataServiceQuery<Product> Product                              #D
   {
      get
      {
         return CreateQuery<Product>("Product");
      }
   }
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
}
```

#A - TableServiceContext is an Azure version of DataServiceContext
#B - Read in our account details from config
#C - Use our account details to populate the context

Listing 11.2 shows the context object for our Hawaiian Shirt Product entity. As we can see most of the complexity of the context class is abstracted away in the classes that we inherit from. The TableServiceContext class in-fact inherits from the standard ADO.NET Data Services context class DataServicesContext. The TableServiceContext class provides some additional functionality beyond what is provided out of the box with ADO.NET Data Services including Retry Policies etc.

In Listing 11.2 we automatically populate our storage account details, and credentials from configuration. This allows us to simplify our calling classes meaning that we won't need to get the endpoint and credentials everytime we wish to use the context class.

Finally, as you can see the Product property is what we will use to perform LINQ queries on our Product table. Let's now look at how we can start using this.

Code Generation

Although code generation is outside the scope of this book if you are generating a large number of tables and entities you may wish to consider a code generation tool such as T4 to auto generate code where possible. Typical areas to consider generating code automatically would include:

Table Context Classes

Table Creation Scripts

11.6.2 Adding entities

Now that we have our context class, we should get on creating our management web page. To do this we show add a new asp.net web page called products.aspx. At this stage we will not generate the grid listing all the Hawaiian Shirts for sale and we will only write the code required to add shirts to the product table. Therefore we will only add the markup for the bottom section of the products page. Listing 11.3 contains the ASPX code that you should add to the products.aspx page.

Listing 11.3 Add shirt section of the products.aspx page

```
<fieldset>
   <div>
      <div>id: <asp:TextBox id="txtId" runat="server" /></div>      #A
      <div>name: <asp:TextBox id="txtName" runat="server" /></div>  #B
      <div>
          description:
          <asp:TextBox id="txtDescription" runat="server" />        #C
      </div>
      <asp:button id="btnAdd" text="Add"
                  runat="server" onclick="btnAdd_Click" />          #D
   </div>
</fieldset>
```

#A - Id Text Box (used to store the Product Id (or RowKey))
#B - Product Name Text Box
#C - Product Description TextBox
#D - Add Button, on click inserts the shirt into the products table

On click of the add button at #D in Listing 11.3 the shirt entity will be added to the product table. The code in listing 11.4 for the click event of the add button can be added to the products.aspx code-behind.

Listing 11.4 Add the entity to the product table

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
protected void btnAdd_Click(object sender, EventArgs e)
{
   var shirtContext = new ProductContext();                          #1

   var newShirt = new Product                                        #2
       {
          PartitionKey = "Shirts",
          RowKey = txtId.Text,
          Name = txtName.Text,
          Description = txtDescription.Text
       };

   shirtContext.AddObject("Product", newShirt);                      #3

   shirtContext.SaveChanges();                                       #4

   BindGrid();                                                       #5
}
```

To add the new shirt details entered on the web page to the product table we will extract all the information entered about the product (id, name and description) and create a new instance of our product class called newShirt at #2. Once we have created an instance of the shirt at #3 we add the shirt entity to a tracking list held in the context object for our product list. When we add the entity to the tracking list at #3 we also specify the table that the entity should be added to. The product context object (shirtContext) maintains a list of all objects that we have changed as part of this operation. We can create, update or delete objects from our product list and we can add all these changes locally to the tracking list. Eventually when we wish to perform all the changes server side we can then invoke the SaveChanges method on our context object which will apply all the tracked changes server side using ADO.NET Data Services via the REST API. In the next chapter we will look at some of the more advanced scenarios when applying changes such as batching of changes, concurrency, retry logic and transactions.

DID OUR ENTITY GET ADDED?

You should now be able to run the products web page and add new shirts to the product list. At this stage we have not implemented the grid to display the list of entities in the product list. In the next section we will implement this logic to see our newly created entities. In the mean time you can always check your entity exists by querying the development storage database using the following SQL in SQL Management Studio:

```
SELECT * FROM TableRow
```

11.6.3 Listing entities

We will now extend our products.aspx web page to display the list of shirts stored in the products table. Listing 11.5 contains the code required for our grid. This code should be placed above the code in Listing 11.3.

Listing 11.5 ASP.NET Grid for displaying our shirts

```
<asp:GridView ID="GridView1"
              OnRowCommand="GridView1_RowCommand"                  #A
              OnRowDeleting="GridView1_RowDeleting"                #B
              OnRowEditing="GridView1_RowEditing"                  #C
              OnRowCancelingEdit="GridView1_RowCancelingEdit"      #D
              OnRowUpdating="GridView1_RowUpdating"                #E
              AutoGenerateColumns="false"                          #F
              AutoGenerateEditButton="true"                        #G
              runat="server">
   <Columns>
      <asp:TemplateField>
         <ItemTemplate>
            <asp:LinkButton ID="btnDelete" runat="server"
                            Text="Delete"
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
                                CommandName="Delete"
                                CommandArgument='<%#Eval("RowKey")%>'/>     #H
            </ItemTemplate>
         </asp:TemplateField>
         <asp:BoundField HeaderText="Id"
                         DataField="RowKey" ReadOnly="true"/>             #I
         <asp:BoundField HeaderText="Name" DataField="Name" />             #J
         <asp:BoundField HeaderText="Description"
                         DataField="Description" />                       #K
      </Columns>
   </asp:GridView>
```

#A - Row Command Event - fired when the delete button is clicked
#B - Row Deleting Event - fired when a row is deleted (is required otherwise the grid throws a wobbly)
#C - Row Editing Event - fired when the edit button is clicked
#D - Row Canceling Edit Event - fired when an edit is cancelled by clicking the cancel button
#E - Row Updating Event - fired after an edit when the data needs updated
#F - Don't autogenerate columns we will specify them
#G - Autogenerate an edit button
#H - Delete button passing in the Product Id (RowKey) as the command argument
#I - Display the Product Id (Row Key) as a read only (non editable) field
#J - Display the Product Name (this is editable)
#K - Display the Product Description (this is also editable)

The grid in Listing 11.5 will display the Product Id, Name and Description of each shirt in our products table. The name and description columns will both be editable but the Product Id will not be. At this stage we have defined the markup required to edit and delete the shirts in our table however we will not write the code-behind for these events just yet. Listing 11.6 contains the code-behind for our products.aspx page that we will require to populate our new grid with the list of shirts.

Listing 11.6 - Populating the List of shirts

```
protected void Page_Load(object sender, EventArgs e)
{
   if (!IsPostBack)                                                        #1
   {
      BindGrid();
   }
}

private void BindGrid()                                                    #2
{
   var shirtContext = new ProductContext();                                #3
   GridView1.DataSource = shirtContext.Product;                            #4
   GridView1.DataBind();                                                   #5
}
```

As we can see the code to bind our GridView to the list of shirts held in our products table is pretty simple. On the first load of our web page at #1 we will call the BindGrid method at #2 to populate the grid with our list of shirts retrieved from the products table.

To retrieve the list of shirts we instantiate our Product context object (shirtContext) at #2 and set the data source of the grid to Product property of the context object.

If we look back to the code we used to define our product context in Listing 11.2, the Product property at #3 returned an IQueryable list of products. By returning an IQueryable list of products from our context object we can define a query using LINQ that will be executed server side when we enumerate our list of object when the grid is data bound at #5. To keep our example simple we will not perform any server side filtering at this stage and will simply just return a list of all shirts in the product table as shown at #4. In the next chapter we will look at how you can build efficient server side queries using LINQ.

You should now be able to run your web page and list all the shirts in the products table.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

11.6.4 Deleting entities

In listing 11.5 at #A and #B we included some event definitions in our markup to handle deletes. We will now implement those event handlers so that when you click the delete button in the grid it will delete the corresponding shirt from the products table held in the table service. Listing 11.7 contains the code-behind that relates to the delete button.

Listing 11.7 Code Behind to Delete Shirts

```
protected void GridView1_RowCommand (object sender,
                                     GridViewCommandEventArgs e)     #A
{
   if (e.CommandName == "Delete")
   {
       DeleteShirt(e.CommandArgument.ToString());                   #B
       BindGrid();
    }
}

private void DeleteShirt(string rowKey)                             #1
{
   var shirtContext = new ProductContext();                         #2

   var entity = (from item in shirtContext.Product
                 where item.PartitionKey=="Shirts" &&
                 item.RowKey==rowKey
                 select item).First();                              #3

   shirtContext.DeleteObject(entity);                               #4

   shirtContext.SaveChanges();                                      #5

   BindGrid();                                                      #6
}

protected void GridView1_RowDeleting(object sender, GridViewDeleteEventArgs e){}
                                #C
```

#A - Called when the Delete Button is clicked
#B - If the command is the Delete Command call DeleteShirt passing the Product Id (Row Key)
#C - The row deleting event, we must define this otherwise the grid throws a wobbly

The code in Listing 11.7 shows how we can delete a shirt from the product table when the delete button is clicked in the grid. If we look at the DeleteShirt method as defined at #1 we can see that the code is very similar to the code in Listing 7.4 that we used to add a shirt. Essentially we instantiate our shirt context at #2, we add the shirt to be deleted to the context object's tracking list at #4 and then finally we submit the changes to the table service at #5. Finally rebinding the grid to display our updated list at #6

FINDING THE SHIRT TO DELETE

There are a couple of differences between adding and deleting a shirt. When adding a shirt to our product list we can simply create a new Product object and add that to the tracking list. If we are deleting an object we first need to get a local copy of the object that we wish to delete (as we have done at #3) and add then add that object to the tracking list (as shown at #4).

For the sake of simplicity we will pass the object to be deleted by re-fetching the object from the table service. In production code we would never re-fetch an object for deletion as this performs an unnecessary call to the the table service, in the next chapter we will look at how to delete an object in a production web scenario. In this example however, we take the ProductId (RowKey) that was passes as the Delete Buttons' command argument and then perform a LINQ query to retrieve that entity from our table service. The following code shows the LINQ query that we used to retrieve the individual shirt:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
from item in shirtContext.ProductTable
where item.PartitionKey=="Shirts" && item.RowKey==rowKey
select item
```

Looking at the LINQ query we can see that we are using the IQueryable ProductTable from the shirtContext object as the datasource. As this query is IQueryable we can then modify the query before it gets sent to the server to restrict the result set to only return those entities that reside in the "Shirts" partition whose RowKey match the passed in Product Id. As this query is manipulated before it's sent to the server all the filtering is performed by the Table Service rather than by our client application.

At this stage you will now be able to add, list and delete shirts in our products table.

11.6.5 Updating entities

Finally we need to update the code-behind to allow us to edit the grid and save any changes back to the products table. Listing 11.8 contains the code we need to edit the grid.

Listing 11.8 - Updating Entities

```
protected void GridView1_RowCommand(object sender,
                                    GridViewCommandEventArgs e)
{
   if (e.CommandName == "Delete")
   {
      DeleteShirt(e.CommandArgument.ToString());
      BindGrid();
   }
   if (e.CommandName == "Edit")
   {
      BindGrid();                                                    #A
   }
}

protected void GridView1_RowEditing(object sender,GridViewEditEventArgs e)
{
   GridView1.EditIndex = e.NewEditIndex;                             #B
   GridView1.DataBind();
}

protected void GridView1_RowCancelingEdit(object sender,
                                     GridViewCancelEditEventArgs e)
{
   GridView1.EditIndex = -1;                                         #C
   GridView1.DataBind();
}

protected void GridView1_RowUpdating(object sender,
                                     GridViewUpdateEventArgs e)
{
   GridViewRow row = GridView1.Rows[e.RowIndex];                     #1
   string id = row.Cells[2].Text;                                    #2

   var shirtContext = new ProductContext();                          #3

   var entity = (from item in shirtContext.ProductTable
                 where item.PartitionKey == "Shirts" && item.RowKey == id
                 select item).First();                               #4

   entity.Name = ((TextBox)(row.Cells[3].Controls[0])).Text;         #5
   entity.Description = ((TextBox)(row.Cells[4].Controls[0])).Text; #6

   shirtContext.UpdateObject(entity);                                #7
   shirtContext.SaveChanges();                                       #8

   GridView1.EditIndex = -1;                                         #9
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
    BindGrid();                                                    #10
}
```

#A On click of the edit button we need to rebind the grid
#B Put the grid into Edit Mode (i.e. show textboxes rather than labels)
#C On cancel take the grid out of edit mode (i.e. show labels rather than textboxes)

After editing the grid with our new values we need to store the modified data back to the products table. In listing 11.8 at #1 we have just entered the GridViews RowUpdating event where will now perform that update. The process of updating data is very similar to deleting an entity as shown earlier.

At #1 we retrieve the row that has been edited and extract the Product Id displayed in the second cell of the row. As before we instantiate our product context at #3 and at #4 we retrieve a copy of the edited entity from the table service using a LINQ query passing the partition key and the row key. At #5 and #6 we replace the current name and description of the shirt with the modified data extracted from the textboxes of the row being edited. Finally we add the modified entity to shirtContext's tracking list via the UpdateObject method at #7. We then commit the changes back to the table service at #9 using the SaveChanges method and finally taking the grid out of edit mode at #9 and rebinding the grid to a fresh copy of the data returned from the table service.

Replication in the above examples

Earlier we mentioned that due to the Windows Azure architecture if we performed at write followed by a read that we would be guaranteed to get the latest data and never be served staled. In both Listing 11.7 and Listing 11.8 we experience that exact scenario. In listing 11.7 at #8 we update the entity held in table storage and at #10 we re-query the table service to return all shirts expecting the newly modified data to be returned. If we were to perform this code against an Amazon S3 database, due to replication latency we would be served stale data. With the Windows Azure Table Storage Service we will always receive the newly modified data.

11.7 Summary

In this chapter we gave an overview of the Table Storage Service, how it provides massively scalable storage and how it differs from traditional relational database such as SQL Server.

Using this knowledge, we returned to our Hawaiian Shirt Shop example and created a web page that we can use to manage our Hawaiian Shirt product list. This example web page allowed us to perform all major operations against the table service (Create, Read, Update and Delete); it also gave us an understanding of the differences between the development and live version of the Table Service.

We now have enough knowledge to create and deploy web applications that run in the cloud and can store data in the Table Service. In the next chapter, we will expand upon our new knowledge and look at some of the more advanced scenarios and patterns of usage.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

12

Working with the Table REST API

This chapter covers:

- Understand the Storage Client and REST API
- Understand how to effectively modify data
- Understand the querying capabilities of the table service

In the previous chapter we looked at how data is stored in the Windows Azure Table Service, and we also got started writing our first web page that used the table service. In this chapter we will delve into the depths of the Storage Client Library and the REST API, exploring the power and capabilities of the service.

In the previous chapter we looked at how we could get started with the table service using the Storage Client Library and the ADO.NET Data Services client. In this chapter we will look at the underlying REST API to gain a better understanding of the communication between our application and the table service.

12.1 Performing operations against the storage account using the REST API

For now we'll concentrate purely on the operations that you can perform against a storage account using the REST API. Although we have already looked at these operations using the storage client library, it is still useful to look at the REST API. Ultimately the storage client library is just a wrapper library for the calls we're about to look at. Over the next few sections look at the following operations that you can perform against the storage account via the REST API:

- Create Table
- Delete Table
- List Tables

END POINTS

In the blobs chapters earlier on in the book, we described how you could interact with the storage services using various endpoints. We won't go over that subject again in this chapter (feel free to read the blobs chapters) but it is worth looking at the endpoint URI of the table service. As you can see the URI of the table service endpoint uses the following structure.

```
http://<storageaccount>.table.core.windows.net/
```

If our storage account was named silverlightuk then our URI would be:

```
http://silverlightukstorage.table.core.windows.net/
```

For the development storage Table Service the following URI would be used

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
http://127.0.0.1:10002/devstoreaccount1/
```

Now that we know what the URI's will look like, let's try and use them now.

12.1.1 Listing tables in the development storage account using the REST API

Earlier in the book we created a small console application that we used to list all the containers in a Blob Storage account using the REST API. We will now create a similar console application that we can use to list all the tables contained in our development storage account. Figure 12.1 shows the result of our console application.

Figure 12.1 Console Application that returns a list of tables in a storage account, gosh what a lot of Xml to return the name of a table.

If you look at the output in Figure 12.1 you can see that the Products table that we created in the previous chapter is returned in the list of storage accounts.

> **Does that funny looking Xml follow some sort of standard?**
>
> As you may have already gathered normal people don't create API's that output Xml like that. You need a standard to generate that level of verboseness and complexity.
>
> The standard that the Table Service uses to expose its data is known as AtomPub. Don't worry we'll discuss this in more detail later.
>
> If you are interested in being able to identify AtomPub documents in the wild you can always look xml namespace. As you can see from Figure 12.1 it's referencing the Atom namespace (as shown below).
>
> `xmlns=http://www.w3.org/2005/Atom`

If we wanted to list all the tables in a storage account using the storage client library, we could do the following:

```
var storageAccount =
      CloudStorageAccount.Parse(
         ConfigurationManager.AppSettings["DataConnectionString"]);

   CloudTableClient tableClient =
      storageAccount.CreateCloudTableClient();
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
   tableClient.ListTables();
```

Let's now take a look at how this could be done using the REST API directly. In order to get a list of all tables that exist in the storage account all we need to do is perform a GET request against the following URI's:

```
   http://127.0.0.1:10002/devstoreaccount1/Tables (for dev storage)
   http://<storageaccount>.table.core.windows.net/Tables (for live)
```

Listing 12.1 contains the code for our console application, which generates the output shown in figure 12.1.

Listing 12.1 Listing the tables in our storage account

```
static void Main(string[] args)
{
   HttpWebRequest hwr =
         CreateHttpRequest(new Uri(@"http://127.0.0.1:10002/devstoreaccount1/Tables"),
                                        "GET", new TimeSpan(0, 0, 30));  #1

var storageAccount =
        CloudStorageAccount.Parse(
           ConfigurationManager.AppSettings["DataConnectionString"]);   #2

   storageAccount.Credentials.SignRequestLite(hwr)                      #3


   using (StreamReader sr = new StreamReader(hwr.GetResponse().GetResponseStream()))
#4
   {
      XDocument myDocument = XDocument.Parse(sr.ReadToEnd());
      Console.Write(myDocument.ToString());
   }
}

private static HttpWebRequest CreateHttpRequest(Uri uri, string httpMethod, TimeSpan
timeout)                                          #5
{
   HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(uri);
   request.Timeout = (int)timeout.TotalMilliseconds;
   request.ReadWriteTimeout = (int)timeout.TotalMilliseconds;
   request.Method = httpMethod;
   request.ContentLength = 0;
   request.ContentType = "application/atom+xml";                        #6
   return request;
}
```

Before you can create the application in Listing 12.1, you will need to create a new Console Application in Visual Studio. To keep the example simple, we are reusing the Storage Client Libraries' credential signing library. You will need to add a reference to this assembly (this is the same project that we used in the previous chapter).

UNDERSTANDING THE CODE

Replace #1-6 in the following paragraphs with a cueball (through the text until listing 12.2)

The code displayed in Listing 12.1 will simply make a GET request to development table service asking for a list of all the tables in the storage account (`http://127.0.0.1:10002/devstoreaccount1/Tables`).

At #1, we generate our http request by calling the CreateHttpRequest method at #5. This method creates the HttpRequest for the given URI (`http://127.0.0.1:10002/devstoreaccount1/Tables`) and http verb (GET) and returns the request back to calling method.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Sign Request Lite

You will probably have noticed that we are making use of the storage account credentials from the storage client library at #2 even though we are using the REST API. The major reason is that signing the http request manually is hard and horrible. Rather than writing that nasty code, it's easier just to use the storage client library method as shown at #3. Later on we will discuss request signing in a little more details however.

Finally, with the request generated and signed, we can make the request to the development service at #4. The table service will return an XML response containing a list of all the tables in our account, which is written to the console window as shown in figure 12.1.

SWITCHING TO THE LIVE SERVICE

In Listing 12.1 we made our request to the development storage table service. If we want to change our application to the live service (for my silverlightukstorage storage account) then we will need to change the URI at #1 to:

```
http://silverlightukstorage.table.core.windows.net/Tables
```

It is worth pointing out that although we have hardcoded the URI in Listing 12.1 at #1, we could extract it from the storageAccount object:

```
storageAccount.TableEndpoint.AbsoluteUri.ToString()
```

As mentioned earlier, there are three operations that you can perform against tables in a storage account (Create, Query and Delete). We've just looked at Query, we will now take a look at how we can change our code to delete a table (and its underlying data).

12.1.2 Deleting Tables using the REST API

To delete a table using the storage client we just need to call the DeleteTable method of our CloudTableClient object passing in the name of the table that we wish to delete. The following code will delete the products table that we created in the previous chapter.

```
var storageAccount =
    CloudStorageAccount.Parse(
       ConfigurationManager.AppSettings["DataConnectionString"]);

CloudTableClient tableClient =
   storageAccount.CreateCloudTableClient();

tableClient.DeleteTable("Products");
```

If we wanted to delete the same table using the REST API directly, we could do this by performing an http DELETE (rather than GET) request to the following URI:

```
http://silverlightukstorage.table.core.windows.net/Tables('Products')
```

To delete the product table using the REST API directly, we could change #1 in Listing 12.1 to the following

```
HttpWebRequest hwr =
      CreateHttpRequest(new
Uri(@"http://silverlightukstorage.table.core.windows.net/Tables('Products')")),
"DELETE", new TimeSpan(0, 0, 30));
```

As we can see from the code above we have changed the original GET request to a DELETE and we have modified the URI to the one specified earlier. Since we no longer need to process an xml response we would also need to change #4 to:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
hwr.GetResponse();
```

Finally, since we are now using the Live Table Service rather than the development storage version, you will also need to set the correct credentials.

Now that we have had a chance to interact with the Table Service both with the storage client library and the REST API directly, let's look at some of the technologies used to implement the Table Service REST API.

12.1.3 ADO.NET Data Services and AtomPub

ADO.NET Data Services (formerly known as Astoria) is a Data Access Framework that allows you to create and consume data via REST-based API's from your existing data sources (such as SQL Server databases) using HTTP.

Rather than creating a whole new API for the Table Service, the Windows Azure team has implemented the REST-based API's using ADO.NET Data Services. Although not all aspects of the framework have been implemented, the Table Service supports a large subset of the framework.

One of the major advantages of ADO.NET Data Services is that if you are already familiar with the framework then getting started with the Windows Azure Table Service is pretty quick. Even if you have not already used the ADO.NET Data Service previously then any knowledge gained developing against Windows Azure will help you with future developments that may use the framework.

ADO.NET Data Services Client Libraries

ADO.NET Data Services provides a set of standard client libraries that abstract away the complexities of the underlying REST API's and allow you to interact with services in a standard fashion regardless of the underlying service. So whether you are using ADO.NET Data Service with the Windows Azure Table Service or SQL Server, your client side code will be pretty much the same. As seen in the previous chapter, using these libraries to communicate with the table service allows you to develop very simple standard code against the Table Service very quickly.

AtomPub

The Windows Azure table service uses the ADO.NET Data Services implementation of the Atom Publishing Protocol (AtomPub) to interact with the Table Service. AtomPub is an HTTP-based REST-like protocol that allows you to publish and edit resources. AtomPub is a protocol that is often used by Blog Services and Content Management Systems to allow the editing of resources (articles and blog postings) by various third party clients. Windows Live Writer is a well-known example of a blog client that uses AtomPub to publish articles to various blog platforms (blogspot, wordpress, windows live spaces etc). In the case of Windows Azure Storage Accounts, Tables and Entities are all considered as resources.

Although ADO.NET Data Services can support other serialization formats (such as JSON) the Table Service implementation of ADO.NET Data Services only supports AtomPub. In this book, we will not look at the AtomPub protocol specifically, but will only point out its usage.

If you were to look at all the previous examples in this chapter (Table Delete and List Tables in a Storage account) and compared them to the AtomPub protocol, you would see that the REST API's map directly.

If you are interested in reading more about the AtomPub protocol (RFC5023) you can read the full specification at the following web page:

http://bitworking.org/projects/atom/rfc5023.html

Now that we have a basic awareness of AtomPub, we can look at how the AtomPub protocol and the Atom document format are used to create a table using the Table Service REST API.

12.1.4 Creating a Table using the Rest API

In the previous chapter, we created a table using the following storage client call:

```
var storageAccount =
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
    CloudStorageAccount.Parse(
       ConfigurationManager.AppSettings["DataConnectionString"]);

CloudTableClient tableClient =
   storageAccount.CreateCloudTableClient();

tableClient.CreateTableIfNotExist("ShoppingCartTable");
```

Ultimately the storage client library just wraps the REST API exposed by the table service. Let's take at how this is done.

CREATING A TABLE USING ATOMPUB

To create a new table we must perform a POST request against the same URI as we used earlier to list and delete tables in the silverlightukstorage storage account.

```
POST http://silverlightukstorage.table.core.windows.net/Tables
```

Since the Table Service implements the AtomPub protocol the body of the POST request needs to be in the Atom document format. The following Atom document instructs the table service to create a new table called "ShoppingCartTable" in our storage account:

```
<?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>2009-03-18T11:48:34.9840639-07:00</updated>
      <author>
         <name/>
      </author>
      <id/>
      <content type="application/xml">
        <m:properties>
          <d:TableName>ShoppingCartTable</d:TableName>    #A
        </m:properties>
      </content>
   </entry>
```

#A - Name of the table to create (ShoppingCartTable)

As we can see from the very verbose piece of junk above that to create a simple table, you require a whole bunch of useless information that is never used.

My 2 cents about the REST API method of creation

Sub Rant

BEGIN

So this is a complete rant and I do apologize for it but it just has to be said. The REST API for creating tables is unnecessarily complex and verbose. I know you won't care as you will probably use the storage client to create tables rather than the REST API. I also know that it's not Microsoft's fault they are just following the standard ☺ but could we not have a simpler API call?

The method of deleting the shopping cart table is pretty simple; it is just the HTTP verb DELETE with the following URI:

```
http://silverlightukstorage.table.core.windows.net/Tables('
ShoppingCartTable')
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

It's pretty simple isn't it? Why does AtomPub have that very mad method of creating a table? I can't help thinking that a simpler method of creating tables would be to use the same URI as DELETE and change the http verb from DELETE to POST.

END

Let's now look at how we can take this piece of Xml simplicity and actually create the table via the REST API.

CREATING THE TABLE USING THE REST API IN A CONSOLE APPLICATION

We will now create a small console application that will generate the shopping cart table in our storage account using AtomPub and the REST API. Listing 12.2 contains the code for the console application.

Listing 12.2 Creating tables using the REST API and AtomPub

```
static void Main(string[] args)
{
   HttpWebRequest hwr = CreateHttpRequest(new
Uri(@"http://silverlightukstorage.table.core.windows.net/Tables"), "POST", new
TimeSpan(0, 0, 30));                                #1

  string xml = string.Format(@"<?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>{0:yyyy-MM-
ddTHH:mm:ss.fffffffZ}</updated><author><name/></author><id/><content
type=""application/xml""><m:properties><d:TableName>{1}</d:TableName></m:propertie
s></content></entry>", DateTime.UtcNow, "ShoppingCartTable");
                         #2

   byte[] bytes = Encoding.UTF8.GetBytes(xml);                  #3
   hwr.ContentLength = bytes.Length;

   var storageAccount =
      CloudStorageAccount.Parse(
        ConfigurationManager.AppSettings["DataConnectionString"]);   #A

   storageAccount.Credentials.SignRequestLite(hwr)                  #B

   using (Stream requestStream = hwr.GetRequestStream())        #4
   {
      requestStream.Write(bytes, 0, bytes.Length);
   }

   using (StreamReader sr =
      new StreamReader(hwr.GetResponse().GetResponseStream()))  #C
   {
      XDocument myDocument =
         XDocument.Parse(sr.ReadToEnd());

      Console.Write(myDocument.ToString());
   }
}
```

#A Get our credentials as before
#B Sign the request
#C Make the request to the table service

Replace #1-4 in the following paragraphs with a cueball

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

If we look at the code in Listing 12.2 we can see the console application is pretty much the same as the one we used in Listing 12.1 to list tables. There are few differences between the two applications, however. At #1, we have changed the URI to the URI of our table endpoint and we have change the http verb from GET to DELETE. At #2 we set the AtomXml that we defined earlier as a string and convert it to a byte array at #3. Finally, we write the AtomXml byte[] to the request body at #4.

Over the past few sections we've looked at how we can interact with the REST API directly. In each example we've been using the storage client library to sign our request but we really haven't spent any time explaining that. So let's take a little time out to do that now.

12.2 Authenticating Requests against the Table Service

In the blobs chapter we described the shared key authentication method for Blobs. This method of authentication is used by both the Blobs and the Queue Service. The table service however supports 2 different types of authentication mechanisms:

- Shared Key Authentication for Table Service
- Shared Key Lite Authentication for Table Service

Let's take a look at Shared Key Authentication first.

12.2.1 Shared Key Authentication

Shared Key Authentication for Table Service is the most secure method of authenticating against the Table Service using the REST API. The method of generating an authentication key is pretty similar to the method used for Blobs (with a few subtle differences).

As we discovered in the blobs chapter to generate a shared key you would need to canonicalize (see blobs chapter for more details) the http request and then hash it using a SHA256 algorithm storing the hashed value in the Authorization header. The following value represents the authorization header for a request hashed using the Shared Key mechanism:

```
SharedKey devstoreaccount1:J5xkbSz7/7Xf8sCNY3RJIzyUEfnj1SJ3ccIBNpDzsq4=
```

The major difference between the shared key of blobs and the table service is that you do not include the canoncalized headers (or If-xxx-xxx) in the signature. The following code shows you how to generate the string prior to SHA256 hashing.

```
unhashedString =
   VERB + "\n" + Content-MD5 + "\n" + Content-Type + "\n" + Date + "\n" + canonicalizedRequest;
```

Once you have generated the above string, you can then hash it and stuff it into the authorization header.

Although you can use the shared key mechanism with the table service it can only be used with the REST API directly. Unfortunately the ADO.NET Data Service client does not support the shared key mechanism which means that the Table Service also supports Shared Key Lite as an authentication mechanism (which is supported by ADO.NET Data Services Client). Let's take a look at that authentication mechanism.

12.2.2 Shared Key Lite

The Shared Key Lite mechanism of signing a request is very similar to the shared key method of signing requests. Like the Shared Key mechanism, it will canoncalize and hash the request using a SHA256 algorithm storing the result in the authorization header. The following value represents the authorization header for a request hashed using the Shared Key Lite mechanism:

```
SharedKeyLite devstoreaccount1:0c4bknVWVmQ+L1r5jCIYFiNDkSXHata8ZYW8mjQhPLo=
```

Although SharedKeyLite follows the same process of hashing a request and uses the same key to hash the request as the SharedKey mechanism, SharedKeyLite is a lighter and less secure mechanism. The SharedKey

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

mechanism includes more data as part of the hash meaning you have a better chance of tampering with the request using SharedKeyLite rather than SharedKey.

TAMPERING WITH REQUESTS

Replace #4 in the following paragraphs with a cueball

If we return to listing 12.2, we can see that we sign the request prior to writing the Atom Xml to the request body at #4. Therefore we know that the Xml is not part of the hash and can therefore be tampered with. If an http request is intercepted within the time window then you would be able to modify the request to create a table of a different type (e.g. NastyHackerTable). You would not be able to perform any other types of request however as the hash prevents the HTTP verb from being tampered with.

> **Which Authentication Method should I use?**
>
> If you are using ADO.NET Data Services to communicate with the Table Service and your application runs purely within the data center then you can continue to use SharedKeyLite (to be honest you don't have a choice as it's the only authenticated mechanism supported by ADO.NET Data Services Client.
>
> If you are using the REST API directly, have an application communicating with the Table Service API outside of the Windows Azure Data Center or want the highest possible security then you should use the REST API directly with Table Service Authentication,

By now, you should have a good feel for Storage Accounts, the REST API, AtomPub, and which operations you can perform on a storage account using both the REST API directly and the Storage Client library. Next, we will continue to look both at the REST API and the storage client library while we look at how perform cruddy stuff (inserts, updates and deletes, we'll do querying later).

12.3 Modifying entities with the REST API is CRUD

Over the next few sections, we will try and understand how the REST API is used to communicate with the Table Service in regard to entities. In particular we will look at:

- Inserting Entities
- Deleting Entities
- Updating Entities

Before we can delete or update and entitiy, I guess we need one inserted first. Let's look at how that is done.

12.3.1 Inserting Entities

Before we look at how to insert an entity using the REST API, let's look at the code that we would write to store an entity in a table using the storage client library:

```
var shirtContext = new ProductContext();

shirtContext.AddObject("Products",
   new Product
   {
      PartitionKey = "shirts",
      RowKey = "RedShirt",
      Name = "Red Shirt",
      Description = "A Red Shirt"
   });

shirtContext.SaveChanges();
```

In the example above we insert a new instance of our product entity into the "Products" table. The new product will be stored in the shirts partition with a RowKey of "RedShirt". For a detailed description of how data is stored

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

in the table service and how to use a context class to insert data into table, please refer to the first Table Service chapter..

Unit of Work Pattern

ADO.NET Data Services and therefore the Storage Client Table Service implement the Unit of Work pattern for saving data back to the database. This simply means that all changes are tracked locally (AddObject) and then all changes are saved back to the table service when the SaveChanges method is called.

Non Unit of Work Patterns would not track the changes to the entities locally and apply the changes directly to the Table Service when the AddObject method is called.

Now that we have reminded ourselves of how to insert a new entity into a table using the storage client library let's look at how this would be done using the REST API directly.

USING THE REST API

To use the AtomPub based REST API to insert a new entity in a table we would need to perform an http POST request to the following URI:

```
http://<storageaccountname>.table.core.windows.net/<TableName>
```

Therefore to insert a new entity into the Products table in the silverlightukstorage storage account, we would use the following URI:

```
http://silverlightukstorage.table.core.windows.net/Products
```

Finally to insert the same entity as we created earlier with the storage client library we would need to define the request body as the following Atom Xml:

```
<?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>2009-07-27T14:22:48.8875037Z</updated>
  <author>
    <name />
  </author>
  <id />
  <content type="application/xml">
    <m:properties>
      <d:Description>A Red Shirt</d:Description>
      <d:Name>Red Shirt</d:Name>
      <d:PartitionKey>shirts</d:PartitionKey>
      <d:RowKey>RedShirt</d:RowKey>
      <d:Timestamp m:type="Edm.DateTime">0001-01-01T00:00:00</d:Timestamp>
    </m:properties>
  </content>
</entry>
```

If we look at the AtomPub xml that creates our entity, we can see that it follows a very similar format to the xml that we used to create the storage account table in section 12.1.3. As we can see from the AtomXml above that not only are the values of each property of the entity included in the Xml but so is the name and type of each property (notice that the Timestamp property is of type Edm.DateTime). Since a Table Service table is effectively schema-less and each row could contain an entity with entirely different set of properties, it is important that the entities being inserted into the table are self-describing.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

If you wanted to modify the console application in Listing 12.2 to insert an entity instead of creating a table, you could replace the URI at #1 with the URI we defined earlier and replace the Atom Xml at #2 with the above document.

Now that we have created our lovely entity, let's nuke it ☺

12.3.2 Deleting Entities

In the product management web page that we built in the previous Table Service chapter we could delete shirts from our product list using the following call.

```
var shirtContext = new ProductContext();
shirtContext.DeleteObject(shirtToDelete);
shirtContext.SaveChanges();
```

Unit of Work

Deleting an entity is very similar to adding an entity when using the storage client library. If we wished to delete one of our shirts from the Product table then we would need to add the shirt to be deleted to the context objects' (shirtContext) tracking list using the DeleteObject method. All changes are again tracked locally by the context object and are only saved back to the table service when the SaveChanges method is called (following the Unit of Work Pattern)

Let's now take a look at the REST API for deleting entities from a table. To delete an entity from a table we need to make a DELETE request to the following URI passing in the correct table name, partition key and row key:

```
http://silverlightukstorage.table.core.windows.net/Products(PartitionKey="Shirts",
RowKey="RedShirt")
```

The above URI would delete an entity called RedShirt from the Shirts partition of our products table. Listing 12.3 shows the code required to the delete the entity from a console application.

Listing 12.3 Deleting Entities using REST

```
static void Main(string[] args)
{
   HttpWebRequest hwr = CreateHttpRequest(new
Uri(@"http://silverlightukstorage.table.core.windows.net/Products(PartitionKey="Sh
irts", RowKey="RedShirt")
"), "POST", new TimeSpan(0, 0, 30));                                #A

   var storageAccount =
      CloudStorageAccount.Parse(
        ConfigurationManager.AppSettings["DataConnectionString"]);   #B

   storageAccount.Credentials.SignRequestLite(hwr)                   #C

   hwr.GetResponse();                                                #D
}
```

#A - DELETE the Entity from the table
#B - Set up credentials
#C - Sign the Credenitials
#D - Execute the request

Optimizing Delete Performance in Web Grids

As explained previously, to delete a entity using the storage client library, we first need a local copy of the entity in order to add it to context object's tracking list. In windows client applications this is not such a big issue as we will already have a local copy of the entities already. In an ASP.NET application if we listed our entities in a

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Grid (as we did in our product management web page) then we would no longer have a local copy of our entity as each web page call is stateless.

Although we could store a copy of all entities in the ASP.NET page state this would massively increase the page size and reduce overall performance. Similarly storing the entities in the session would put unnecessary overhead on the web server again affecting the performance. Even if the grid is populated from a cache then unnecessary calls to the cache would still have a slight impact on performance. So let's look at what other options we may have.

In the Product management web page in chapter 7 for the sake of simplicity we just fetched the entity from the table service and then added that entity to the shirtContexts' tracking list. The following code shows how we used a LINQ query to fetch the data from the table service:

```
var shirtContext = new ProductContext();

shirtToDelete = (from item in shirtContext.Products
                        where item.PartitionKey == "Shirts"
                         && item.RowKey == e.CommandArgument.ToString()
                 select item).First();
shirtContext.DeleteObject(shirtToDelete);

shirtContext.SaveChanges(saveOptions);
```

Re-fetching the entity to be deleted is not something you should consider in a production environment as any unnecessary calls to the table service will also impact the performance of your application and will add to the overall running cost of your service (calls to the table service are billable).

As we discovered from the REST API, we really don't need all the data of the object. In-fact we only need the Partition Key and Row Key (it's useful understanding this REST stuff). Therefore rather than fetching the whole object we could just construct a lightweight version of the object to be deleted.

In this case we will define a lighweight instance as instance of our Product class that only contains the PartitionKey and RowKey values. As long as the object held in the tracking list holds the correct PartitionKey and RowKey (which if we remember uniquely identifies an entity) then we would have enough have information to perform the delete therefore there is no need to fetch every property of the entity. The following code shows how we can delete an entity from the table service using a lightweight instance:

```
var shirtContext = new ProductContext();
shirtContext.DeleteObject
(
   new ProductKey
   {
      PartitionKey = "Shirts",
      RowKey = e.CommandArgument.ToString()
      }
);
shirtContext.SaveChanges(saveOptions);
```

As you can see in the code above there is no need to fetch the entity from the table service as we already know both our PartitionKey (Shirts) and the RowKey can be extracted from the command argument of the delete button. This optimization saves us that extra fetch.

Now that we have explored both Insert and Delete, it's worth completing the set and looking at updates.

12.3.3 Updating Entities

When deleting an object using the storage client we needed to keep track of the objects to be deleted in the context object for the Products Table. We can use similar logic to update objects in our application. The following piece of code will retrieve the "redshirt" entity from the "shirts" partition in our product table. We will then modify the description of the entity and save the changes back to the product table.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
var shirtContext = new ProductContext();

shirtToUpdate = (from item in shirtContext.Products
                  where item.PartitionKey == "Shirts"
                   && item.RowKey == "redshirt"
                 select item).First();
shirtToUpdate.Description = "I have been modified";
shirtContext.UpdateObject(shirtToUpdate);
shirtContext.SaveChanges(saveOptions);
```

MERGING DATA

By default the SaveChanges method will merge any changes made to the object back to the entity stored in the table service rather than performing a replacement update. Before we can understand what this means we need to look at an extract of AtomXml that describes the entity held in the Table Service.

```
<content type="application/xml">
    <m:properties>
      <d:Description>A Red Shirt</d:Description>
      <d:Name>Red Shirt</d:Name>
      <d:PartitionKey>shirts</d:PartitionKey>
      <d:RowKey>RedShirt</d:RowKey>
      <d:Timestamp m:type="Edm.DateTime">0001-01-01T00:00:00</d:Timestamp>
    </m:properties>
  </content>
```

By merging the date we can efficiently send back data to the table service by only sending the modified data instead of the full entity. Table 12.1 shows how this would work locally, remotely and mergedly (if that's a word).

Table 12.1 - Merging data with updates

Scenario	PartitionKey	RowKey	Name	Description
1	Shirts	RedShirt	Red Shirt	A Red Shirt
2	Shirts	RedShirt	Red Shirt	A Pink Shirt
3	Shirts	RedShirt	Red Shirt	A Pink Shirt

Scenario 1 - Remote copy of the entity stored in the Product Table
Scenario 2 - Local copy of the entity
Scenario 3 - The changes in the local version of the entity are merged back with the remote version

As we can see from table 12.1, the only property that has changed for our entity is the Description. This means that the client application does not need to send back the Name property in the Atom Xml describing the entity. The following extract of the AtomXml would describe what would be returned back to the table service as part of the merge operation

```
<content type="application/xml">
    <m:properties>
      <d:Description>A Pink Shirt</d:Description>
    </m:properties>
  </content>
```

If you need to replace the entity stored in the table service with your local version rather than performing a MERGE then you can use the following setting in your storage client code.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
shirtContext.SaveChangesDefaultOptions =
    SaveChangesOptions.ReplaceOnUpdate;
```

The Atom Xml sent using the REST would instead contain the full description of the entity rather than just the changed properties in this case.

USING THE REST API TO MERGE OR UPDATE

Using the REST API to update or merge data is really a combination of the DELETE and the insert REST API's.

The URI to update or merge the local entity back to the table storage is the same URI as the delete operation as shown below:

```
http://silverlightukstorage.table.core.windows.net/Products(PartitionKey="Shirts",
RowKey="RedShirt")
```

As you can see in the above URI you need to specify the PartitionKey and RowKey of the entity being modified. Depending on the operation that you are performing you should set the http verb to either MERGE or PUT. Finally the body of the http request should be set to the AtomPub xml document that describes the entity (this is the same as the xml used to create the entity).

If you wish to modify the insert entity console application then you would need to change URI and http verbs in the code.

If-Match Header

Finally you would need to add a new header to the request called "If-Match". This header is used to ensure that the date held in the remote version of the entity has not changed since you grabbed the local version. If you wish to ensure that data is only modified if the data is unchanged then you should set the If-Match header to the entities e-tag that was returned originally.

If you wish to perform an unconditional update then the value of this header should be set to "*".

In this section, we've gained knowledge of how we can perform inserts, updates and deletes against our entities. However we tend not to work with single entities, so we need to get an understanding of some of the complications of updating data (i.e. batching and transactions).

12.4 Batching Data

In the previous sections, we used both the storage client and the REST API to insert new entities into the Product table. In this section, we will look at how we can both improve performance and perform transactional changes by batching up data.

The following code inserts multiple entities into the product table using the storage client library:

```
var shirtContext = new Products();

for (int i = 0; i < 10; i++)
{
   shirtContext.AddObject("Products",
   new Product
   {
      PartitionKey = "shirts",
      RowKey = i.ToString(),
      Name = "Shirt" + i.ToString(),
      Description = "A Shirt"
   });
}
shirtContext.SaveChanges();
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The above code will create 10 new shirts and add each new shirt to a list of objects that are to be tracked by calling the AddObject method on the shirtContext object. Following the Unit of Work Pattern, the context object will not send any changes back to the Table Service until the SaveChanges method is called. It will then enumerate through the list of tracked objects and insert them into the Product table. By default the SaveChanges call will insert each entity into the table one by rather than batching the inserts into a single call. Figure 12.2 shows a capture of HTTP Traffic for the above call using Wireshark (a packet sniffing tool)

12	21.950029	192.168.0.5	65.55.33.205	HTTP	POST /ProductTable HTTP/1.1 (application/atom+xml)
16	22.241154	65.55.33.205	192.168.0.5	HTTP	HTTP/1.1 201 Created (application/atom+xml)
17	22.251847	192.168.0.5	65.55.33.205	HTTP	POST /ProductTable HTTP/1.1 (application/atom+xml)
22	22.633432	65.55.33.205	192.168.0.5	HTTP	HTTP/1.1 201 Created (application/atom+xml)
23	22.641256	192.168.0.5	65.55.33.205	HTTP	POST /ProductTable HTTP/1.1 (application/atom+xml)
27	22.898821	65.55.33.205	192.168.0.5	HTTP	HTTP/1.1 201 Created (application/atom+xml)
28	22.906475	192.168.0.5	65.55.33.205	HTTP	POST /ProductTable HTTP/1.1 (application/atom+xml)
32	23.163614	65.55.33.205	192.168.0.5	HTTP	HTTP/1.1 201 Created (application/atom+xml)
33	23.171723	192.168.0.5	65.55.33.205	HTTP	POST /ProductTable HTTP/1.1 (application/atom+xml)
37	23.429996	65.55.33.205	192.168.0.5	HTTP	HTTP/1.1 201 Created (application/atom+xml)
38	23.438125	192.168.0.5	65.55.33.205	HTTP	POST /ProductTable HTTP/1.1 (application/atom+xml)
42	23.694816	65.55.33.205	192.168.0.5	HTTP	HTTP/1.1 201 Created (application/atom+xml)
43	23.702653	192.168.0.5	65.55.33.205	HTTP	POST /ProductTable HTTP/1.1 (application/atom+xml)
47	23.960189	65.55.33.205	192.168.0.5	HTTP	HTTP/1.1 201 Created (application/atom+xml)
48	23.967479	192.168.0.5	65.55.33.205	HTTP	POST /ProductTable HTTP/1.1 (application/atom+xml)
52	24.226081	65.55.33.205	192.168.0.5	HTTP	HTTP/1.1 201 Created (application/atom+xml)
53	24.233846	192.168.0.5	65.55.33.205	HTTP	POST /ProductTable HTTP/1.1 (application/atom+xml)
57	24.491098	65.55.33.205	192.168.0.5	HTTP	HTTP/1.1 201 Created (application/atom+xml)
58	24.499442	192.168.0.5	65.55.33.205	HTTP	POST /ProductTable HTTP/1.1 (application/atom+xml)
62	24.756697	65.55.33.205	192.168.0.5	HTTP	HTTP/1.1 201 Created (application/atom+xml)

Figure 12.2 - By default the context object will save each entity as an individual request rather than as a batch.

As you can see from figure 12.2, to insert 10 shirts, the application must perform 10 HTTP POST requests to the table service. If you are inserting a large number of entities then this method of individually inserting data into the table will cause large performance problems if your application is outside of Windows Azure or your Web / Worker role is not affinitized to the same data center as your storage account.

Be aware of Latency outside of the Data Center

Due to latency, inserting 10 shirts using the above code took **4 seconds** between my machine and the live table service. When running the same code as a web role in the windows azure data center, it took milliseconds.

Although minimizing latency will give large performance benefits you can gain larger performance improvements by batching up inserts into single calls using the Entity Group Transactions functionality.

PERFORMANCE NOTE

ALWAYS affinitize your web roles, worker roles and storage service to the same data center to minimize latency.

12.4.1 Entity Group Transactions

Passing SaveChangesOptions.Batch into the SaveChanges calls as shown below will batch up all changes into a single http POST

```
shirtContext.SaveChanges(SaveChangesOptions.Batch);
```

By batching up the data as above, this reduced the same insert of 10 shirts from my local machine to the live service from 4 seconds to 1 second.

By default when no SaveOption (or SaveOptions.None) is passed as part of the SaveChanges method and a tracked entity fails to be inserted then the context object would stop attempting to save any further entities. Any entities that were saved successfully would not be rolled back and would remain in the table.

If SaveOptions.ContinueOnError is passed as part of the SaveChanges call and an entity fails to save then the context object will continue to save all other entities.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Finally if you pass SaveOptions.Batch as part of the SaveChanges call then all entities will be processed as a batch in the scope of a single transaction (known as an entity group transaction). If any of the entities being inserted as part of the batch fails to be inserted then the whole batch will be rolled back.

Rules of using entity group transactions

1) A maximum of 100 operations can be performed in a single batch

2) The batch may not exceed 4Mb in Size

3) All entities in the batch must have the same partition key

4) You can only perform a single operation against an entity in a batch

In this book we will not discuss the REST implementation of entity group transactions due to the complexity of the implementation. It is worth pointing out that if you decide to use the REST implementation, the Table Service only implements a subset of the available functionality. As of the PDC09 release, the Table Service only supports single changesets (a changeset being a set of inserts updates or deletes) within a batch. If you are interested in looking at the REST implementation of batching, then you should look at the following MSDN article:

http://msdn.microsoft.com/en-us/library/dd894038.aspx

Entity Group Transactions are executed using an isolation method known as Snapshot Isolation. This method of Isolation is a standard method used in relational databases such as SQL Server or Oracle, it is also known as Multi Version Concurrency Control (MVCC). At the beginning of a transaction a snapshot of the data is taken which is used for the duration of the transaction. This means that all operations within the transaction will use the same set of isolated data that cannot be interfered with by other concurrent processes. Since the data is isolated from all other processes then there is no need for locking on the table meaning that operations cannot be blocked by other processes. On commit of the transaction if any modified data has been changed by another process since the snapshot began then the whole transaction must be rolled back and retried.

12.4.2 Retries

In order to handle such a model then your code must be able to perform such a retry. The ability to handle retries is built into the Storage Client library and can be configured using the following code:

```
shirtContext.RetryPolicy =
    RetryPolicies.Retry(5, TimeSpan.FromSeconds(1));
```

The above retry policy will re-attempt the SaveChanges operation up to 5 times, retrying every second. If you do not wish to set a retry policy then you can always set the policy as no retry

```
shirtContext.RetryPolicy = RetryPolicies.NoRetry;
```

If you need to provide some more complicated retry polices with randomized back-off timings or if you wish to define your own policy this can also be achieved by setting an appropriate retry policy. Unfortunately, if you are using the REST API directly then you will need to roll your own retry logic.

Finally, in order to make use of the retry logic, you will need to use the SaveChangesWithRetries method rather than the SaveChanges method as shown below:

```
shirtContext.SaveChangesWithRetries();
```

Use for queries too

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

> Although retry policies are vital when using entity group transaction they can also be useful when querying data. If you remember that your web and worker roles are based in the cloud and can shutdown and restarted at anytime by the fabric controller (such as a Windows Update) then in order to provide a less buggy-looking application, it may be advisable to use retry policies when simply querying data.

So far we have covered the modification of data in quite a lot of detail. As we have discovered in this section however entity group transactions can also be useful for querying data. With that in mind it's probably worth breaking away from data updates and now focusing on how to retrieve data via the REST API.

12.5 Querying Data

In this section we will take a look at how we can query data held in the table service using both the storage client library and the REST API directly. The knowledge that we gain from this section will help us when we look at how to store data efficiently. In particular we will look at how we can:

- Retrieve entities using the REST API
- Query using LINQ
- Filter Entities using the REST API
- Filter Entities using LINQ
- Select Data using LINQ
- Paging Data

To get started we will look at how we can return data from a table using the REST API.

12.5.1 Retrieving all entities in a table using the REST API

We will now look to build a small console application that will display all the entities in our Product Table . This application is very similar to the one that we built earlier to list tables in storage accounts.

The base URI used to query the product table is the same base URI as we used to Insert, Update and Delete entities from the table. To return all shirts stored in the Product table in the Development Storage account you would make an http GET request using the following URI:

```
http://127.0.0.1:10002/devstoreaccount1/Products
```

The code in Listing 12.4 will return all entities in our product table and display the result in the Console Window.

Listing 12.4 Listing the entities in the product table using the REST API

```
static void Main(string[] args)
{
   HttpWebRequest hwr =
      CreateHttpRequest(new Uri(@"http://127.0.0.1:10002/devstoreaccount1/Products"),
"GET", new TimeSpan(0, 0, 30));                                                  #1

   var storageAccount =
      CloudStorageAccount.Parse(
        ConfigurationManager.AppSettings["DataConnectionString"]);   #A

   storageAccount.Credentials.SignRequestLite(hwr)                   #B

   using (StreamReader sr = new StreamReader(hwr.GetResponse().GetResponseStream()))
                  #C
   {
      XDocument myDocument = XDocument.Parse(sr.ReadToEnd());
      Console.Write(myDocument.ToString());
   }
}
```

#A Setup the credentials for our storage account

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

#B Sign the request
#C Execute the request and display the Xml Output

Replace #1 in the following paragraphs with a cueball

The code in listing 12.4 is pretty much the same code in Listing 12.1 that we used earlier to list all the tables in a storage account. The only modification to that code is that we have changed the URI of the http request at #1 to the URI of our product table.

The REST API call that we used to list all product entities adheres to the AtomPub protocol and therefore the result that will be returned to the console window will display all entities stored in the product table in Atom Xml format as shown below:

Listing 12.5 - AtomXml output from the Console Application in Listing 12.4

```
<feed xml:base="http://127.0.0.1:10002/devstoreaccount1/" xmlns:d="http://schema
s.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">Products</title>
  <id>http://127.0.0.1:10002/devstoreaccount1/Products</id>
  <updated>2009-08-01T11:23:48Z</updated>
  <link rel="self" title="Products" href="Products" />
  <entry m:etag="W/&quot;datetime'2009-07-23T19%3A55%3A38.7'&quot;">

<id>http://127.0.0.1:10002/devstoreaccount1/Products(PartitionKey='Shirts',RowKey=
'BlueShirt')</id>
    <title type="text"></title>
    <updated>2009-08-01T11:23:48Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="Products" href="Product(PartitionKey='Shirt
s',RowKey='BlueShirt')" />
    <category term="AiAChapter7Web_WebRole.Products" scheme="http://schemas.
microsoft.com/ado/2007/08/dataservices/scheme" />
    <content type="application/xml">
      <m:properties>
        <d:Name>Blue Shirt</d:Name>
        <d:Description>A Blue Shirt</d:Description>
        <d:Timestamp m:type="Edm.DateTime">2009-07-23T19:55:38.7</d:Timestamp>
        <d:PartitionKey>Shirts</d:PartitionKey>
        <d:RowKey>BlueShirt</d:RowKey>
      </m:properties>
    </content>
  </entry>
  <entry m:etag="W/&quot;datetime'2009-07-23T19%3A13%3A28.09'&quot;">
    <id>http://127.0.0.1:10002/devstoreaccount1/Products(PartitionKey='Shirt
s',RowKey='redshirt')</id>
    <title type="text"></title>
    <updated>2009-08-01T11:23:48Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="Products"
href="Product(PartitionKey='Shirts',RowKey='redshirt')" />
    <category term="AiAChapter7Web_WebRole.Products"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <content type="application/xml">
      <m:properties>
        <d:Name>red shirt</d:Name>
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
          <d:Description>a red shirt</d:Description>
          <d:Timestamp m:type="Edm.DateTime">2009-07-23T19:13:28.09</d:Timestamp>
          <d:PartitionKey>Shirts</d:PartitionKey>
          <d:RowKey>redshirt</d:RowKey>
        </m:properties>
      </content>
    </entry>
  </feed>
```

VERBOSITY OF THE RETURNED ATOMXML

The AtomXml output above returns both entities stored in the Product table (redshirt and blueshirt). As you can see the returned xml is very descriptive (or verbose) including the name of the property, the value and the data type.

With the returned xml being so verbose it means that returned datasets are usually pretty large so you should be careful to cache data wherever possible (explained in later chapters) and return the minimum amount of data required.

Hopefully in the future the Windows Azure table service team will support a less verbose serialization format such as JSON and also support LINQ Data Shaping (explained later on in this section). JSON would be an ideal format to support as ADO.NET Data Services (just not the table service implementation) already supports this method of serialization. Using JSON would require very little changes to your application code but would gain very large benefits in terms of reduced bandwidth. The following code shows how the previously returned atom xml could be represented in JSON:

```
Products:
 [
      {
             "Name" : "blue shirt"
             "Description" : "a Blue Shirt"
             "Timestamp" : "2009-07-23T19:13:28.09"
             "PartitionKey" : "Shirts"
             "RowKey" : "blueshirt"}
     },
     {
             "Name" : "red shirt"
             "Description" : "A Blue Shirt"
             "Timestamp" : "2009-07-23T19:13:28.09"
             "PartitionKey" : "Shirts"
             "RowKey" : "redshirt"}
     },
]
```

As you can see, the JSON representation is much more readable and terse meaning that the size of the returned documents would be greatly reduced and will therefore improve the speed of your application. Hopefully this will be supported in future versions of the Table Service.

12.5.2 Querying with LINQ

Since the Table Service is implemented using ADO.NET Data Services, we can use the ADO.NET Data Services client library to perform server side queries using LINQ rather than querying the REST API directly.

The following code shows how we exposed our Products Table in the Product Context class that we created in Listing 11.2:

```
public DataServiceQuery<Product> Products
{
   get{return CreateQuery<Product>("Products");}
}
```

Rather than executing and returning a list of products from the Products Table stored in the table service the Products property will generate and return a new query that will not be executed until the collection is

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

enumerated. Since the execution of the DataServiceQuery is deferred this means you can modify the returned query to include any filters that you may require prior to the execution of the query.

Since DataServiceQuery implements the IQueryable interface that means we can define the query that should be executed by the table service in our application using LINQ. The following code is a LINQ query that will return all the products in the shirts partition of our product table:

```
var shirts = from shirt in shirtContext.Products
             where shirt.PartitionKey == "shirts"
             select shirt;

foreach (var shirt in shirts)
{
}
```

In the above query the Products property of shirtContext is IQueryable hence why we can make this the datasource of our LINQ query. Because the query will not be passed to the Table Service for execution until the "for loop" is executed, we can therefore add additional filter criteria to the query (i.e. restricting the returned data to only those shirts that reside in the shirts partition).

NOTE

If we did not include the "where" criteria from our LINQ statement, then the underlying REST API that is executed would be the same call that we made in listing 12.4.

12.5.3 Filtering Data with the REST API

In the previous section, we modified the LINQ query to include a WHERE clause to restrict the data returned from the table service to only include those shirts in the "shirts" partition. We will now look at how we can modify our usage of the REST API from Listing 12.4 to perform server side filtering.

RETURNING A SINGLE ENTITY

As stated previously the combination of the PartitionKey and the RowKey uniquely identifies an entity in a table. If we therefore wish to return a single entity from a table and we know these two values we can efficiently return the entity. The following URI shows how to return the "redshirt" entity from the "shirts" partition:

```
http://127.0.0.1:10002/devstoreaccount1/Products(PartitionKey='shirts',RowKey='redshirt')
```

Replace #1 in the following paragraphs with a cueball

To execute this query, we could run our console application in Listing 12.4, replacing the URI at #1 with the above URI. The AtomXml returned from the query would be similar to the data returned in Listing 12.5 but would only contain the "redshirt" entity.

QUERYING ENTITIES

If we need to return zero or more entities based upon a filter criteria (such as all shirts that cost $10) we could use the following URI to define our REST query:

```
http://127.0.0.1:10002/devstoreaccount1/Products$filter=<query>
```

Replace #1 in the following paragraphs with a cueball

We should replace the <query> with the filter that we wish to run server side. On execution of the query all entities that match the query will be returned in Atom Xml format as shown in Listing 12.5. Again, we can modify the console application in Listing 12.4 at #1 to use the new URI.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

QUERY EXPRESSIONS

In the following scenario we will look at the syntax of the Query expression applied in the REST API before we look at how we can use these queries in the more familiar LINQ syntax. It is worth noting that as of the PDC09 release of the Table Service only supports the query expressions listed in table 12.2.

Table 12.2 All query expressions supported by the Table Service

Supported Query Expression	Description (C# equivalent operator)
eq	equals (==)
gt	greater than (>)
ge	greater than or equal to (>=)
lt	less than (<)
le	less than or equal to (<=)
ne	not equal to (!=)
and	and (&&) - boolean properties only
not	not (!) - boolean properties only
or	or (||)- boolean properties only

To apply these queries to our REST API query to return all shirts with the description "A Red Shirt" we would define the following URI:

```
http://127.0.0.1:10002/devstoreaccount1/Products$filter=
Description%20eq%20'A%20Red%20Shirt'%
```

To return all shirts in the "shirts" partition that have the description "A Red Shirt" the following URI would be used:

```
http://127.0.0.1:10002/devstoreaccount1/Products$filter=
PartitionKey%20eq%20'shirts'%20and%20Description%20eq%20'A%20Red%20Shirt'%
```

12.5.4 Filtering Data with LINQ

In the previous section, we looked at how we can filter queries server side using the REST API. We will now look at how the REST API maps onto the LINQ queries. As you may have figured, the LINQ queries that we generate earlier eventually get resolved to the REST API URI's that we specified in the previous section. Therefore, although the LINQ has a large and rich syntax only those methods which map directly to the REST API can be supported. Figure 12.3 shows a LINQ query being mapped back to the REST API.

```
var shirtContext = new ProductContext();
    ⊞ shirts | {http://silverlightukstorage.table.core.windows.net/ProductTable()?$filter=PartitionKey eq 'shirts'}
var shirts = from shirt in shirtContext.ProductTable
             where shirt.PartitionKey == "shirts"
             select shirt;
```

Figure 12,3 Mapping a LINQ Query back to the REST API

While we are debugging a LINQ query in Visual Studio, you can either hover over or put a watch on the context object (shirtContext) and you will be able to see the underlying REST API query. Figure 12.3 shows the REST API query for a LINQ query that returns all products in the "shirts" partition. We will now look at the typical queries that you will be able to perform.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

EQUALITY COMPARISONS

If we look at the table of supported query expressions in Table 12.2 we can see that only equality, range comparisons and Boolean comparisons can really be performed using the table service. The following queries are typical equality comparisons that can be performed:

```
where shirt.RowKey == "Red Shirt"
```

or

```
where shirt.Description != "A Red Shirt"
```

or

```
where shirt.Partition == "Shirts"
&& shirt.Description != "A Red Shirt"
```

RANGE COMPARISIONS

The table service also supports the filtering of range data using range queries. For example, the following WHERE clause would returns those shirts priced between 50 and 70 dollars

```
where shirt.Price >= 50 && shirt.Price < 70
```

Since data is stored in the Table as they're native types rather than as string representations the table service will perform comparison routines using the native types rather than performing string comparisons. The following query will return all shirts greater than $50.20

```
where shirt.Price >= 50.20
```

If this query was performed as a string comparison (which you would have to do with Amazon SimpleDB) then it would not returns shirts priced at $60 (as there are less characters in the string than 50.20) unless the data was stored as 60.00. In Windows Azure Table Service the only time you need to worry about performing equivalent string comparisons is if you store a non-native string type as a partition or row key. Partition and Row Keys are always represented as strings in the Table Service and therefore if you need to perform range comparisons you will need to ensure that you ensure that string lengths of the stored data are correct.

BOOLEAN LOGIC

As stated earlier, the Table Service does respect property types. This means you can perform Boolean logic against entity properties that are defined as bool. For example you could perform the following WHERE clause against a shirt that is marked as a genuine Hawaiian shirt e.g.

```
where shirt.IsMadeInHawaii && shirt.Price > 50
```

PREFIX QUERIES

Using the Range Comparision and Boolean Logic as described above we can manipulate our LINQ / REST queries return all entities that start with a particular string. For example if we wanted to return all shirts that were present in "partition1", "partition2", "partition3" or "partition4" we could use the following query:

```
where shirt.PartitionKey.CompareTo("Partition1") >= 0 &&
shirt.PartitionKey.CompareTo("Partition5") < 0
```

LINQ TO OBJECT QUERIES

Even though only a small subset of the LINQ syntax is available to be executed by the table service you can still perform in-memory LINQ queries (LINQ to Objects). Although in-memory LINQ queries do provide full access to the LINQ syntax all queries are executed client side and therefore will required the full dataset to the be returned by the table service first. This approach is not suitable for situations where you are working with a large set of data.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

By now you should have a flavor of the types of queries that you can perform against the Table Service, let's now look at how we can shape the data that is returned from our queries.

12.5.5 Selecting Data using the LINQ Syntax

As you will have noticed in the supported LINQ syntax list there was no mention of the SELECT statement. Of course we can use the SELECT statement to return the entire entity but we cannot use the SELECT to instruct the table service to only return a subset of the entity properties.

RETURNING AN ENTIRE ENTITY USING SELECT

To illustrate the point, let's relook at a LINQ query that returns a product entity in its entirety:

```
var shirts = from shirt in shirtContext.Products
             where shirt.PartitionKey == "shirts"
             select shirt;
```

The LINQ statement above is the same query that we used earlier to return all entities that reside in the Product Table. The code below is an Atom Xml extract of one of the entities returned by the above LINQ query:

```
<content type="application/xml">
   <m:properties>
    <d:PartitionKey>shirts</d:PartitionKey>
    <d:RowKey>shirts0</d:RowKey>
    <d:Timestamp m:type="Edm.DateTime">
       2009-07-29T21:14:45.022Z
    </d:Timestamp>
    <d:Description>A Shirt</d:Description>
    <d:Name>shirtshirts0</d:Name>
 </m:properties>
</content>
```

Looking at the XML for the returned entity we can see that every property of our product entity is returned by the Table Service (PartitionKey, RowKey, Timestamp, Description and Name). If our products table was held in SQL Server rather than the Table Service and our LINQ statement was executed against the database using LINQ2SQL or LINQ2Entities then the following SQL statement would be generated and executed on the SQL Server database.

```
SELECT PartitionKey, RowKey, Timestamp, Description, Name
FROM Products
WHERE PartitionKey = 'shirts'
```

SHAPING THE QUERY

Using LINQ2SQL or LINQ2Entities if we do not need to return the entire entity (e.g. we only need the Name and Description properties of the entity) then we may wish to write a more efficient LINQ query that only requests and returns those columns from our SQL Server Database. The following SQL statement represents the desired SQL to be executed on the SQL Database.

```
SELECT Name, Description
FROM Products
WHERE PartitionKey="shirts"
```

As we can see from the code above the desired SQL only requests the Name and Description column. The above SQL statement be less intensive to execute on the server (as there is less data being queried) and will also utilize less network bandwidth due to the reduced dataset being returned to the application.

Using LINQ2SQL or LINQ2Entities we can modify our LINQ statement to generate the more efficient SQL statement by modifing the select statement from:

```
select entity
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

To:

```
 select new
{
   Name = newShirt.Name,
   Description = newShirt.Description
};
```

This would modify our previous LINQ example to the following.

```
var shirts = from shirt in shirtContext.Products
             where shirt.PartitionKey == "shirts"
             select new
             {
                Name = newShirt.Name,
                Description = newShirt.Description
             };
```

Unfortunately, since the Table Service does not support data shaping using the SELECT statement if you attempted to run the above LINQ query it would generate a very nasty exception. Therefore whenever you are executing queries against the table service every property of the entity will always be returned as part of the query.

If you do really need to shape the returned data in your application and do not mind that the entire entity will be returned from the server you always use the following code:

```
var shirts = from newShirt in
             (
                from shirt inshirtContext.Products
                where shirt.PartitionKey == "shirts"
                select shirt
             ).ToList()
             select new
             {
                Name = newShirt.Name,
                Description = newShirt.Description
             };
```

In the example above we use the same LINQ query as before to filter our data in the table service but this time we return the entire entity. By calling the ToList method on the inner LINQ query we can ensure that the server side query will return all properties of the entity.

Finally the result of the ToList method is fed into our outer LINQ2Object query that performs in-memory shaping of the data returning a new anonymous type containing the 2 properties that we need. Be aware that although this query returns us the entities shaped as we desire it will not improve server side or bandwidth efficiency. If we have a very large entity with an infrequently used property that we do not need in a particular query this unused property will still be returned by the Table Service.

12.5.6 Paging of Data

By default SELECT queries will only return 1000 items in a single result set. Not only is the default amount of data returned but it is also the maximum amount of data returned. If you wish to set a smaller amount of data to be returned you can achieve this using the take statement in LINQ as follows:

```
(from shirt inshirtContext.Products
where shirt.PartitionKey == "shirts"
select shirt).Take(100);
```

The above LINQ statement will return the first 100 items in the shirts partition. The LINQ Take extension method would be resolved to the following querystring parameter in the URI for the REST API call.

```
&top=100
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

If there are more items that can be returned in the query than is present in the resultset then continuation tokens will be provided to allow you to retrieve the next set of data in the query. This method of using continuation tokens effectively provides a method of paging.

If we wanted to return the all items in the shirts table (but we potentially have more than 1000 items) we may run the following REST API query:

```
http://silverlightukstorage.table.core.windows.net/Products()?$filter=PartitionKey%20eq
%20'shirts'&NextPartitionKey=shirts&NextRowKey=1001
```

Since there are more than 1000 items returned in the query, we receive the following continuation tokens in the response.

```
x-ms-continuation-NextPartitionKey: Shirts
x-ms-continuation-NextRowKey: 1001
```

Therefore if we wanted to return all those items in shirts partition that were not returned as part of the original query, we could return the next set of data using the following query:

```
http://silverlightukstorage.table.core.windows.net/Products()?$filter=PartitionKey%20eq
%20'shirts'&NextPartitionKey=shirts&NextRowKey=1001
```

The above query would then return us all products in the "shirts" partition from RowKey 1001 onwards.

12.6 Summary

In this chapter, we have taken quite deep dive into using both the storage client library and the table service REST API.

We have learned how to use the REST API to modify tables in our storage account, and perform CRUD functionality against those tables with our entities. Using our new found REST knowledge we have looked at the AtomPub format and understand how this impacts our application. Finally we looked at how we can efficiently update and query data.

By gaining an understanding of the REST API we know how to maximize the performance of our application as well as understanding the limits of the service.

From the last 2 chapters we should now have a pretty good idea on how to use the Table Service. Later on in the book, we will look at how we can apply this knowledge in our applications.

For now, we will continue on our data journey and look at SQL Azure, the relational database in the Cloud.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

13

SQL Azure and Relational Data

In this chapter:

- How to leverage the power of SQL Server for your cloud applications
- Easy ways to migrate an on-premises db to the cloud
- Find the potholes to avoid

Most applications that work with data today use a relational data model. It is a model we are familiar with, and we know how to manage and develop with it. SQL Server has been with us, by our side all these years, and it is going to be there with us as we move into the cloud.

Over the years SQL Server has matured to meet the different needs of its customers. It started as a spunky departmental server, and moved into the desktop space, mobile device space, and into the enterprise space. The data engine was the first component to ever make any of these moves. The rest of the components would follow shortly after, such as Integration Services (SSIS), Reporting Services (SSRS), and Analysis Services (SSAS). The cloud is not any different. The SQL team is bringing the data engine to the cloud first, and will follow it quickly with the rest of the components of the system.

13.1 - The march of SQL Server to the cloud

When Azure was first released as a CTP in November of 2008 at the PDC SQL Azure wasn't on stage. There was something like SQL involved, but it wasn't a relational engine. It was more like Azure Tables, but geared for true enterprise needs (beyond just the massive scale Tables gives you).

Developers were scared and confused. If they were going to move to the cloud, they surely need a data platform they were comfortable with, something that made it easy for them to migrate their applications without having to rewrite the data tier. Microsoft heard this feedback, tabled what they had (pun unfortunately intended) and delivered what we call SQL Azure today.

Developers wanted something like SQL in the cloud because they needed a data platform they could easily migrate to from on-premises databases. SQL Azure not only gives you the relational database you are used to, but also supports all of the common tools and APIs you are used to working with, such as ADO.NET and TDS.

You can easily port a traditional database from SQL Server to SQL Azure in a matter of days, if not hours. There are some restrictions, and we will cover those, but for the most part it is easy and straight forward.

Future versions of SQL Azure are sure to contain the other systems of SQL Server. Many customers we speak with want to keep their applications local, but move all of their business intelligence applications, and their heavy compute needs to the cloud.

13.2 - Setting up SQL Azure

The first step in creating a database in SQL Azure is to log into your Azure portal, and select the SQL Azure tab on the left. If you haven't created a SQL Azure server yet, you will be prompted to create a SQL Azure management account. This is essentially like the SA account of a normal SQL Server. You will need to provide a

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

username, password, and which datacenter region you would like the server to be provisioned. You will most likely want it in the same region that your Azure applications are running in.

Figure 13.1 - Creating a server administrator account for SQL Azure is easy. This is essentially your SA account in the cloud. You can't make your password <blank>.

Once your account is created, your server will be provisioned and you will be given the server name. This screen is where you are able to manage your databases, and your SQL Azure firewall settings. In figure 13.2, my server name is mlwwmqca6u. Sometimes you will need the fully qualified server name, which in my case is mlwwmqca6u.database.windows.net.

Figure 13.2 - Your SQL Azure server name. It is short and easy to remember. I know I can turn this one into a backronym.

Once you have a SQL Azure server provisioned, you will be able to create database in that server. An on-premises SQL Server has a lot of infrastructure concerns with it. How much RAM, the file groups, what resources are assigned to which instance, resource governing, and so on. You have none of these concerns with a SQL Azure server. In this case your server is just a logical grouping of the databases you have created, and does not represent a pool of resources are all. The only resource shared by a SQL Azure server is the firewall configuration. Your SQL Server in the cloud isn't really a server, but a collection of settings. It mimics a server when you connect to it, but your databases are actually spread across a farm of commodity hardware running SQL Server.

13.2.1 - Creating the database

You will be given your own master database, which is listed in Figure 13.3. At the bottom of the list you will need to use the 'create database' button.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 13.3 - Your database list and the buttons to manage them. I haven't created any yet, but I was given my own master database. You aren't charged for the master database.

You can create a database manually by connecting to the master database with the SQLCMD command prompt application and executing a CREATE DATABASE command. You can provide a MAZSIZE parameter to select either the 1 or 10 GB size. If you do not select a size, a 1GB size database will be created. You will not be able to connect with SQLCMD, or anything else, until you configure the SQL Firewall to let you. We will cover this in a little bit.

Figure 13.4 - Adding a new database to SQL Azure with the SQLCMD tool, just like in the old days. You have to have the firewall configured to do this first.

13.2.2 - Connecting to your database

Once you click the 'create database' button, you will be prompted to provide a database name, and choose the size limit of the database. Your options for a size limit are currently 1GB and 10GB. The reasons for these sizes will be discussed later in this chapter. I created a database called AzureInAction, with a size of 1GB. Once I have created my database, I can retrieve the connection string to that database by selecting it in the list and using the 'connection string' button near the bottom of the list.

Figure 13.5 - SQL Azure connection strings are provided in the management portal. This makes it easy to know what you need to use in your code. Sometimes this is all you need to do to move your application's data to the cloud.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The connection string will contain the full address of the server, your username, and a place for your password. You will eventually create additional SQL user accounts in your database. You should create a special account for each user or application using the db, and not let them use the sa account. The sa account should only be used by you for management purposes.

Notice the brianhprince@mlwwmqca6u user name in the ODBC connection string. Some tools will require this form of a username because of how they connect to a SQL Server. This is because of how they implement the TDS protocol, which is what underlies all connections to SQL Server.

One last thing we need to do to start using our database is to allow Azure based applications to connect to the database. This is done on the firewall tab of the main administration screen. You will need to enable the 'Allow Microsoft Services to access this server' check box. This will create a default rule allowing Azure based applications the right to connect to your server. The firewall will be discussed in more detail later in this chapter.

Figure 13.6 - Enabling the SQL Firewall to allow Azure application traffic to connect to your database. Only IP addresses listed on this screen are allowed to connect to your databases. The checkbox at the top allows for all Azure services under your account to connect.

When you have completed these few steps you will have created a server, created a database on that server, and retrieved the connection strings you will need to start using the server.

13.3 - Size Matters

There are two basic size limits you can choose from in SQL Azure, 1GB and 10GB. These sizes are the maximum size that database can be, and is the basis for how Microsoft will bill you for usage. The 1GB database (the Web Edition) is charged at $9.99 per month, and the 10GB (called the Business Edition) is charged at $99.99 per month. The charges are fixed, and do not take into account the real size of the database, just the potential maximum size it can have. The size of your master database, logs, and indexes do not count towards the total size of the database when it comes to the size limits and charges.

You will also be charged for any bandwidth usage by SQL Azure server that crosses a data center boundary. For example, if you have a mobile application accessing the SQL Azure database, bandwidth used by SQL Azure to those clients will be metered and billed for. If your Azure application is using the db, and they both reside in the same data center (aka geographic region) then the bandwidth used will not be billed for because you aren't crossing a datacenter boundary.

The size of a db is fixed when it is created. If a 1GB database grows too large it will stop inserting new data. Databases will not automatically upgrade to a larger size. If you need a larger size you will need to create a new database and migrate your schema and data. One way to do this would be with SSIS or bulk copy commands.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The biggest reason for this limited size selection is the SLA that Microsoft provides to its customers. SQL Azure runs on a very scalable and robust platform, with your data being replicated to different physical servers and disks in real time. The 1GB and 10GB sizes are the only sizes they feel they can currently support with the SLA defined as 'Monthly Availability' of 99.9%. At this time they couldn't guarantee a high SLA like this with a massive database. How quickly could you restore full function to a 32TB database after a cataclysmic failure?

You should select the smallest size necessary to run your application. Currently there aren't any differences in either option, except for the size limit. In the future, the higher end options will include extra features such as auto partitioning, CLR support, and fanouts. If your database is larger than 10GB you have a few options. Both options are common in the SQL world when a database starts to overwhelm the hardware it is running on. In this case we will use these strategies to work around the size limits that SQL Azure presents us.

13.3.1 - Partitioning your data

The first option you have in dealing with a database that is too large for SQL Azure is to partition your data. Imagine a normal application database schema. There are lots of tables, but all tables can be grouped into families. In our fictional DayOldSushiOnline.com website we might have one set of tables that focus on customer data, a second set on orders and order history, and a third set of product data and pricing.

Figure 13.7 - A sample application database schema for DayOldSushiOnline.com. The tables naturally fall into three families: customer data, orders and order history, and product data and pricing.

Data partitioning is the strategy of breaking your database in a vertical manner, breaking your schema along family lines. In our example this would create three databases, with the names DOSO_customers, DOSO_orders, and DOSO_products. This does tend to break relationships and queries across these families. This alone can represent a significant amount of rework in your application code related to data handling, as well as the stored procedures and queries in your database.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 13.8 - Our schema partitioned into families of tables.

This strategy will then create three smaller databases, each hopefully fitting into the 10GB limit of SQL Azure. This is commonly done with on-premises databases because it allows the infrastructure team that supports the SQL Servers to tune each database, and its storage system, to the needs of that database. In our example, the products database is going to receive a lot of read traffic, with very little write traffic. On the other hand, the Orders database is likely to be a mixture of both, with more emphasis on write operation related performance. This is a great approach for keeping the load on one part of the database from negatively impacting another part of the database.

This strategy can lead to more complex backup and restore operations because you will need to guarantee transactional consistency across the different databases. This isn't that much of a challenge when everything is in the same database.

13.3.2 - Sharding your data for easier scale

I find that partitioning you data is fine, but it becomes troublesome once your system reaches the true scale only the Internet can provide. You might be able to break a family down once more, but you are eventually backed into a corner and have to resort to expensive hardware tricks to continue to scale. This also continues to make your data access related code more and more complex.

Another option is to shard your data. If partitioning is breaking your schema along vertical lines, sharding is breaking your data across horizontal lines. Notice I used the word schema with the partitioning strategy, and the word data with the sharding strategy.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 13.9 - What the DayOldSushiOnline.com databases look like after they have been sharded. We have three identical databases, each with a horizontal slice of the customer data.

When we shard a database, we create several databases. Each new shard has the exact same schema as the original. Then we break the data into chunks, and place each chunk into its own shard. The strategy you use to define the boundaries of your chunks will vary based on the business and growth natures of your system. A simple strategy to start with, but one that will need rebalancing shortly is to break the chunks on the customer name field, as in our example in figure 13.9. Each shard has its own copy of common or static data, this case the product data. This helps performance by providing local replicas of commonly needed data. This data could be partitioned out, but this would result in more complexity in your infrastructure. Our first shard is going to have all of the data that is related to any customer with a last name starting with the letters ranging from A to K. The other two shards will use the letters L and Q, and R and Z respectively. As new customer records are created, they are routed to the appropriate database. This approach can keep your data layer code fairly straightforward, as all of the code is identical regardless of which database the customer is located in. The only code that needs to be added is some logic to look up which db connection string should be used for that customer. This can easily be isolated in your data access layer, and the connection strings can be managed through the configuration system.

As I mentioned above, with this simply strategy your shards can become out of balance. This is when some shards are significantly larger than others, and this leads to a disproportionate use of the resources on each server. There are better strategies for how you might shard your data. One approach is to mix all of the customers up, and just assign each new customer to the latest, smallest database. This in effect shards the database on when the customer account was created, and not by customer name. This makes it easy to create new shards in the future, as the size of each database reaches its effective limit. When the last shard reaches capacity, you dynamically create a new shard, enroll it in the configuration system, and start provisioning new customers to it.

13.4 How does SQL Azure work?

While we use the euphuism that a SQL Azure database is just SQL Server in the sky, that is not entirely accurate. Yes, SQL Azure and Windows Server are involved, but not like you might think. When you connect to SQL Azure server, and its database, you aren't connecting to a physical SQL Server. You are connecting to a simulation of a server. We would use the term 'virtual', but it has nothing to do with Hyper-V or application virtualization.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The endpoint that you connect to with your connection string is a service that is running in the cloud, and it mimics a SQL Server, allowing for all of the TDS and other protocols and behavior you would expect to see when connecting to a SQL server.

This 'virtual' server then uses intelligence to route your commands and requests to the backend infrastructure that is really running SQL Server. This intermediate virtual layer is how the routing works, and how the replication and redundancy is provided, without exposing any of that complexity to the administrator or developer. It is this encapsulation that provides much of the benefit of the Azure platform as a whole, and SQL Azure is no different.

Figure 13.10 - The logical shape of SQL Azure, and how your code will see it. The SQL Azure Server is really just a service that emulates a real SQL Server. Each of your databases are actually spread across a farm of SQL Servers.

This is why SQL commands and features that are related to any of the physical aspects of the infrastructure are not allowed in SQL Azure. This encapsulation layer removes the concern of the physical infrastructure. As a rule of thumb, any command or operation that affects the physical infrastructure is not allowed. When creating a database you can't set where the files will be, or what they will be called, because you don't know any of those details. The services layer manages these details behind the scenes.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 13.11 - How SQL Azure works behind the scenes, encapsulating the physical infrastructure.

The data files that represent your database are stored on the infrastructure as a series of replicas. The SQL Azure fabric controls how many replicas are needed, and creates them when there aren't enough available. There is always one replica that is elected the leader. This is the replica that will receive all of the connections and execute the work. The SQL Azure fabric then makes sure any changes to the data are distributed to the other replicas using a custom replication fabric. If a replica fails for any reason, it is taken out of the pool, and a new one is created on the spot.

When a connection is made the routing engine looks up where the current replica leader is located, and routes the request to the correct server. Because all connections come through the router, the lead replica can change and the requests will be rerouted as needed. The fabric can also move a replica from one server to another for performance reasons, keeping the load smooth an even across the farm of servers that run SQL Azure.

What is really happening behind this encapsulation is really quite exciting. The infrastructure layer contains the physical disks and networks needed to redundantly and reliably store the bits that are part of your database. This is similar to the common SAN that many database server infrastructures use. The redundancy of the disks and the tight coupling of the networks provide both performance and reliability for your data.

Sitting on top of this infrastructure layer is a series of servers. Each server is running a set of management services, SQL Server itself, and the SQL Azure fabric. The SQL Azure fabric is the component that communicates with the other servers in this layer to help them communicate with one another. The fabric provides the replication, load balancing, and failover features for the platform.

On top of the servers is a series of services that manages the connection routing (including the firewall features), billing, and provisioning. This services layer is the layer that you connect with, and the layer that hides all of the magic.

Deep down under all of these covers SQL Server really is running. Microsoft has added these layers to provide an automated and redundant platform that is easily managed and reliable.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

13.5 Managing your database

SQL Azure is heavily automated, and protects you from the concerns of the physical infrastructure. This frees you up to completely focus on managing your database. There are several aspects to managing your database. Many of these aspects do not change, just because the database is in the cloud.

If you ever need for a script to know what version of SQL it is running in you can check the Edition and Engine Edition values of the server. When running in SQL Azure, you will receive "SQL Azure" and "5" respectively.

Usually the first task you have after creating your database is moving your data up into the cloud.

13.5.1 Moving your data

Just because your database is in the cloud, and you don't have to manage the disks doesn't mean you don't have to backup your data. It is true that SQL Azure provides a very robust and reliable platform for your data. It stores you data in multiple copies, and load balances requests across those copies. In some cases this might be enough for you to meet your disaster recovery needs.

But remember, that regardless of the vendor, there is not any service in the cloud that protects you from stupid, no service that makes a bad application work right, and no platform that can fix bad decisions. What it can do is make them stupider, badder, and break faster, for less money. Do not be lulled into complacency with the scale and redundancy the cloud gives you. You still need to think of the risks to your system, and to your data, and plan for them. The redundancy protects you against catastrophic loss. But not all loss is catastrprophic in nature. Sometimes it is just a poorly written update SQL script that renames all of your customers to Terrance. This means you still need to backup your data, to protect your data from yourself and your code.

This won't protect against the disaster of the datacenter getting hit by a falling satellite, a radioactive monster, or a massive power outage. In these cases, having your bits on several servers in the same data center won't help you. Only you, and the business you support, can determine what level of disaster recovery you need.

Disaster is not your only risk. What if an upgrade goes awry, and you accidently delete a customer's data in your system? Or if an automated job takes a left turn and wipes out the order history for the past month? SQL Azure protects you against faults in the platform, but it doesn't protect you against faults in your own code or policies.

Even if you don't need to backup your data, you will likely need to move data at some point. You will run into this when you are migrating an existing on-premises database to the cloud, or when you want to move the data from SQL Azure to another location.

Right now there are only three ways to work with SQL Azure data. The first option would be to use a developer's API to access the data and save it off in some format that you can backup and restore with. There are a lot of options in this case; you could use ADO.NET, ODBC, or ADO.NET Data Services. This is tedious and breaks my rule of not writing any plumbing code. I don't know about you, but I don't want to have to write my own backup tool. You can do this if you want, but it would be tedious work.

The second option would be to use BCP, which is the command for bulk copy in SQL Server. It is one of your best bets. BCP stands for Bulk Copy Program and is used quite often to move large amounts of data into and out of a SQL Server. When using BCP it is common for the DBA to disable indexing until the import is completed to speed up the transfer process. Once the data is loaded, the indexes are enabled, and the indexes and updated based on the new data.

To extract the data from your local SQL Server, you will need to run BCP in a command prompt. The command is really quite simple.

```
bcp databasename.schema.tablename out filename.dat -S servername -U username -P password -n
-q
```

When exporting data with BCP you need to use the `out` keyword. The command is straightforward. Provide the fully qualified tablename, the name of the file to write the data to, the server name, and your username and password. The parameters tell BCP to keep the SQL data types native (don't convert them), and to use quoted identities in the connection string. Depending on the size of your table this might take a few minutes. Once it is

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

done you can open the data file and see what it looks like. Once you are done poking around we can use BCP in the other direction to blast the data into your SQL Azure db. We will use a similar command.

BCP does operate at a table level, so you will need to run it several times, once for each table. A common trick is to write a script that will export all of the tables, one at a time, so you don't have to do it manually. This also helps with automating the procedure, reducing the risk of mistakes, and making it easy to reproduce the process at any time.

Your last option with working with data you have stored, or want to store in SQL Azure is SQL Server Integration Services, or SSIS. SSIS is SQL Servers platform for performing extraction, transforming, and loading of data. It is a common tool used to move data in and out of on-premises SQL servers, and can connect to just about any data source you need it to. You can migrate your existing packages to SQL Azure. Please note that they have to be SQL Server 2008 packages, you cannot run packages from SQL Server 2005.

13.5.2 Controlling access to your data with the firewall

Before an incoming connection will be routed to the current leader server for your replicas, the source IP is checked against a list of allowed sources. If the source IP is not in this list the connection is denied.

The list of approved sources is a true white list, meaning that the IP must be on the list to be allowed in. There are no other rules supported, just the list of allowed IPs. The list is stored in the master db for your SQL Azure database server.

The most common way to adjust the firewall settings is the one showed above, namely by using the SQL Azure portal, and making the changes through the admin pages. You can also add entries directly into the master db by using the stored procedures provided. The rules are stored in a table called `firewall_rules`, and you can query this like a normal table. Since the table is in the master db, the connection to the database performing the query must be under the system administrator account you setup.

	id	name	start_ip_addr...	end_ip_addr...	create_date	modify_date
1	3	BP Home	99.17.241.139	99.17.241.139	2009-10-17 17:03:38.490	2009-10-17 17:03:38.490
2	1	MicrosoftServices	0.0.0.0	0.0.0.0	2009-10-16 03:23:15.190	2009-10-16 03:23:15.190

Figure 13.12 - You can see what firewall rules are being enforced with 'Select * from sys.firewall_rules'.

To manage your firewall rules through code, you would create a connection to the master db with your administrator account, and use the provided stored procedures. `sp_set_firewall_rule` will create a firewall rule, and `sp_delete_firewall_rule` will remove a rule. When creating a rule you will need to provide a unique name, and the two IP addresses that form the range of addresses allowed. These addresses can be the same if you want to grant access to a single IP address. When deleting a rule you only need to provide the name of the rule to be removed.

Even applications running in Windows Azure can't connect to your server until you grant them permissions to connect. Since application servers in Azure use virtual IPs, and the roles could shut down and restart anywhere, causing their IPs to change, there is a special rule you can use in the firewall. If you create a rule with 0.0.0.0 as the starting and ending addresses, then any Azure application will be able to connect to your server. That connection must still provide a valid username and password.

A common problem is to make a change to the firewall and then to immediately try to access the server from that IP, which then fails. The cache that is used to speed up firewall rule checks refreshes every five minutes. If you make a rule change, it is worth waiting up to five minutes before trying to see if the change works.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

13.5.3 Creating user accounts

Managing user accounts in SQL Azure is just like in on-premises SQL Servers. There are a few small differences, mostly with the names of specific roles, but in general it is the same. Your first step in granting someone access to a database is to create a login for the server itself. You need to connect to the master db in your server, with an account with the `loginmanager` role. This would normally be your administrator account you created when you created your server, but any other account with the role will work. Once connected, you can list the existing logins by executing the `sys.sql_logins` view. To create a login, simply use the same command you are used to.

```
CREATE LOGIN theloginname WITH password='strongpasswordhere';
```

Figure 13.13 - Creating a new SQL Azure login is just like with SQL Server on-premises, using the CREATE LOGIN command.

There are two roles that you can assign to a login in the master db. The first has already been mentioned, the `loginmanager` role. This role allows the user to manage user accounts and logins on the server. The second is the `dbmanager` role. Granting this role to a user account gives it all the possible permissions it could have, making it equivalent to the administrative account you first created.

Once you have created a server login, you can create a user account in a specific db based on that login. This is the same approach as in on-premises SQL Server. While you can easily do this in SSMS, many dba's like to do this with sql commands. You would execute the following command while connected to the database you want to create the user account in.

```
CREATE USER newusername FROM LOGIN theloginname;
```

Once the user is created in the db, you can grant and revoke privileges like a normal SQL database, either with the admin tools, or with the GRANT and REVOKE commands.

13.6 - SQL Azure versus Azure Tables

SQL Azure and Azure Tables have some very significant differences, which we have tried to cover in this chapter, and the series of chapters on Tables. These differences help make it a little easier to pick between SQL Azure and Tables, and the deciding factor usually comes down to if you have a database already or not. If you do, and you want to keep using it, then use SQL Azure. If moving it to the cloud would cause you to have to refactor some of the schema to support partitioning or sharding, then you might want to consider some options.

If size is the issue, that would be the first flag that you might want to consider Azure Tables. As long as the support Tables has for transactions and queries meets your needs. The size limit surely will, at 100TB.

If you are staying with SQL, and are going to upgrade your db schema to be able to shard or partition, take a moment to think about also upgrading it to support multi-tenant scenarios. If you have several copies of this

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

database, one for each customer that uses the system, now would be a good time to add in support needed to run those different customers on one database, but still in an isolated manner.

If you are building a new system that doesn't need sophisticated transactions, or a complex authorization model, then using Azure Tables is probably best. People tend to fall into two groups when they think of Tables. They are either from 'ye olde country' and think of Tables as a simple data storage facility that will only be used for large lookup tables and very flat data or they are able to see the amazing power a flexible schema model and distributed scale can give them. Looking at Tables without the old blinders on is challenging. We have been beaten over the head with relational databases for decades, and it is hard to consider something that deviates from that expected model. The Windows Azure platform does a good job of providing a platform that we are familiar and comfortable with, while at the same time giving us access to the new paradigms that make the cloud so compelling and powerful.

13.7 - Migrating an application to SQL Azure

One of the goals Microsoft had in delivering SQL Azure was to make it as easy as possible to migrate an existing application, and its database to their Azure platform. Before SQL Azure was released people were worried they would have to re-architect their systems in order for them to run in the cloud.

The easiest way to migrate a database to SQL Azure, and one that I have used myself, is to just find a DBA that already knows how to do it and buy them lunch. Barring that, there are a few steps you can go through.

The first step is to make sure that you have created your SQL Azure server, and know the administrative username and password. Once that is done you can create the database that will house your schema and data. Our goal will be to script out the existing database, including both schema and data, and then run that script against your cloud server database to recreate it all.

We will use SQL Server Management Studio to script out our source database. Navigate to the database in SSMS, and right click on it. Select **Generate Scripts** from the **Tasks** menu. Walk through the wizard like normal, but we will have some special settings to avoid some of the commands that SQL Azure doesn't support. Make sure that you select the option to script all objects in the database. On the options screen, make your selections according to this list.

- Convert UDDTs to Base Types -> True
- Script Extended Properties -> False
- Script Logins -> False
- Script USE DATABASE -> False
- Script Data -> True

Finish the wizard. When you are done the entire script will be loaded in a window. We now need to remove some parts of the script to make sure it will work in the cloud.

- Delete any occurrence of `SET ANSI_NULLS ON`.
- Delete any occurrence of `WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]` in a CREATE TABLE command.
- Delete any occurrence of `ON [PRIMARY]` in a CREATE TABLE command.

Once you have made these changes to the script you may connect to the destination database and execute the script. You can use SQLCMD for this, or SSMS.

13.8 - Limitations

While SQL Azure is based on SQL Server, there are some differences and limitations that you will need to be aware of. We have run into some of these as we have moved through this chapter, but we will try to cover them all in this section.

The most common reason for any limitation is the services layer that sits on top of the real SQL Servers and simulates a SQL Server to the consumer. This abstraction away from the physical implementation is usually the

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

cause, or the routing engine itself. For example, you cannot use the USE command in any of your scripts. This is because the routing layer is stateful, and because the TDS protocol is a session based protocol. You connect to a server, create a session, and then execute commands over it. When we connect in SQL Azure, the fabric routes our commands to the SQL Server that is hosting the leader of our data replicas. If we call the USE command, that database may not be on the same physical server as the database we are switching from. To avoid this problem the USE command is not allowed.

To get around this limitation you will need to make a separate connecting for each different database you want to connect with. You should just assume that each of your databases are on different servers.

Any T-SQL command that refers to the physical infrastructure is also not supported. For example, some of the CREATE DATABASE options that can configure which `filegroup` will be used aren't supported, because as a SQL Azure user, you don't know where the files will be stored, or even how they will be named. Some commands are outright not supported, like BACKUP.

You can only connect to SQL Azure over port 1433. You cannot reconfigure the servers to receive connections over any other port or port range.

You can use transactions with SQL Azure, but you cannot use distributed transactions. These are transactions that enroll several different systems into one transaction update. SQL Azure does not support the network ports that are required to allow this to happen. Be aware that if you are using .NET 2.0 that a normal transaction may be elevated to a distributed transaction in some cases. This will cause an error, and you won't know where it is coming from.

Each table in your database schema must have a clustered index. Heap tables are not supported (a fancy DBA term for a table without an index). If you import a table without a clustered index you will not be able to insert records into that table until one has been created.

All commands and queries must execute within five minutes. At this time that is the system wide timeout. Any request taking longer than that will be cancelled, and an error code will be returned. This limit might change in the future, as Microsoft tunes the system to their customer's needs.

There are some limitations that are very niche in nature, and more commands are supported with each new release. Please read the appropriate MSDN documentation on the SQL Azure limitations to have the most recent list.

13.9 - Common Scenarios

There are several common scenarios with how people are using SQL Azure in their applications. They generally fall into two categories, 'near data' and 'far data.' These terms refer to how far away the code that is calling into the SQL server is. If it is creating the connection over what might be a local network (or even close with named pipes or shared memory) then that is a 'near data' scenario. If the code opening the connection is anywhere else, then that is a 'far data' scenario.

13.9.1 - Far Data Scenarios

The most common far data scenario is when I am running my application, perhaps a web application, in an on-premises data center, but are hosting the data in SQL Server. This is a good choice if you are slowly migrating to the cloud, of if you want to leverage the amazing high availability and scale SQL Azure has to offer without spending $250,000 yourself.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 13.14 - A web server using SQL Azure in a far data scenario. The data is far away from the code that is using it. In this case the web server is on-premises, and the data is in the cloud with SQL Server.

In a far data scenario the client doesn't have to be a web browser over the Internet. It might be a desktop WPF application in the same building as the web server, or any other number of scenarios. The one real drawback to far data is the processing time and latency of not being right next to the data. In data intensive applications this would be a critical flaw, while in others it is no big deal.

Far data works really well when the data in the far server doesn't need to be accessed in real time. Perhaps you are offloading your data to the cloud as long term storage, and the real processing happens on site. Or perhaps you are trying to place the data where it can easily be accessed by many different types of clients, including mobile public devices, web clients, desktop clients, and the like.

A near data scenario would be doing calculations on the SQL server directly, or executing a report on the server directly. The code using the data is running very close to the data. This is why the SQL team added the ability to run managed code (with the CLR support) into the on-premises version of SQL Server. This feature is not yet available in SQL Azure.

Figure 13.15 - Hosting a data service in an Azure web role helps your application be in a 'near data' scenario. This improves the performance of the application when it comes to working with the data.

One way to convert a far data application to near data is to move the part of the application accessing the code as close to the data server as possible. With SQL Azure this means creating a services tier and running that

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

in a role in Azure. Your clients can still be web browsers, mobile devices, and PCs, but they will call into this data service to get the data. This data service will then call into the SQL Server. This encapsulates the use of SQL Azure, and helps you provide an extra layer of logic and security in the mix.

13.10 - Summary

SQL Azure is a powerful data platform that is familiar to us. It just happens to be running in the cloud. This makes it easier to move an application to the cloud, or to build something new, using paradigms and tools that are know and love.

We looked at how to create and manage a database in the cloud. The processes and approaches are eerily similar to what you would do when working with a local database. This makes it easy to adopt SQL Azure for your application. There are some limitations because of the 'virtual' nature of the SQL Server, and because the db is running in a shared environment.

The sophisticated data engine that is SQL Azure isn't the last stop for SQL in the cloud, it is just the beginning. The tools will be upgraded for full support, and the rest of the SQL family will move into the cloud over time.

While the firewall and other security features make it easier to trust putting your data in the cloud, you still need to use critical thought as to if it is the right place for your data, or if a blended approach is best for your scenario.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

14

Working with different types of data

This chapter covers:

- Working with static data
- Working with dynamic data
- Working with infrequently changing data

In previous chapters we've shown you what is possible with The Table Service, SQL Azure and Caching. In this chapter we will look at how you can use these technologies to make the correct choices of when to use which technology. Rather than focusing on the technology in the chapter we will look at the type of data that you can store and how each technology will help you store said data.

In this chapter we will look at 3 types of data and how we can store and retrieve that data, particular:

- Static data
- Infrequently changing data
- Dynamic data

In this chapter we will also focus on how where to store the data and how to efficiently retrieve the data. We will be looking at the Table Service, SQL Azure and Caching technologies. We'll start with the first type of data, static data.

14.1 Static Reference Data

Every application typically has some sort of frequently accessed static reference data. This data is usually very small in size and typically used for data normalization purposes. Let's return to our Hawaiian Shirts Website and look at how we might normally represent some data describing our shirt products.

For each shirt that we display in our Hawaiian Shirt Shop web page we may wish our customer to be able to specify the following criteria about the shirt they buy:

- Shirt Personage Type (Men's, Ladies, Boys, Girls)
- Shirt Size (small, medium, large, extra large)
- Shirt Material (cotton, silk, wool)

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

As we can see from the list above, the data listed is fairly static in nature and is commonly applied across all shirts. Regardless of the shirt sold, it would have a size or it would have a material (admittedly not wool). This data can be considered static, once defined it is very unlikely that it would ever be changed (Hawaiian shirts are unlikely to suddenly to start being made in platinum).

Figure 14.1 shows a page of our website where that data can be used. In this instance the customer can browse shirts that are designed for Men or Ladies, and for a particular shirt they can choose the size or material.

Figure 14.1 Product Detail Page of the Hawaiian Shirt Website

As you can see from Figure 14.1 our web page represents the static data Style and Size as dropdowns, whereas the Personage Type is represented as a hyperlink that will perform a search of different shirt types. For just now we will focus just on the two dropdowns (style and size).

So the first question that you are probably asking, is where and how do we represent this data? Well let's take a look at how this could be done using:

- SQL Azure
- Table Service
- Cache

We'll start with SQL Azure, as this is probably the most familiar way of representing data.

14.1.1 Representing Simple Static Data in SQL Azure

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

As you have learned from our earlier chapters, SQL Azure is a relational database, so the design that we would come up with, would be a typical relational model. Figure 14.2 shows the database diagram for our shirt website in SQL Azure.

Figure 14.2 - Shirts, SizeTypes and Materials database in SQL Azure with no relationships defined.

From Figure 14.2, we can see that each the data for each of our dropdowns (SizeTypes and Materials) are currently stored in their own table. As of yet, we have not defined any relationships between the static tables and the shirts table (that stores details of our Hawaiian shirts).

Now that we have defined the tables, let's take a look at how we retrieve the data from our database and populate the dropdowns on our page.

POPULATING THE DROPDOWN LISTS

If we wish to populate the Materials or SizeTypes dropdown directly from a database then we can just make standard ADO.NET call to the database (either using ADO.NET directly or using your favorite Data Access Layer technology e.g. Linq2SQL, ADO.NET Entity Framework, NHibernate etc). The following code shows how to bind the size dropdown using ADO.NET directly.

```
DataSet ds = new DataSet();
using (SqlConnection conn =
   new SqlConnection(mySqlAzureDBConnectionString))
{
   conn.Open();

   using (SqlDataAdapter da =
      new SqlDataAdapter("SELECT SizeId, SizeName FROM SizeTypes",
          conn))
   {
      da.Fill(ds);
   }
}

sizeDropDown.DataSource = ds.Tables[0];
sizeDropDown.DataBind();
```

The previous piece of code is really there just to illustrate that you can bind your dropdown lists to a table in SQL Azure just as easily as you can with a regular SQL Server database.

NOT PRODUCTION LEVEL CODE

Obviously the listed code is not production level and you should not mix data access code with presentation layer code however it does illustrate the point.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Cost Issues with SQL Azure

Although SQL Azure is a fixed price model, if your database is tied up servicing static data calls (which always return the same set of data), then you may hit the limits of your database very quickly unnecessarily, and require you to scale out just to meet the demand. In this instance caching the data is probably the most cost efficient approach.

As stated earlier, SQL Azure is not the only method of storing static data, so let's look at how we can store this data using the Table Service.

14.1.2 Representing the Simple Static Data in the Table Service

Just as easily as each type of static data could be represented in a SQL Azure table, each static table could simply be represented as entity stored in a The Table Service Service table. The following C# class would represent our Size Type entity in The Table Service.

```
public class SizeType : TableServiceEntity
{
   public string SizeCode { get; set; }
   public string Description { get; set; }
}
```

In the above entity, the PartitionKey for our SizeType entity is not really relevant due to the size of the table. In this instance and the RowKey would be represented by the size type name (small, medium, large, extra large).

Surrogate Keys

Since The Table Service is not a relational database and we do not need to considered clustered indexes we have no need for the surrogate key SizeTypeId that is present in the SQL Azure implementation.

CONTEXT CLASS

The following code represents the data service context class for our Size Type table.

```
public class SizeTypeContext : TableServiceContext
{
  private static CloudStorageAccount storageAccount =
  CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

  public ProductContext()
     : base(storageAccount.TableEndpoint.ToString(),
            storageAccount.Credentials)
   {

   }

  public DataServiceQuery<SizeType> SizeTypeTable
  {
     get
     {
        return CreateQuery<SizeType>("SizeTypeTable");
     }
  }
}
```

For a detailed explanation of the context class then you should refer to the Table Service chapters.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

STORING A SIZE TYPE IN THE TABLE

To store a size in the SizeTypeTable table in The Table Service we could use the following code:

```
var sizeTypeContext = new SizeTypeContext();

var newSizeType = new SizeType
{
          PartitionKey = "SizeTypes",
          RowKey = "Small",
          SizeCode = "S",
          Description = "A shirt for smallish people"
};

sizeTypeContext.AddObject("SizeTypeTable", newSizeType);
sizeTypeContext.SaveChanges();
```

The code above will store the "Small" size entity in the Size Type Table. The method used to store the data in the table is explained in more detail in the The Table Service chapter earlier on in the book.

> **Partitioning**
>
> In this particular example our Sizes table will never grow beyond a few rows. It is therefore not worth splitting the table across multiple servers. Since the data will always be returned together we will store the data in a single partition called SizeTypes.

POPULATING THE DROP DOWN LIST

With a fully populated Size Type table then you could bind the Size Type table to the dropdown list using the following code:

```
var sizeTypeContext = new SizeTypeContext ();
sizeDropDown.DataSource = sizeTypeContext.SizeTypeTable;
sizeDropDown.DataBind();
```

In the above example you can see that we are populating our dropdown directly from The Table Service.

> **Cost Issues with the Table Service**
>
> Since you are charged for each request that you make to The Table Service, if you can reduce the requests your monthly bill will obviously cheaper. In a website where you receive 1000 page views of the Product Details page then this would translate to 4000 the Table Service requests. Based on these figures your hosting bill could get very costly, very quickly following this model. Caching the data is probably the most efficient thing to do in this instance

We've already talked a little about the performance disadvantage of using SQL Azure or the Table Service for accessing static data. Let's look at this in more details especially on why you should avoid chatty applications in Windows Azure.

14.1.3 Performance disadvantages of a chatty interfaces

In the previous 2 sections we discussed how you could store static data such as Size Types and Shirt Material Types in either SQL Azure or The Table Service. We will now look at the call sequences we would need to make to render the web page shown in Figure 14.1, and discuss the pro's and cons.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

SYNCHRONOUS CALLS

In our Hawaiian Shirts product details page shown in Figure 14.1 to retrieve all the data required to display the page we would need to make at least 4 calls to our storage provider as we would need to:

- Retrieve the Product Details
- Retrieve the Size Types List
- Retrieve the Style List (Materials)
- Retrieve the Personage Types List (Men's, Ladies)

When developing an ASP.NET Web Page although we should consider making asynchronous calls to improve performance, most developers will typically write synchronous calls to retrieve the data. Figure 14.3 shows the synchronous call sequence for our web page:

Figure 14.3 Synchronous Call Sequence, each result must be returned before we can make the next call

As you can see from Figure 14.3 due to the synchronous nature of the page we have to wait until we retrieve data back from the Data Provider (SQL Azure DB or The Table Service) before we can process the next section of the page. Due to the blocking nature of synchronous calls and the latency involved between cross server communication the rendering of this page would be much slower than it needs to be.

ASYNCHRONOUS CALLS

Since Size Types, Styles (Materials), and Personage Types are sets of data that are both independent from each other and independent from the returned product details, we could use asynchronous calls instead. Retrieving the data from the Table Service asynchronously allows us to retrieve data from the data storage provider without having to wait for the data to be returned before continuing onto the next statement in our web page. Figure 14.4 shows the same call sequence for our web page but this time using Asynchronous calls.

r corrections to the Author Online forum:
mID=579

Figure 14.4 Asynchronous Call Sequence

As we can see from figure 14.4 we no longer have to wait for data to be returned before we process the next statement. Once all data is returned we can then render the page.

> **Asynchronous calls are relevant whenever data is not related**
>
> In this instance we have used the static data calls as our example. We can however use asynchronous calls whenever we don't have any relationship between sets of data rendered on a page.

So now that we understand how we can store and retrieve our static data, let's take a look at how we can improve our performance (and hosting bill) by using cached data instead.

14.1.4 Caching Static Data

For cost and performance reasons regardless of the chosen storage platform (SQL Azure or The Table Service) you should consider caching the static data (Size Types, Materials and Personage Types) rather than continually retrieving the same data from the table will bring large performance and cost benefits.

Since static reference data hardly ever changes and is usually a pretty small set of data, the in-process memory cache (such as the default ASP.NET cache) would be a suitable caching choice. You can always use a Cache Dependency to keep the cache and the underlying backing store synchronized. For more details on how to setup an in-process cache please see the "Scaling Web Roles" chapter.

Let's now take a look at how we can use the cache in our Hawaiian shirts website.

POPULATING THE CACHE

For frequently accessed static data you should probably populate the web role cache on startup of the application. The following code shows how this is done:

```
protected void Application_Start(object sender, EventArgs e)
{
   var sizeTypeContext = new SizeTypeContext;
   Cache["sizeTypes"] = sizeTypeContext.SizeTypeTable;
}
```

In the above example if you wish to populate the Size Types when your web role starts up you can simply populate the cache with the static data. In this example we have populate the data from the Table Service, however this technique is the same also for SQL Azure.

POPULATING THE DROPDOWNS

Now that we have our data in the cache, let's take a look at how we can populate our dropdowns with the data from the cache, as shown below:

```
sizeDropDown.DataSource = (IEnumerable<SizeType>)Cache["sizeTypes" ];
sizeDropDown.DataBind();
```

As you can see from the code above we no longer need to return to the data store to populate the dropdown and can use the cache directly. Since the cache will be scavenged when memory is scarce you can always give static data higher priority that other cache items using the cache priority mechanism (meaning other items will be scavenged first):

```
var sizeTypeContext = new SizeTypeContext;
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
HttpContext.Current.Cache.Insert("sizeTypes",  sizeTypeContext.SizeTypeTable,  null,
new DateTime(2019, 1, 1), Cache.NoSlidingExpiration, CacheItemPriority.High, null);
```

In the code above our SizeTypes list will be stored in the cache with a High priority and has an absolute expiration of the 1st of January 2019. If the web role is restarted at anytime then the cache will still be flushed but as long as the process is running this data should remain in memory until this date. If the static data may change in the future then you can always set a cache dependency to keep the cache synchronized or manually restart the role when updating the data.

PROTECTING YOUR CODE FROM AN EMPTY CACHE

Since cache data is volatile you may wish to protect your application from the cached data being flushed by checking that the data is cached prior to populating the dropdown from the cache e.g.:

```
private IEnumerable<SizeType> GetSizeTypes()
{
   if (Cache["sizeTypes"] == null)
   {
      var sizeTypeContext = new SizeTypeContext;
   Cache["sizeTypes"] = sizeTypeContext.SizeTypeTable;
   }

   return Cache["sizeTypes"];
}

sizeDropDown.DataSource = GetSizeTypes();
sizeDropDown.DataBind();
```

In the code above, before we populate the dropdown list we just check if the data is already stored in Cache. If the data is not held in cache we will then go back to our backing store and repopulate that cache item.

SUMMARY

By effectively caching static data we can both reduce our hosting bill and improve the performance of our application. The use of our in-memory cache for static data means that for our Product Details page we now have 1 data storage call per page view rather than 4 data storage calls per request.

The use of an in-memory cache for static data will also mean that we do not need to consider the retrieval of this data when where to store our underlying data.

Please be aware that the examples in this section are no production level code and have been simplified to illustrate the concepts. If you are implementing such a solution you should also take the following items into consideration:

- Abstract your caching code into a separate caching layer
- Do not use magic strings (e.g. Cache["sizeTypes"], use constants instead)
- Use cache dependencies
- Prioritize your cache properly
- Check your cache is populated prior to returning data
- Handle exceptions effectively

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

14.2 Storing static reference data with dynamic data

In the previous section we looked at how we can represent static data for the purposes of data retrieval. This is really only one side of the picture as typically we will want to associate the static data with some dynamic data.

For example in the Product Details web page shown in Figure 14.1 we will eventually purchase the shirt displayed on the page. The real question in this instance is how should we store that data so that we can easily retrieve it?

Depending on the implementation of the web page, you can either allow the user to purchase the item directly or simply add the item to a shopping cart basket. In either case the method of storing the data will be the same, so we will look at the storage of items within a shopping cart

14.2.1 Representing the shopping cart in SQL Azure

Since most of us are probably more familiar with relational databases we will first look at how we can store this data in SQL Azure prior to looking at the Table Service. Our shopping cart can be persisted across sessions and if the user is a registered logged in user then we will associate this account with their user id. Figure 14.5 represents a typical data model for our shopping cart.

Figure 14.5 Data Model for a shopping cart

In figure 14.5 we represent the shopping cart as two tables (ShoppingCart and ShoppingCartItems). The ShoppingCart table represents the shopping cart for each user and each item of the shopping cart is stored in the ShoppingCartItems table. For each item of the shopping, the ShirtId, MaterialId and SizeId is stored and the appropriate foreign key relationships between tables are established.

If the website user is a registered user then the UserId would be stored in ShoppingCart table and if the user is unregistered we could always use the session id.

Although we are choosing only to represent a shopping cart at this stage, if you wished to model an orders table the data structure used would be very similar to the shopping cart version.

RETRIEVING THE DATA

Since SQL Azure is a relational database if we need to query across multiple tables we can always use JOIN clauses in our SQL statements, if required. For example to return all items in the Shopping Cart for UserId 12345 including the shirt name, size and material, we could issue the following SQL query:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
SELECT sci.ShoppingCartItemId,
        s.ShirtName,
       s.Description,
       s.Price,
       sz.SizeId,
       sz.SizeName,
       m.MaterialId,
       m.MaterialName
FROM ShoppingCart sc
JOIN ShoppingCartItems sci ON sci.ShoppingCartId = sc.ShoppingCartId
JOIN Shirts s ON s.ShirtId = sci.ShirtId
JOIN SizeTypes sz ON sz.SizeId = sci.SizeId
JOIN Materials m ON m.MaterialId = sci.MaterialId
WHERE sc.UserId = "12345"
```

USING THE CACHE TO IMPROVE THE QUERY

Since the shopping cart table will never hold a large amount of data, and our static data is held in an in-memory cache, there is no need to return the static data as part of our SQL Query. We can therefore use the following SQL Query instead:

```
SELECT sci.ShoppingCartItemId,
       s.ShirtName,
       s.Description,
       s.Price,
       sci.SizeId,
       sci.MaterialId,
FROM ShoppingCart sc
JOIN ShoppingCartItems sci
   ON sci.ShoppingCartId = sc.ShoppingCartId
JOIN Shirts s ON s.ShirtId = sci.ShirtId
WHERE sc.UserId = "12345"
```

By not returning the static data (SizeTypes and Materials) we have both improved the performance and reduced the complexity of the query by reducing the number of joins.

Once the data is returned from the database we can combine the data returned from the SQL Azure with our in-memory cached data using LINQ2Objects.

PERFORMANCE NOTE

This strategy is perfectly acceptable when dealing with static reference data as the cached data tends to be very small. This would not be an acceptable strategy when dealing with data that contains millions of rows.

14.2.2 Partitioning the SQL Azure Shopping Cart

The problem with the data model that we designed in Figure 14.5 is that it looks like that it is only usable for single server databases. Actually this is not strictly the case as the model we have designed will easily scale out as I will explain in a second.

One of the issues with using SQL Azure is that currently there is no built-in method of partitioning data across multiple servers. Therefore to avoid bottlenecking our application on a single database we need to partition (or shard) our data across multiple servers in our application layer. In Chapter 13, we spent a little

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

time talking about sharding already but this time we'll look at the application of it in relation to the Hawaiian Shirt Shop.

SPLITTING THE DATA MODEL ACROSS MULTIPLE SERVERS

As I said earlier the data model that we designed earlier will work across multiple servers, we just have to be a little smart about how we do this. Since our shopping cart tables have no dependencies on any other part of the application, we can separate these tables into its own separate database. In our application layer we would therefore separate the shopping cart data access layer methods and connection strings into their own layer. By keeping the functionality logically separated we can split our application across multiple databases (where appropriate) if required. In our Hawaiian Shirts example we could easily maintain separate databases for our Shopping Cart, Orders, Customers, Products and Static data if required.

> **Wow all those databases, is that not expensive?**
>
> The good news about this type of design is that we can still start with a single SQL Azure when our application still has a small amount of users. As the traffic increases on our system we can then decide to split the shopping cart tables onto their own separate database when needed. To split out our databases we could simply creating a new SQL Azure, migrate the shopping cart data across and then change the connection string for the Shopping Cart data layer.

WHAT IF I NEED TO PARTITION MY DATA FURTHER?

In your web application you may get to the point that the shopping cart database is still bottlenecking and you ideally need to partition the data further. Actually this is not as difficult as it sounds ☺

Since the shopping cart data is really only used by a single user and we do not query across multiple users, we can easily partition out the data by user. In our database we may have 100,000 registered users that maintain a shopping cart. If we currently have a single database and wish to split that database into 2 or more databases we could use a partitioning function in our application. For example:

```
If (userId.ToString()[0] > 'N')
{
   // Use Connection String for Database 1
}
else
{
  // User Database 1
}
switch (userId.ToString().[0])
{
   // Use Connection String for Database 2
}
```

In the above example if the first character of the userId begins with N then we would our second shopping cart database otherwise we would use the first database. If required we can obviously break this down into further partitions if required.

As stated earlier this type of partitioning can only be done in the application layer at present, in future releases of SQL Azure then it is planned that there will be some partitioning built-in server side.

> **How do I maintain Referential Integrity**
>
> One of the issues with splitting data across multiple servers is keeping referential integrity. In the shopping cart example the data that the shopping cart is referencing is either static (SizeTypes, Materials or Personage) or Infrequently Changing (and administratively controlled) so referential integrity is not such a big deal as we

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

are ultimately controlling the data. Therefore we can break our foreign key relationships and store the shopping cart database and store the tables on a separate server.

If you do need to maintain referential integrity then you could keep a copy of the static data on each instance of the database. Although there is data duplication this is such a small amount of data that it's not such a big deal to synchronize the databases. In this case you would tend to keep one master database allowing one-way synchronization.

In the shopping cart example, since we have no need to query across multiple tables (beyond the cache) then we can safely break referential integrity.

What if I need to query across partitions?

Unfortunately there is no real way of efficiently querying across multiple databases in the current implementation of SQL Azure hence why you can only really partition functionally independent data.

However if you only need to query across partitions when reporting (i.e. all customer orders in the past week) then you can always export data out of your SQL Azure real time database into a large reporting database (outside of Windows Azure) where you can make use of a full-blown version of SQL Server with BI capabilities

In this example we have shown you how you can use SQL Azure with Windows Azure when storing dynamic data with static data (and partition where necessary). We have purposely not looked at how this works with large sets of infrequently changing data as we will look at this slightly later on in this chapter

14.2.3 Representing static data for the shopping cart in The Table Service

In the previous section we looked at how we could store our shopping cart in SQL Azure and how we could scale it out horizontally if necessary. Although it is possible to scale out a SQL Azure it still requires some manual partitioning code at the application layer. We will now look at how we can represent the Shopping Cart table in The Table Service instead and how we can just use the built-in partitioning model to scale out our tables.

Due to the architecture of the Table Service we have no facility to perform a server side join between a Shirts table and the Shopping Cart table. This effectively leaves us 2 options when have data that resides across 2 different tables, either:

- Duplicate Data
- Join the data client side

For just now, we will ignore the duplicate data option (we will cover that in the next section), and focus on client side data joining. Before we looking at joining the data client side, let's take a peek at how we would represent our shopping cart table in the Table Service.

SHOPPING CART ENTITY

In the SQL Azure implementation of our Shopping Cart data model we had 2 tables (ShoppingCart and ShoppingCartItems). Since The Table Service is not a relational database and does not support joining between tables, we will represent our Shopping Cart as a single table (ShoppingCart). Within the ShoppingCart table we will store the entity as follows:

```
public class ShoppingCartItem : TableServiceEntity
{
   public string Shirt {get;set;}
   public string Material { get; set; }
   public string Size { get; set; }
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
}
```

If we look at our definition of our ShoppingCartItem entity, because the both the material and size data are cached in-memory, we can merely store a reference to the data (i.e. the RowKey of the Material and Size) and then perform a client side join between our shopping cart entity and our cached versions of the Material and SizeType entities.

Partitioning The Table Service

Unlike SQL Azure partitioning our The Table Service implementation of the Shopping Cart is pretty simple. All we really need to do is set a reasonable value for the PartitionKey for our shopping cart table and Windows Azure The Table Service will take care of the rest. In this case we will only be retrieving the shopping cart items of an individual user at any one time, so it would make sense to partition our data by UserId. Therefore by setting the PartitionKey as our UserId then our data can be physically partitioned out to as many servers as necessary but the data for a single user's shopping cart will always physically reside together. For more details on how The Table Service Partition works please refer back to the The Table Service Chapter.

Because our cache data is a small set of static reference data that is held in-memory and is being joined to a very small set of shopping cart data, this technique is very appropriate. To do that we will define a new entity that will represent a strong version of our Shopping Cart Item as shown below:

```
public class StrongShoppingCartItem
{
   public Shirt SelectedShirt { get; set; }
   public SizeType Size { get; set; }
   public Material Material { get; set; }
}
```

As we can see from the code above, this time we have represented our shopping cart item with a reference to each entity rather than the using an Id reference. Now that we have the stronger version of the entity, we need to populate it, as shown below.

```
var materials = (IEnumerable<Material>)Cache["materials"];
var sizeTypes = (IEnumerable<SizeType>)Cache["sizeTypes"];

var shoppingCartItemContext = new ShoppingCartItemContext ();

var shoppingCartItems =
   shoppingCartItemContext.ShoppingCartItem.ToList();

var q = from shoppingCartItem in shoppingCartItems
        join sizeType in sizeTypes
           on shoppingCartItem.Size equals sizeType.RowKey
        join material in materials
           on shoppingCartItem.Material equals material.RowKey
        select new StrongShoppingCartItem
        {
           SelectedShirt = new Shirt(),
           Material = material,
           Size = sizeType
        };
```

The cool thing about the above query is that because the size type and materials are cached, we make no extra table service calls to get our reference data. This is all performed in-memory using LINQ2Objects.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

PERFORMANCE WARNING

This technique is super cool for small datasets, just check your performance on large datasets (e.g. millions of rows).

In-memory joins for static data saves money

Not only will In-memory joins improve the performance of your application, it will save you lots of cash. Ultimately the less calls you make to the Table Service or SQL Azure, the more money you will save. With the Table Service you save money directly by making less requests, with SQL Azure you save money indirectly by requiring less SQL databases to service your queries.

In the previous example, we improved performance and saved money by joining our static data. Although this works well for static data, it doesn't work so well for non static data such as dynamic data or infrequently changing data.

14.3 Joining Dynamic and Infrequently changing data together

In the example in the previous section, we easily saved money and improved performance by performing joins between our dynamic data and our static data. In this section we will look at what options we have with joining dynamic and infrequently changing data together.

For our earlier example we skipped over how we would represent our shirt in the Shopping Cart. As mentioned earlier, we have two choices, we can either perform a client side join (as we did in the previous section with the static data) or we can duplicate data.

In this section we will look at how we can:

- Duplicate data
- Join Uncached Data
- Join Cached Data

Let's take a look at how we can duplicate data.

14.3.1 Duplicate Data instead of joining

Because the shirt data is not static data (but is in-fact infrequently changing data), we need to find a different way of associating that data with our dynamic shopping cart item. In this case we will duplicate the shirt data within the shopping cart item, as shown below:

```
public class ShoppingCartItem : TableServiceEntity
{
   public string ShirtName { get; set; }
   public string ShirtDescription { get; set; }
   public int Price { get; set; }
   public string Material { get; set; }
   public string Size { get; set; }
}
```

In the above instance we have chosen to store a complete copy of the selected shirt as part of the shopping cart item entity.

Higher Storage Costs

Although this approach will mean that we will not need to perform a join with the shirts table, it does mean that our table will be much larger than a traditional relational table meaning higher storage costs.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

So now that we have our shopping cart item, to correctly display our StrongShoppingCartItem object from our earlier example. The following code shows how we can modify our earlier query to do this.

```
select new StrongShoppingCartItem
        {
           SelectedShirt = new Shirt {PartitionKey="Shirts",
                              RowKey=shoppingCartItem.ShirtName,
                        Description=shoppingCartItem.ShirtDescription,
                        Price=shoppingCartItem.Price},
           Material = material,
           Size = sizeType
        };
```

As you can see from the above code, we simply project our duplicate data into our StrongShoppingCartItem object by instatiating a new Shirt object with the data from the ShoppingCartItem.

Data Synchronization

Another issue with duplicating data is data synchronization. Although the shirt data is infrequently changed when a changed does occur (such as the price) then all items that are present in a customer's shopping basket will not be automatically updated with the new price. If your business model allows you to have stale data then this obviously not such a problem but if your pesky customers (who needs them) really wants the correct price to reflected in the shopping basket then you need to perform some method of synchronizing the master table to all the duplicates. A simple method of keeping the data synchronized is to publish a message to a queue to state that an item has changed then have a worker role pick up that message and update all items in the table with the correct data

14.3.2 Client side joining uncached data

If data synchronization is a big concern and our dynamic data is a very small set of data, we could actually take the hit of performing a client side join. To achieve that we would need to modify our ShoppingCartItem to support the join:

```
public class ShoppingCartItem : TableServiceEntity
{
   public string Shirt {get;set;}
   public string Material { get; set; }
   public string Size { get; set; }
}
```

In the above example we have just removed the duplicate shirt properties and just store a reference to the shirt (as we did before). Now that we have modified our entity to store a reference, we need to modify our query to join the data together. The following code shows how we could do this:

```
   var materials = (IEnumerable<Material>)Cache["materials"];
   var sizeTypes = (IEnumerable<SizeType>)Cache["sizeTypes"];

   var shoppingCartItemContext = new ShoppingCartItemContext ();

   var shoppingCartItems =
      shoppingCartItemContext.ShoppingCartItem.ToList();

   var shirtsContext = new ShirtContext();

   var q = from shoppingCartItem in shoppingCartItems
           join sizeType in sizeTypes
              on shoppingCartItem.Size equals sizeType.RowKey
           join material in materials
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
            on shoppingCartItem.Material equals material.RowKey
         select new StrongShoppingCartItem
         {
            SelectedShirt = (from shirt in shirtsContext.ShirtTable
                             where shirt.PartitionKey=="Shirts" &&
                             shirt.RowKey == shoppingCartItem.Shirt
                             select shirt).First(),
            Material = material,
            Size = sizeType
         };
```

The key thing to note about the above example is that our shirt query is not cached, and will invoke a call to the Table Service for each item returned. The following extract shows where this is performed:

```
SelectedShirt = (from shirt in shirtsContext.ShirtTable
                 where shirt.PartitionKey=="Shirts" &&
                 shirt.RowKey == shoppingCartItem.Shirt
                 select shirt).First(),
```

Since the data returned from the shopping cart is pretty small this is a pretty useful technique to perform such a filter. If you were dealing with a much larger set of shopping cart data (hundreds of items) then this would start to perform very badly.

Caching In-frequently changing data

So the above example will work for small datasets such as shopping carts but this will still potentially cost us quite a bit of cash (as we will be charged for each request made.

We can improve performance of the previous query by making use of a distributed cache (such as AppFabric Caching or Memcached). Rather than calling out to the Table Service for each request, we could make a call to a distributed cache, such as Memcached. For more details on how to use Memcached, see Chapter 6

Although not having the ability to join data across table does present challenges it's not always impossible challenges. For the most part we can use some of the different techniques above to get around those problems.

SQLite

If you are working with infrequently changing data (such as shirts) then you may wish to consider using SQLite to host a local cached version of your data. This would allow you to perform SQL queries on local cached data without calling out to the Table Service or SQL Azure.

LOTS OF JOINS INVOLVING DYNAMIC DATA

In some circumstances it's just going to be impossible to use the Table Service. If you have lots of dynamic data performing live queries across various table joins then you are just not going to be able to represent that easily in the Table Service. In these instances SQL Azure is the most appropriate choice. Also if you need to perform a transaction across various tables then SQL Azure is again the right choice.

Lucene.NET

If you are storing your data in SQL Azure or the Table Service and you need to perform text searching, you can export your data out of these databases and perform searches with Lucene.NET. For more details see:

http://incubator.apache.org/lucene.net/

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

14.4 Summary

In this chapter we have tried to break away from automatically building relational database only based solutions and instead try and build hybrid solutions that give us the benefits of both platforms.

We have seen that by breaking foreign key constraints we can make effective use of the cache to store our static data allowing us to use either the table service or SQL Azure as our underlying storage platform.

In this chapter we have also see that that if we chose to make use of the Table Service (which is much more scalable than SQL Azure) then we can easily join our dynamic data back to our static reference data without financial or performance penalties.

Even though SQL Azure isn't as naturally scalable as the Table Service we have looked at how we can shard SQL Azure to build a highly scalable relational database solution.

Finally we have seen that when it comes to using infrequently changing reference data (e.g. Hawaiian Shirts) then depending on what we want to do with that data should influence what technologies we choose. In some circumstances live fresh data, transactional data, data with many joins across dynamic tables then SQL Azure is the only choice. However if you are looking to join dynamic data with in-frequently changing data then you can still use the Table Service by thinking about our queries and our performance.

Regardless of the solution you plan to use then really you should think about what your application needs to do before you choose what technology you should choose. Don't automatically reach for SQL Azure (which is awesome) but think if the Table Service will meet your needs.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

15

Processing with Worker Roles

In this chapter:

- Discover how to scale the backend
- How to process messages
- How to use the service management APIs to control your application

In Azure there are two roles that run your code. The first, the web role, has already been discussed. It plays the role of the web server, communicating with the outside world. The second role is the worker role. It plays the role of the backend server. You might use one to run asynchronous or long running tasks. The worker role is usually message based. It will usually be receiving these messages by polling a queue or some other shared storage.

It is important to remember that a worker role is a template for a server in your application. The service model for your application defines how many instances of that role need to be started in the cloud. The role definition is similar to a class definition, and the instances are like objects.

If your system has Windows services or batch jobs, they can easily be ported to a worker role. When you are doing this you will want to keep in mind that Windows Azure doesn't have a job scheduler facility, and you might need to build your own.

Many systems have a series of regularly scheduled backend tasks. These might be in the form of processing all the new orders each night at 11pm. Or perhaps you have a positive pay system for banking, and it needs to connect to your bank each day before 3pm, except for banking holidays.

The worker role is intended to be started up and left running to process messages. You will likely want to dynamically increase and decrease the number of instances of your worker role to meet the demand on your system, as it increases and decreases throughout your business cycle.

You could easily build a scheduling worker role that polls a table that represents the schedule of work to do. As jobs are needed to execute, it could create the appropriate worker instance, pass it the job's instructions, and then shut down the instance when the work is completed. You could easily do this with the service management APIs, which are discussed later in this chapter.

15.1 A simple worker role service

When it is all said and done, working with worker roles is quite easy. The core of the code for the worker role is the code that you need executed to get the work done. This code is the normal business code that you need to execute. There isn't anything special about the bulk of a worker role because of this. It's the wrapper or handler around the business code that is interesting. There are also some key concepts you will want to pay attention to so that you build a reliable and manageable worker role.

In this section, we are going to build a basic worker role service. We have to have some way to communicate with the worker role. To start off with we will first send messages to the worker through a queue, showing you

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

how to poll a queue. We won't go too deep into queues, since they are covered thoroughly in their own chapter. We will then upgrade the service so you can use inter-role communication to send messages to your service. Along the way we will cover debugging, logging, and service health as well.

I will use the term 'service' fairly loosely when we are talking about worker roles. I see worker roles as providing a service to the rest of the application, hopefully in a decoupled way. We don't mean to imply the use of WS-* and web service protocols necessarily, although that is one way to communicate with the role.

15.1.1 No more Hello, World

Since 'Hello World' doesn't really cut it as a sample this late in any book we are going to build a service that reverses strings. This is an important service in any business application, and the string reversal industry is highly competitive. There will be two parts to our service. The first part will be the code that actually does the work. While it is composed of some unique intellectual property, it isn't very important in our sample. This is the piece that will be reversing strings. This is the business end of your service, where the service does what it needs to on your behalf. The other part of the service is how we get messages into the service or requests for work. This second part can have many shapes, and it comes down to the architectural model you want to support. Some workers never receive messages; they just constantly poll a database, or file system, and process what they find.

To build our software we are going to need to open up Visual Studio 2010, and start a new cloud project. For this project, just add one worker role, and give it the name "WorkerRole–ProcessString".

Figure 15.1 – We are starting our string reversal company. To build our service we will start with a worker role. It will do all of our work, and make it easy for us to scale as our business grows, especially during the string reversal peak season.

At this business end will be our proprietary and award winning algorithm for reversing strings. We intend to take the string reversal industry by storm, and really upset some industry captains. The core method will be called ReverseString, and will take a string as its only parameter.

Listing 15.1 – The magical string reversal method

```
private string ReverseString(string originalString)
{
   int lengthOfString = originalString.Length;
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
    char[]reversedBuffer = new char[lengthOfString];

    for (int i = 0; i < lengthOfString; i++)
    {
       reversedBuffer [i] = originalString[lengthOfString - 1 - i];
    }

    return new string(reversedBuffer);
}
```

The code in listing 15.1 is fairly simple, and is probably something you have written in the past. The method simply declares a character array to be a buffer that is the same length as the original string (because our R&D department has discovered that every reversed string is exactly as long as the original string.) We then loop over the string, and taking characters off the end of the original string and put it at the front of the buffer, moving along the string and the buffer in opposite directions. To finish it out we return the string that is in the buffer back to the caller.

For sample purposes, we are going to put this business logic right in the class called `WorkerRole.cs`. Normally this code would be contained in its own class, and referenced into the application. You can do that later if you want, but we want to keep the sample simple so you can focus on what is important.

You should have a good grip on this code because it is just normal .NET code, and something you would write on any platform that supports .NET (mobile, desktop, on-premises servers, etc.), not just something in the cloud.

We have chosen to put this service in a worker in the cloud so that we can dynamically scale how many servers we have running the service, based on usage and demand. We don't want to distract our fledgling company from writing the next generation of string reversal software with the details and costs of running servers.

If you ran this project right now you wouldn't see anything happen. The cloud simulator on your desktop would start up, and the worker role would be instantiated, but nothing would get done. By default the worker service comes with an infinite polling loop in the `Run` method. This `Run` method is what is called once the role instance is initialized and ready to run. I like that they called it `Run`, but calling it `DoIt` would have been funnier.

Like we mentioned above, now that we have our code in the worker role, how do we access it and use it? The next section will focus on the two primary ways we can send messages to a worker role instance in an active way.

15.2 Communicating with a worker role

Worker roles can receive the messages they need to process in either an push or a pull way. Pushing a message to the worker instance is an active approach, where we are directly giving it work to do. The alternative is to have the role instances call out to some shared source to gather work to do, in a sense pulling in the messages they need. When pulling messages in, remember that there will possibly be several instances pulling in work, and you will need mechanism similar to what the Azure Queue provides to avoid conflict from the different worker role instances trying to process the same work.

Remember the trick here is the difference between roles and role instances, which we have covered earlier. While sometimes it is convenient to think of workers as one entity, when they are running, they don't run as a role, but as one or more instances of that role. When you are designing and developing your worker roles keep this duality in mind, think of the role as a unit of deployment and management, and the role instance as the unit of work assignment. This will help reduce the number of problems in your architecture. One advantage that worker roles have over web roles is that they can have as many service endpoints as they would like, using almost any transport protocol and port. Web roles are limited to HTTP/S and can have only up to three endpoints. We will use this flexibility to provide several ways to send messages to the worker role.

There are three approaches to sending messages to a worker role instance that we will cover. The first is a pull model, where each instance will poll a queue for work to be completed. The second and third approaches will be push oriented, and we will be sending messages directly to the instances from the producer. In the second approach the producer of the message is from outside Azure, while the third approach will be when the message sender is inside your application in Azure.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

15.2.1 Consuming messages from a queue

The most common way for a worker role to receive messages is through a queue. This will be covered in depth in Chapter 16 on Message Queues, but we want to cover it briefly here.

The general model is to have a while loop that never quits. This approach is so common that the standard worker role template in Visual Studio provides one for you. The role instance tries to get a new message from the queue it is polling on each iteration of the loop. If it gets a message, it will process the message. If it doesn't, it will wait a period of time (perhaps five seconds) and then poll the queue again.

The core of the loop is what is calling our business code. Once we have a message we pass it off to the code that does our work. Once that work is done we delete the message from the queue, and then poll the queue again.

```
while (true)
{
      CloudQueueMessage msg = queue.GetMessage();
      if (msg != null)
      {
          DoWorkHere(msg);
          queue.DeleteMessage(msg);
      }
      else
      {
           Thread.Sleep(5000);
      }
}
```

You might jump to the conclusion that you could easily poll an Azure table for work instead of polling a queue. Perhaps you have a property in your table called `Status` that defaults to `new`. The worker role could poll the table, looking for all entities that have the property of `Status` equal to `new`. Once a list is returned, the worker could process each entity, and then set their `Status` to `complete`. At its base this sounds like a simple approach.

This approach is a red herring as it suffers from some severe drawbacks, drawbacks you might not find until you are in testing or in production because they won't show up until you have multiple instances or your role running.

The first problem is of concurrency. If you have multiple instances of your worker role polling the table, they could each retrieve the same entities in their queries. This would result in them being processed multiple times, and possibly leading to the status updates getting entangled. This is the exact concurrency problem the Azure queues were designed to avoid.

The other issue, and more important issue, is one of recoverability and durability. You want your system to be able to recover if there is a problem processing a particular message. Perhaps you have each worker role set the `status` property to the name of the instance to track that the entity is being worked on by a particular instance. When the work is completed the instance would then set the `status` property to done. On the surface this approach seems to make sense. The flaw is that when an instance fails during processing (which will happen) the entity will never be recovered and processed. It will never be reset from the status of being flagged with the instance name of the worker processing the item. Because of this, it will never be cleared, and never picked back up in the query of the table to be processed. It will in effect be 'hung'. The system administrator would have to go in and manually reset the status property back to new. In effect, there isn't a way for the entity to be recovered from a failure, and reassigned to another instance.

It would take a fair amount of code to overcome the issues of polling a table by multiple consumers, and in the end you would end up having built the same thing as the Azure queue system. The queue system is designed to play this role, and removes the need to write all of this dirty plumbing code. The queue provides a way for work to be distributed amongst multiple worker instances, and to easily recover that work if the instance fails. A key concept of cloud architecture is to design for failure recoverability in an application. It is to be expected that

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

nodes go down (for one reason or another), and will be restarted and recovered, possibly on a completely different server.

Queues are the easiest way to get messages into a worker role, and they will be discussed in detail in the next chapter. Next we will discuss inter-role communication, which lets a worker role receive a message from outside of Azure.

15.2.2 Exposing a service to the outside world

Web roles are built to receive traffic from outside of Azure. Their whole point in life is to receive messages from the Internet (usually from a browser) and respond with some message (usually HTML.) The great thing is when you have multiple web role instances they are automatically enrolled in a load balancer. This load balancer automatically distributes the load across the different instances you have running.

Worker roles can basically do the same thing, but since we are not running in IIS (which is not available on a worker role) we have to host our service ourselves. The only real option is to build the service as a WCF service.

We are going to start this by removing the loop that polls the queue, and put in some service plumbing. When we host a service in a worker role, regardless if it will be for external or internal use, we need to declare an endpoint. Depending on how we configure this endpoint will determine whether it allows traffic from sources internal or external to our application. If it is configured to run externally then it will use the Azure load balancers and distribute service calls across all of the role instances running the server, very similar to how the web role does it. We will look at internal service endpoints in the next section.

Figure 15.2 – Worker Roles have two ways of exposing their services. The first is as an INPUT service. These are published to the load balancer and available externally (role 0). The second are INTERNAL services, which are not behind a load balancer, and only visible by your other role instances (role 1).

Our goal is to convert our little string reversal method into a WCF service, and then expose that externally so that customers can call the service. The next step in our process is to define the endpoint. You can do this the macho way in the configuration of the role, or you can do it in the Visual Studio properties window. If you right click and choose properties on the "Worker-Process String" worker role in the Azure project you will see the window in figure 15.3.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 15.3 – We are adding an external service endpoint to our worker role. This service endpoint will be managed by Azure, and be enrolled in the load balancer. This will make the service available outside of Azure.

Name the service endpoint "StringReverseService." Set it to be an inbound port, using TCP on port 2202. We won't be using any certificates or security at this time. After you save this setting you will find the equivalent settings in the ServiceConfiguration.csdef file.

```
<Endpoints>
 <InputEndpoint name="StringReverserService" protocol="tcp" port="2202" />
</Endpoints>
```

Normally we might host our service in IIS or WAS, but we don't have those available in a worker role. In the future we would be able to use AppFabric, but that isn't available yet, so we are going to have to do this the old fashioned way. We are going to host our WCF service using ServiceHost, which is exactly that, a host that will act as a container to run our service in. We need something to contain our service, manage the endpoints and configuration, and handle the incoming service requests.

Next we are going to add a method called `StartStringReversalService`. This method will wire up the service to the `ServiceHost` and the endpoint we defined.

Listing 15.2 – The StartStringReversalService method wires up our service

```
private void StartStringReversalService()
{
   this.serviceHost = new ServiceHost(typeof(ReverseStringTools));        A

   NetTcpBinding binding = new NetTcpBinding(SecurityMode.None);

   RoleInstanceEndpoint externalEndPoint =
RoleEnvironment.CurrentRoleInstance.InstanceEndpoints["StringReverserService"];        B

serviceHost.AddServiceEndpoint(
        typeof(IReverseString),
        binding,
        String.Format("net.tcp://{0}/StringReverserService",
[CA]ExternalEndPoint.IPEndpoint));

   try
   {
      this.serviceHost.Open();                                              C
   }
   catch (Exception ex)
   {
      Trace.TraceError("Could not start string reverser servicehost. {0}",[CA] ex.Message);
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
D
    }

    while (true)
    {
        Thread.Sleep(500000);                                    E
    }
}
```

#a-#e should be cueballs

Listing 15.2 is an abbreviated version of the real method so that it fits into the book better. We didn't take anything out that is super important. We took out a series of trace commands (#D) so we could watch the startup and status of the service. We also abbreviated some of the error handling, something you would definitely want to beef up in a production environment.

Most of this code is normal for setting up a `ServiceHost`. We first have to tell the service host the type of the service that is going to be hosted, #a, in this case it is the `ReverseStringTools` type. When we go to add the service endpoint to the service host we are going to need three things, the ABCs of WCF Address, Binding, and Contract.

The Contract is provided by our code, IReverseString, and is a class file that you can reference to share service contract information(or use MEX like a normal web service). The Binding is a normal TCP binary binding, with all security turned off. I would only run with security off for debug and demo purposes!

Then the Address is needed. We can setup the actual address by referencing the service endpoint from the Azure project. The real IP address the service will be running under will not be known until runtime, so we have to build it on the fly. Line #B does this by accessing the collection of endpoints from the `RoleEnvironment.CurrentRoleInstance.InstanceEndpoints` collection. The collection is a dictionary, so you can pull out the endpoint you want to reference with the name you used when setting it up, in this case `StringReverserService`. Once we have a reference to the endpoint we can access the IP address that we need to set up the service host.

Figure 15.4 – A simple client that consumes our super string reversing service. The service is running in a worker role, running in Azure, behind the load balancers. kltpzyxM! kltpzyxM! kltpzyxM!

Once we have that wired up we can start the service host. This will plug in all the components, fire them up, and start listening for incoming messages. We do this on line #C, with the `Open` method.

Once the service is up we want the main execution thread to sleep forever so that the host stays up and running. If we didn't have the sleep loop at line #E the call pointer would fall out of the method, and we would lost our context, losing the service host. At this point our worker role instance is just sitting there, sleeping, while the service host is running listening for and responding to messages.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Exposing public service endpoints is very useful, but there are times when you want to expose services for just your use, and you don't want them public. In this case we will want to use inter-role communication, which we will look at next.

15.2.3 Inter-role communication

Exposing service endpoints, as we just discussed, can be useful. But there are many times when you just need a way to communicate between your role instances. Usually I would use a queue, but there are times when there might be a need for direct communication, either for performance reasons, or because the process is synchronous in nature.

We can enable communicating directly from one role instance to another, but there are some issues you should be aware of first. The biggest issue is that you have direct access to an individual role instance. There isn't any separation that can deal with load balancing, or durability of the work you are performing. If you are communicating with an instance, and it goes down, then your work is lost. You will have to write code to handle this possibility on the client side.

You add an internal endpoint in the same way you add an external endpoint, except you set the type from Input to Internal, as seen in figure 15.5. The port will automatically be set to 'dynamic', and will be managed for you under the covers by Azure.

Figure 15.5 – You can set up an internal endpoint in the same manner that you setup an external endpoint. In this case though your service will not be load balanced, and the client will have to know which service instance to talk to.

Using an internal endpoint is a lot like using an external endpoint, from the point of view of your service. In either case, your service never really knows about the possible other instances running the service in parallel. The load balancing is handled outside of your code when you are using an external endpoint. Internal endpoints don't have any available load balancing. This places the choice of which instance of the service to consume on the shoulders of the consumer themselves.

Most of the work with internal endpoints is handled on the client side, your service consumer. Since there are a differing number of instances of your service running at any time, you have to be prepared to decide which instance to talk to, if not all of them. You also have to be wily enough to not call yourself if you are calling the service from a sibling worker role instance.

You can access the set of instances running, and their exposed internal endpoints with the RoleEnvironment static class.

```
foreach (var instance in [CA]RoleEnvironment.CurrentRoleInstance.Role.Instances)        #1
{
   if (instance != RoleEnvironment.CurrentRoleInstance)                                  #2
      SendMessage(instance.InstanceEndpoints["MyServiceEndpointName"]);
}
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

#1,#2 should be a cueball

In our sample code above, #1, we are looping through all of the available role instances of the current role. We could access a collection of any type of role in our application. For each instance, #2, we are checking to see if that instance is the instance this code is running in. If it isn't, we will send that instance a message. If it is the same instance, we won't send it a message, because sending a message to ourselves is usually not productive.

All three ways of communicating with a worker role have their advantages and disadvantages, and each has a role to play in your architecture. Use a queue for complete separation of your instances from the service consumers. External endpoints are useful to expose your service publicly, and this lets you leverage the Azure load balancer. Internal endpoints are excellent for direct and synchronous communication to a specific instance of your service.

15.3 Common uses for worker roles

seque

15.3.1 Offloading work from the front end

Now that we have talked about the shape of a worker role, and how to communicate with it, we can get down to the business of what we do with a worker role. Most of us have shopped online. You might have even bought this book online. We are all familiar with the user experience of putting products into a shopping cart and then checking out with an online retailer. How they process your cart and your order is one of the key scenarios for how a worker role might be used in the cloud.

Many large online retailers split the checkout process into two pieces. The first piece is interactive, and is user facing. The user happily fills their shopping cart with lots of stuff, and then chooses to check out. At that time, the application gathers the customer's payment details, and tells the customer the order has been processed. They are even given an order number for reference purposes.

After the customer facing piece is done, the backend magic kicks in to complete the processing of your order. You see, when the retailer sent you an order number they were sort of fibbing. All they did was submit the order to the backend, and give you the order number that will be used to track it. When the customer checked out, their order was tagged with an order number, and the email was sent to the customer. The front end then submits the order to the backend processing system via a message queue. One of the servers that are part of the backend processing group picks up the order and completes the processing. This probably involves charging the credit card, verifying inventory, and determining the ability to ship according to the customer's wishes. Once this backend work is completed, a second email is sent to the customer with an update, usually including the package tracking number, and any other final details.

Figure 15.6 – The typical online retailer will process a customer's order in two stages. The first just saves the cart for processing and immediately sends back a thank you email with an order number. Then the magic backend servers pick up the order and actually process it, resulting in a final email with all of the real details.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The work is split into two pieces to reduce the amount of time the customer needs to wait during the checkout process. This also makes it easier to scale the system, and plan for failure. You wouldn't want a web server to crash while processing a customer's order and lose the revenue would you?

By breaking the system into two pieces the online retailer gains a few advantages. The biggest is that the user's experience of checking out is much faster, giving them a nice shopping experience. This also takes a lot of load off of the web servers, which really should just be simple HTML shovels. Since only a fraction of shoppers actually check out (etailers call this the conversion rate) we want to be able to scale the web servers very easily. Having them broken out makes it easy to scale them horizontally (by adding more servers), and makes it possible for each web server to only require simple hardware. The general strategy at the web server tier is to have an army of ants, or many low end servers.

This leaves the backend all the time it needs to actually process the orders. Backend server farms tend to be comprised of fewer, larger servers as compared to the web servers. While we can scale the number of backend servers as well, we won't have to do that as often, as we can just let the flood of orders backup in the queue. As long as our server capacity can process them in a few hours, then that is ok.

Azure does provide a variety of server sizes for our instances to run on, and sometimes we want more horsepower in one box for what we are doing. In that case we would want to use threading on the server to tap that entire horsepower.

15.3.2 Using Threads in a Worker Role

There may be times when the work assigned to a particular worker role instance needs multithreading, or the ability to process work in parallel by using separate threads of execution. This is especially true when you are migrating an existing application to the Azure platform. Developing and debugging multithreaded applications is very difficult. Deciding to use multithreading is not a decision you should make lightly.

The worker role does allow for the creation and management of threads for your use. As with code running on a normal server, you don't want to create too many threads. When the number of threads increases, so does the amount of memory in use. The context switching cost of the CPU will also hinder efficient use of your resources. You should limit the number of threads you are using to two to four per CPU core.

The most common use of threads is to spin up an extra thread in the background to process some asynchronous work. Doing this is ok. If you plan on building a massive computational engine, then you are better off using a framework to do the heavy lifting for you. The Parallel Extensions for .NET is a framework Microsoft is developing to help you parallelize your software. It is expected that the framework will ship with .NET 4.0 sometime in 2010.

While we always want to logically separate our code to make it easier to maintain, sometimes the work involved doesn't need a lot of horsepower, and so we may want to deploy both the web and the worker side of the application to one single web role.

15.3.3 Simulating worker roles in a web role

Architecting your application into discrete pieces, some of which are 'front end' and some of which are 'back end' is a good thing. But there are times when you need the logical separation, but not the physical separation. This might be for speed reasons, or because you don't want to pay for a whole worker role instance when you just need some lightweight background work done.

It is important that if you go down this path that you must architect your system so you can easily break it out into a real worker role later one as your needs change. This means making sure that while you are breaking the physical separation, you are at least keeping the logical separation. You should still use the normal methods of passing messages to that worker code. If it would use a queue to process messages if it was a real worker instance, then it should use a queue in this simulated worker instance as well.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 15.7 – You can simulate a worker role in your web role if it is very lightweight in nature. Make sure that you keep the logical separation intact for later refactoring.

At some point you will need to break the code back out to a real worker role. When you do, you won't want to have to rewrite a whole bunch of code.

OFF THE FABRIC CONTROLLER'S RADAR

Be aware that the fabric controller will be ignorant of what you are doing, and will not be able to manage your simulated worker role. If that worker role code goes out of control, it will take down the web instance it is running in, which could cascade to a series of other problems. Fair enough, you have been warned.

The concept to do this is to put the worker code into a separate library so that primary concerns of the web instance are not intermingled with the concerns of the faux worker instance. You will then reference in that library, and execute it on its own thread, passing messages to it however you would like. This will also make it much easier to split it out into its own real worker role later.

UTILIZING BACKGROUND THREADS

The other concern is how to get a background thread running to execute the faux worker code. An approach we have worked with is to launch the process on a separate thread during the `Session_Start` event of the `global.asax`. This will fire up the thread once when the web app is starting up, and leave it running. Our first instinct was to use the `Application_Start` event but this won't work. The RoleManager isn't available in the `Application_Start` event, so it is too early to start the faux worker.

We want to run the following code:

```
Thread t = new Thread(new ThreadStart(FauxWorkerSample.Start));
t.Start();
```

Since we are putting the thread start code in the `Session_Start` event, this has the effect of trying to start another faux worker every time a new ASP.NET session is started, which is whenever there is a new visitor to your web site. To protect against thousands of background faux workers being started, we use the singleton pattern. This pattern will make sure that only one faux worker is started in that web instance. When we are about to create the thread, we check a flag in the application state to see if a worker has already been created.

```
object obj = Application["FauxWorkerStarted"];

if (obj == null)
{
     Application["FauxWorkerStarted"] = true;
     Thread t = new Thread(new [CA]ThreadStart(FauxWorkerSample.Start));
     t.Start();
}
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

If the worker has not been created, then the flag will not exist in the application state property bag, and will equal `null` in that case. If this is the first session, then the thread will be created, pointed at the method you give it (FauxWorkerSample.Start in this case), and start processing in the background.

When you start it in this manner you will have access to the `RoleManager`, with the ability to write to the log, manage system health, and act like a normal worker instance.

15.3.4 State-directed workers

Sometimes the code that a worker role runs is large and complex and this can lead to a long and risky processing time. In this section we will look at a strategy you can use to break this large piece down into manageable pieces, and a way to gain flexibility in your processing.

As we have said time and time again, worker roles tend to be message centric. The best way to scale them is by having a group of instances take turns consuming messages from a queue. As the load on the queue increases, you can easily add more instances of the worker role. As the queue cools off, you can destroy some instances.

In this section, we'll look at why large worker roles can be problematic, how we can fix this problem, and the inevitable drawbacks. Let's start by looking at the pitfalls of using a few, very large workers.

THE PROBLEM

Sometimes the work that is needed on a message is large, and complicated. This leads to a heavy, bloated worker. This heaviness also leads to a brittle code base that is difficult to work with and maintain because of the various code paths and routing logic. A worker that takes a long time to process a single request is harder to scale, and can't process as many messages as a group of smaller workers. A long running unit of work also exposes your system to more risk. The longer an item takes to be processed, the more likely that work will fail and have to be started over. This is no big deal if the processing takes three seconds, but if it takes 20 minutes or 20 hours, then you have a significant cost to failure.

Figure 15.8 – The 'pig in a python' problem can be seen in many parts of technology and business. This is when a unit of work takes a long time for the process to complete, just like when a python eats a pig. It can take months for the snake to digest the pig, and it can't do much of anything else during that timeframe.

This problem can be caused by one message being very complex to process, or by a batch of messages that are being processed as a group. In either case the unit of work being accomplished is large, and this raises risk. This problem is often called the 'pig in a python' problem, because you end up with one large chunk of work moving through your systems. We need a way to digest this work a little more gracefully.

THE SOLUTION

The best way to digest this large pig is to break the large unit of work into a set of smaller processes. This will give you the most flexibility when it comes to scaling and managing your system.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

You do want to be careful that you don't break the processes down to sizes that are too small. At this level, the overhead of communicating with the queue and other storage mechanisms may actually introduce more overhead than you were looking for.

Analyze the stages of processing on the message and you will likely conceive of several stages to the work. You can figure this out just by drawing a flow diagram of the current bloated worker code. For example, when processing an order from an ecommerce site you might have the following stages.

1. Validate the data in the order
2. Validate the pricing and discount codes
3. Enrich the order with all of the relevant customer data
4. Validate the shipping address
5. Validate the payment information
6. Charge the credit card
7. Verify the products are in stock, and able to be shipped
8. Enter the shipping orders into the logistics system for the distribution center
9. Record the transaction in the ERP system
10. Send a notification email to the customer
11. Sit back and Profit

You can think of each state the message goes through as a separate worker role, connected together with a queue for each state. Instead of one worker doing all of the work for a single order, it only processes one of the states for each order. Each queue represents each state the message could have. There may be some other processing states you may want to plan for. Perhaps one for really bad orders that need to be looked at by a real human, or perhaps there are platinum level customers that get their orders processed and shipped before normal run of the mill customers. The platinum orders would go into a queue that is processed by a dedicated pool of instances.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 15.9 - A monolithic worker role compared to a state driven work role. The big worker completes all the work in one step, leading to the 'pig in a python' problem of being harder to maintain and extend as needed. Instead we look at breaking the process into a series of queue and workers, each dedicated to servicing a specific state or stage of the work to be done.

You could even have a bad order routed to an Azure table. A customer service representative could then access that data, with either a CRM application, or a simple InfoPath form, to fix the order, and resubmit it back into the proper queue to continue being processed. This process is called 'repair and resubmit', and it is an important element to have in any enterprise processing engine.

Figure 15.9 compared a big worker, that performs all of the work, to a series of smaller workers that break the work out (validate, ship, and notify workers).

You won't be able to put the full order details into the queue message, there won't be enough room. The message should contain a complete work ticket, representing where the order data can be found (perhaps an order id), as well as some state information, and any information that would be useful in routing the message through the state machine. This might be the service class of the customer, for example, platinum versus silver.

As the business changes over time and it will, making changes to how the order is processed is much easier. For example, you might need to add a new step between eight and nine. You would need to create a new queue and a new worker role to process that queue. Then the worker role for the state right before the new one would need to be updated to point to the new queue. Hopefully the change of the existing parts of the system can be limited to configuration.

EVEN COOLER - MAKE THE STATE WORKER ROLES ITS OWN AZURE SERVICE

In this scenario, it would make sense to separate each state worker role to its own service in Azure. This further decouples them from each other. When you need to restart one worker role and its queue, you won't affect the other roles. How you want to manage your application in the cloud should be a primary consideration in how you structure the Visual Studio solution. Each solution becomes a single management point. If you want to manage different pieces without affecting the whole system, those should be split out into their solutions.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In a more dynamic organization you might need to route a message through these states based on some information that is only available at runtime. The routing information could be stored in a table, with rules for how the flow works, or by simply storing the states, and their relationships in the cloud service configuration file. Both of these approaches would let you update how orders were processed at runtime without having to change code. I have done this myself when orders needed different stages depending on what was in the order, or where it was going. In one case, if a controlled substance was in the order, the processing engine had to execute a series of additional steps to complete the order.

This approach is often called a poor man's service bus. That is because it has a simple way of connecting the states together, and they are fairly firm at runtime. If you require a greater degree of flexibility in the workflow, then you would want to look at a pattern called the Itinerary[1] pattern. This lets the system build up a schedule of processing stops based on the information present at runtime. These systems can get a little more complicated, but they result in a system that is more easily maintained when there is a complex business process.

OOPS, IT'S NOT NIRVANA

As you build this out, you will discover a drawback. You now have many more running worker roles to manage. This can create more costs, and you still have to plan for when you eventually will swallow a pig. If your system is tuned for a slow work day, with one role instance per state, and suddenly you receive a floodgate of orders, the large amount of orders will move down the state diagram, like a pig does when it is eaten by a python. This forces you to scale up the number of worker instances at each state. While this flexibility is great, this can get expensive. With this model you have several pools of instances, instead of one general purpose pool. This results in each pool having to increase and then decrease as the pig moves through the pipeline. In the case of the pig, this can lead to a stall in the state machine. This can be done easily using the service management APIs, but it takes time to spin up and spin down instances, perhaps a few minutes.

Figure 15.10 – By moving to a consolidated state directed worker, we will have one queue and one worker. The worker will act as a router, sending each inbound message to the appropriate module based on the messages state and related itinerary. This allows us to have one large pool of workers, but make it easier manage and decompose our bulky process.

[1] For more information on the Itenerary pattern, please read the Application Architecture Guide from Patterns & Practices at Microsoft. It can be found at http://apparchguide.codeplex.com.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The next step to take, to avoid the pig in a python problem, is to build you worker roles so that they are generic processors, all able to process any state in the system. You can still keep the separate queues. This makes it easier to know how many messages are in each state. You could also condense the queues down to one, with each message declaring what state the order is in as part of its data. I don't prefer this approach, as it leads to favoritism for the most recent orders placed in the processors, and requires you to restart all of your generic workers when you change the state graph. You can avoid this particular downfall by driving the routing logic with configuration and dependency injection. Then you would only need to update the configuration of the system, and deploy a new assembly to change the behavior of the system.

The trick is to encapsulate the logic for each state in the worker, separating it so it is easily maintainable, while pulling them all together so there is only one pool of workers. The worker, in essence, becomes a router. You can see how this might work in figure 15.10. Each message is routed based on its state and other runtime data, to the necessary state processor. This functions much like a factory. Each state would have a class that knows how to process that state. Each state class would implement the same interface, perhaps IorderProcessStage. This makes it easy for the worker to instantiate the correct class, based on the state, and then processes it. Most of these classes will then send the message back to the generic queue, with a new state, and the cycle would start again.

While you are writing all of this code, and even during runtime, you are going to run into problems. In order to solve these problems you will need to have some insight into what is exactly going on with your code.

15.4 Working with Local Storage

There are times when the code we are working with need to read and write to the local filesystem. Windows Azure allows for you to request and access a piece of the local disk on your role instance. You can create this space by using the configuration of your role. We recommend that you only use local storage when you absolutely have to, because of some limitations we will cover later in this section. You will likely need to use local storage the most when you are migrating existing frameworks or applications to the cloud that require local disk access. You will not have control over the path of the directory you are given access to. Make sure the file path your code needs to access is part of your configuration, a hard coded path will never remain accurate in the cloud environment.

15.4.1 Setting up Local Storage

You can configure the local storage area you need as part of your role by adding a few simple lines of configuration to your role. The tag we are going to work with is the `LocalStorage` tag. This tag will tell the fabric controller to allocate local file storage space on each server the role instance is running on. In the configuration element, you need to name the storage space. This name will become the name of the folder that is reserved for you. You will need to define how much file system space you will need. The current limit is 20GB per role instance, and a minimum of 1MB.

Listing 15.3 – Declaring local file storage

```
<LocalResources>
      <LocalStorage name="FilesUploaded" cleanOnRoleRecycle="false" [CA]sizeInMB="15" />  #1
      <LocalStorage name="VirusScanPending" cleanOnRoleRecycle="true" [CA]sizeInMB="5" /> #2
</LocalResources>
```

#1 and #2 should be cueballs

You can declare multiple local storage resources, as see in listing 15.3. It is important that the local file storage only be used for temporary, unimportant files. The local file store is not replicated or preserved in any way. If the instance fails and it is moved by the fabric controller to a new server the local file store is not preserved. That means that any files that were present will be lost. There is one time when the local file storage

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

will not be lost, and that is when the role is simply recycled, either as part of a service management event on your part, or when the appfabric is responding to a minor issue with your server. In these cases, if you have set the `cleanOnRoleRecyle` parameter to false, the current files will still be there when your instance comes back online.

Instance may only access their local storage. An instance may not access another instance's storage. You should use Azure blob storage if you need more than one instance to access the same storage area.

Now that we have defined our local storage we need to look at how we can access it and work with it.

15.4.2 Working with Local Storage

Working with files in local storage is just like working with normal files. When your role instance is started the agent creates a folder with the name you defined in configuration in a special area on the C: drive on your server. Rules are put in place to make sure the folder does not exceed its assigned quote for size. To start using it you simply need to get a handle for it.

To get a handle to our local storage area we need to use the `GetLocalResource` method. You will need to provide the name of the local resource you defined in the service definition file. This will return a `LocalResource` object.

```
public static LocalResource uploadFolder =
[CA]RoleEnvironment.GetLocalResource("FilesUploaded");
```

Once you have this reference to the local folder, you can start using it like a normal directory. To get the physical path, so you can check the directory contents, or write files to it, you would use the `uploadFolder` reference from above.

```
string rootPathName = uploadFolder.RootPath;
```

In the sample code provided with this book, there is a simple web role that is using local storage to store uploaded files. Please remember this is just a sample, and that you would not normally persist important files to the local store, considering its transient nature. When calling the RootPath method in the local development fabric, my storage is located:

```
C:\Users\brprince\AppData\Local\dftmp\s0\deployment(32)\res\deployment(32).AiA_15___Local_St
orage_post_pdc.LocalStorage_WebRole.0\directory\FilesUploaded\
```

When we publish this little application to the cloud, it returns the following path.

```
C:\Resources\directory\0c28d4f68a444ea380288bf8160006ae.LocalStorage_WebRole.FilesUploaded\
```

Listing 15.4 – Working with local file storage

```
protected void Page_Load(object sender, EventArgs e)
{
 litName.Text = uploadFolder.Name;                                                    #1
 litMaxSize.Text = uploadFolder.MaximumSizeInMegabytes.ToString();
 litRootPath.Text = uploadFolder.RootPath;
}

protected void cmdUpload1_Click(object sender, EventArgs e)
{
 if (FileUpload1.HasFile)
 {
  FileUpload1.SaveAs(uploadFolder.RootPath + FileUpload1.FileName);                   #2
 }

 litUploadedFilePath.Text = uploadFolder.RootPath + [CA]FileUpload1.FileName;
 litUploadedFileContents.Text =   [CA]System.IO.File.OpenText(uploadFolder.RootPath +
[CA]FileUpload1.FileName).ReadToEnd();                                                #3
}
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

#1 Path to our local folder
#2 The path to the file uploaded
#3 Reading the file like a normal file

#1,#2,#3 should be cueballs

Now that we know where I files will be stored, we can start working with the files. In the sample application we have a simple file upload control. #1 When the webpage is loaded we write out the local file path to the local storage folder we have been assigned. #2 Once the file is uploaded, we store it in the local storage and write out its filename and path. #3 We then write the file back out to the browser using normal file APIs to do so. This is a simple sample, and was designed to only work with text files, to keep things simple.

The local storage option is great for volatile local file access. Remember, this storage isn't durable, and may disappear on you. If you need durable storage, look at Azure storage or SQL Azure. If you need shared storage that is super-fast you should consider the AppFabric distributed cache. This is a peer-to-peer caching layer that can run on your roles and provide a shared in-memory cache for your instances to work with.

15.5 Summary

In this chapter we have looked at how you can process work in the background with the worker role in Azure. The worker role is an important tool to the cloud developer. It lets you do work when there isn't a user present. This can be because you have intentionally separated the background process from the user (in the case of the long running checkout process) or because you have broken your work into a discrete service that will process messages from a queue.

Worker roles scale just like web roles, but don't have a built in load balance like web roles do. You will usually aggregate them behind a queue, with each instance processing messages from the queue, thereby distributing the work across the group. This gives you the flexibility to increase or decrease the number of worker instances as the need arises.

It is quite possible to have an Azure application consist of only worker roles. You could have some on-premises transaction systems report system activity (each time a sale is made) to a queue in the cloud. The worker role would be there to pick up the report and merge the data into the reporting system. This allows you to keep the bulk of your application on-premises, while moving the compute intensive backend operations to the cloud. A more robust way of doing this would be to connect the on-premises system with the cloud system using the .NET Service Bus, which is discussed in chapter 17.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

16

Messaging with the Queue

In this chapter you will

- Learn how to loosely couple your system
- Distribute work to a group of service providers
- Learn how to use messaging

Queues are the third part of the Azure storage system. The concept of queues has been around a long time, and it is likely that you have worked with some technology related to queues already.

A common architectural goal during design is to produce a system that is tightly integrated, but also loosely coupled. Any system of size usually has several components. Whether these components are running in the same memory space, or on different boxes, we need them to work closely together. This is what is meant by 'tightly integrated.' We want these different components to work as a team to provide the value of the system in an easy and cohesive manner.

Many times, if your only goal is tight integration, you end up with a system where the components are tightly coupled as well. Tight coupling leads to a system that is very brittle, and one that responds poorly to changes. This makes it difficult to manage the system, and to extend it to future needs.

In a brittle system, a change in a component can cause that change to ripple through the whole system, requiring changes in many other components, not just the one that needed a change. This leads to a system that is difficult to understand, maintain, and troubleshoot.

The easiest way to provide a loosely coupled system is to provide a way for the components to talk with each other through messages. We want these messages to follow a "Tell, don't ask" approach. We should not ask an object for a bunch of data, do some work, and then give the results back to the object for recording. We should tell the object what we want it to do. We should do that at a component and system level as well. This approach helps us to create code that is well abstracted and compartmentalized.

Loose coupling also helps you isolate change from one component to another. An ecommerce website may be communicating with a backend ERP system. When the company chooses to change ERP vendors, the queue will act as buffer, keeping the change from rippling over the queue boundary. All the producer knows is to put messages in a certain format in the queue. The producer has no knowledge of the consuming system, and doesn't really care what happens.

The queue also becomes a pivot point for scaling. Later on we will talk about how you can monitor the length (some use the term depth) or a queue to determine if messages are being consumed fast enough. If not, you can scale out the number of consumers processing the messages, which reduces the number of waiting messages in the queue.

Be careful, however, that you don't put so much effort into loosely coupling your system that you end up building an overly complex monstrosity that becomes completely unmanageable. As with many things, balance is the key.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

16.1 - Decouple your system with messaging

There are many ways to decouple your system, but they usually center on messaging of some sort. One of the most common ways is to use queues in between the different parts of the system, or between completely different systems that must work together.

Queues have two ends. There is the producer end, where messages are put into the queue. This is usually represented as the bottom. The other end is the consumer end, where a consumer will pull messages off of the top of the queue.

Figure 16.1- A queue forms for ticket on the opening night of a new blockbuster movie. Movie goers enter (while wearing their fanboy outfits) at the bottom, or 'end of the line.' As the ticket booth (consumers) process the ticket requests, the movie goers move forward in the queue, until they are at the head of the line.

A queue is a FIFO structure; First in message in - First out message out. This contrasts with a stack, which is LIFO. LIFO stands for Last In- First Out. A real world sample of a queue is the line for tickets at the movie theater, on the opening day of new big movie. As people arrive, they stand at the end of the line. As the consumer (the ticket booth) completes sales, it works with the next person at the head of the line. As people buy their tickets, the line will move forward. At a real busy movie theater there may be many ticket booths consuming customers from the line. Management may open more ticket booths, based on the length of the line or based on how long people have to wait in the line. As the processing capacity of the ticket counter is increased, the theatre is able to sell tickets to more customers each minute. If the line was short at a particular time, the theater manager might close down ticket booths until only one or two are left open.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 16.2 - Producers place messages into the queue, while consumers get them out. Each queue can have multiple produces and consumers.

Your system can use a similar concept with a queue. As the producer side of your system (the shopping cart checkout process for example) produces messages, they are placed in the queue. The consumer side of the system (the ERP system that processes the orders, and charges credit cards) will pull messages off of the queue. In this way the two systems are tightly integrated, but loosely coupled, because of the queue in between.

Queues are one way in nature. A message goes in at the bottom, moves towards the top, and is eventually consumed. In order for the consumer message to communicate back to the producer, a separate process must be used. This could be a queue going in the other direction, but is usually some other mechanism, like a shared storage location.

There is an inherent order to a queue, but you cannot usually rely on queues for strict ordered delivery. In some scenarios this can be important. A consumer processing checkouts from an ecommerce website will not need the messages in a precise order, but perhaps a set of doctors' orders for a patient might be. It really won't matter which check out is processed first, as long as it is in a reasonable order. The order of what tests, drugs, and surgeries are performed on the patient is likely important. We will explore some ways of handling this situation in the context of Azure later in this chapter.

16.2 - Message Definition

Your Azure storage account can have many queues. At any time a queue can have many messages. Messages are the lifeblood of a queue system, and should represent the messages that a producer is telling to a consumer. You can think of a queue as having a name, some properties, and a collection of ordered messages.

In Azure, messages are limited to 8KB in size. This low limit is designed for performance and scalability reasons. If a message could be up to 1GB in size, then writing to and reading from the queue would take a long time. This would also make it hard for the queue to respond quickly when there are many different consumers reading messages off of the top of the queue.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Because of this limit, most Azure queue messages will follow a 'work ticket' pattern. The message will usually not contain the data needed by the consumer itself. Instead, the message will contain a pointer of some sort to the real work needed to be done.

Figure 16.3- Work tickets are used in queues to tell the consumer what work needs to be done. This keeps the messages light, and keeps the queue scalable and performant. The work ticket is usually a pointer to where the real work is.

For example, following along with figure 16.3, a queue that contains messages for video compression won't include the actual video that needs to be compressed. The producer will store the video in a shared storage location (1), perhaps a blog container, or a table. Once the video is stored, the producer will then place a message in the queue with the name of the blob that needs to be compressed (2). There will likely be other jobs in the queue as well.

The consumer will then pick up the work ticket, fetch the proper video from blob storage, compress the video (3), and then store the new video back in blob storage (4). Sometimes the process ends there, with the original producer being smart enough to look in the blob storage for the compressed version of the video, or perhaps a flag in a database is flipped to show that the processing has been completed.

The content of a message is always stored as a string. The string must be in a format that can be included in an XML message, and be UTF-8 encoded. This is because a message is returned from the queue in an XML format, with your content as part of that XML. It is possible to store binary data, but you need to serialize and de-serialize the data yourself. Keep in mind that when you are de-serializing, the content coming out of the message will be base64 encoded.

Listing 16.1- A message in its native XML format.

```
<QueueMessagesList>
 <QueueMessage>
  <MessageId>20be3f61-b70f-47c7-9f87-abbf4c71182b</MessageId>          A
  <InsertionTime>Fri, 07 Aug 2009 00:58:41 GMT</InsertionTime>         B
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
    <ExpirationTime>Fri, 14 Aug 2009 00:58:41 GMT</ExpirationTime>
    <PopReceipt>NzBjM2QwZDYtMzFjMC00MGVhLThiOTEtZ[CA]
                DcxODBlZDczYjA4</PopReceipt>
    <TimeNextVisible>Fri, 07 Aug 2009 00:59:16 GMT</TimeNextVisible>
    <MessageText>PHNhbXBsZT5zYW1wbGUgbWVzc2FnZTwvc2FtcGxlPg==</MessageText>
  </QueueMessage>
</QueueMessagesList>
```

(A) and (B) should be replaced with cueballs.

The content of the message is not the only part of the message that you may want to work with. Every message has the several important properties.

The id property (A) is assigned by the storage system, and is unique. This is the only way to truly uniquely differentiate messages from each other, since several messages could contain the same content.

A message also includes the time and date (B) the message was inserted into the queue. It can be handy to see how long the message has been waiting to be processed. For example, you might use this to determine if the messages are becoming stale in the queue. This is also used by the storage service to determine if your message should be garbage collected or not. Any message that is about a week old in any queue will be collected, and discarded.

16.3 - What is a Queue?

Now that we have discussed what messages are, we are ready to discuss what contains them, the queue itself. The queue is the mechanism that holds the messages, in a rough order, until they are consumed. The queue is replicated in triplicate throughout the storage service, just like tables and blobs. This is for redundancy and performance reasons.

Queues can be created in a static manner, perhaps as part of deploying your application. They can also be created and destroyed in a dynamic manner. This is handy when you need a way to organize and direct messages in different directions based on real time data or user needs.

Each queue can have an unlimited number of messages. The only real limit is how fast you can process the messages, and if you can do so before they are garbage collected after one week's time.

Since a queue's name appears in the URI for the REST request, it needs to follow the constraints that DNS names have.

- It must start with a letter or number, and may contain only letters, numbers, and the dash (-) character.
- The first and last letters in the queue name must be alphanumeric. The dash (-) character may not be the first or last letter.
- All letters in a queue name must be lowercase. This is the requirement that gets me everytime.
- A queue name must be from 3 through 63 characters long.

A queue also has a set of metadata associated with it. This metadata can be up to 8KB, and is a simple collection of name/value key pairs. This metadata can help you track and manage your queues. While the name of a queue can help you understand what the use of the queue is, the metadata can be useful in a more dynamic situation. The name of the queue might be the customer number that the queue is related to, but you could store the customer's service level (tin, silver, molybdenum, and gold) as a piece of meta data. This meta data then lives with the queue, and can be accessed by any producer or consumer of the queue.

Queues are both reliable and a persistent way to store and move messages. They are reliable in that you should never lose a message. We will look at how this works in the section on the message lifecycle. Queues are also strict in how they persist your messages. If a server goes down, the messages are not lost, they remain in the queue. This would differ from a purely memory based system, in which all of the messages would be lost if the server were to have a failure.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

16.4 - Storage Client and the REST API

There are two basic ways to interact with a queue, and its messages. The first is the Storage Client library that ships with the Azure SDK. The Storage Client library is a robust API to use when accessing any part of Windows Azure storage from .NET. This cleanly separates you from having to write out your own REST commands, which can be fun, but not entirely a productive way to solve a problem.

The other mechanism for interacting with queues is to use the REST API directly. You can create and consume REST messages in any way that you want. While this is a little more work, it is worth your time in learning how the REST API works under the covers of any library, so that you understand more fully how the storage system works. The REST entry point will be the key way to access Azure storage when you don't have a handy API lying around like the Storage Client. Microsoft and several open source teams are working to build similar tools for every platform, including Python and PHP. All of these libraries use the REST protocols under the hood.

Each call into the REST API has a request header that includes some basic information. The header needs to include which version of the service you are targeting, the date and time of the request, and the authorization header.

Listing 16.2 - A sample REST request header

```
POST /queue21b3c6dfe8626450880b9e16c70e2425e/messages?timeout=30 HTTP/1.1
x-ms-date: Fri, 07 Aug 2009 01:26:38 GMT
Authorization: SharedKey hsslog:Iow8eYFGeodLGqXrgbEcwDuA+aNOR0emEC9uy3Vnggg=
Host: hsslog.queue.core.windows.net
Content-Length: 80
Expect: 100-continue
```

The service version header is useful to protect against an update to the queue service disrupting your system. You can force your requests to be processed by a specific version of the storage service, allowing you to control when you support and leverage new features in a newer version of the service. If you omit the version header, your request will be routed to the default version of the service.

A queue cannot be made public, or anonymous. Every operation against a queue must be authenticated with the shared key method. Constructing the authorization header for queue requests is the same as for blobs and tables.

Now that we know how to forge the header, or the envelope, for our message, let's look at how to send commands to the queue.

16.5 - Basic queue interactions

To learn how to use the basic queue API operations, we are going to build our own queue browser. This little tool will help us debug any system we are building, by helping us look at the queues that are being used, and see how they are working.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 16.4 - A screen shot of the simple queue browser we will build in this chapter. It is important to note that the authors, while charming, are not graphic designers or UX specialists. This tool will act as a vehicle for understanding the basics of working with queues.

We will be focusing on the simple code that is needed for each of the operations of the browser. We aren't going to focus on how WPF works, or the best application architecture for this little application. This is meant to be 'me-ware', something that works for you, and doesn't have to work for anyone else. You should use it as a harness to play with the different APIs and learn how they work.

There are several basic queue operations that you will need to be able to work with. The first one that we are going to look at is a method that will tell you what queues exist in your account. You may not always need this. Usually, you will know what the queue for your application is, and provision it for yourself.

To get a list of the queues available to you, you need to first connect to the queue service, and then call the method.

Listing 16.3 - Connecting to the queue service, and getting a list of queues

```
private CloudQueueClient Qsvc;
private IEnumerable<CloudQueue> qList;

CloudStorageAccount storageAccount =
[CA]CloudStorageAccount.FromConfigurationSetting("DataConnectionString");

Qsvc = storageAccount.CreateCloudQueueClient();          A

qList = Qsvc.ListQueues();
```

Replace A with a cue ball in the next paragraph.

You will use something like line (A) quite often. This creates a connection to the service, similar to how you create a connection object to a database. You will want to create this once for a block of code, and hold it in memory so that you aren't always reconnecting to the service. In our little application, we create this object in the constructor for the window itself, and store it in a variable that is at the form level. This makes it available to every method in the form, and saves us the trouble of having to continuously recreate the object.

The `CloudStorageAccount` servers as a factory that creates service objects that represent the Azure queue storage service, called the `CloudQueueClient`. There are several ways to create the

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

`CloudQueueClient`. The most common approach is to use it as we did in listing 16.3, by using the `FromConfigurationSetting` method. This looks into your configuration and sets up all of the URIs, usernames, account names, etc. This is better than having to set four or five different parameters when you are 'newing' up the connection.

Once you have a handle to the queue service, you can simply call `ListQueues`. This does not just return a list of strings as you might expect, but instead returns a collection of queue objects. Each queue object represents a queue in your account, and is fully usable.

Listing 16.4 - A sample REST quest for getting a list of queues.

```
GET http://hsslog.queue.core.windows.net/?comp=list&maxresults=50&timeout=30
```

The equivalent call in REST would be like figure 16.4. You can see that it is a simple `GET`.

If you don't have any queues, you will get back an empty collection. Our next step should be to create a queue.

Listing 16.5 - Creating a message queue

```
CloudQueue q = Qsvc.GetQueueReference("newordersqueue");          A
q.CreateIfNotExist();                                             B
```

A and B in the next para should be cueballs.

In (A) we create a `CloudQueue` object. This is an empty object, and this line does not actually connect to the service. It is merely an empty handle that doesn't point to anything. Then the `CreateIfNotExist` method is called (B) on the queue object you just created. This will check if a queue with that name exists, and if it doesn't, it will create one. This is very handy.

You can check if a queue exists before you try to create it by using `q.DoesQueueExist.` This method will return a boolean value, telling you if the queue exists or not.

Our next step is to attach some metadata to the queue. We can store up to 8KB of data in the property bag for each queue. You might want to use this to track some core data about the queue, or perhaps some core metrics on how often it should be polled.

Figure 16.6 - Add and removing queue properties

```
CloudQueue q = Qsvc.GetQueueReference("newordersqueue");
q.Metadata.Add("ProjectName", "ElectronReintroductionPhasing");        A
q.Metadata.Keys["BadKeyNoDoughnut"].Remove();                          B
q.SetMetadata();                                                       C
```

A,B,C should be Cueballs.

The metadata for a queue is attached as a property on the `CloudQueue` object. You work with it like any other name value collection. In A we are adding a new entry to the metadata called 'ProjectName', with a value of 'ElectronReintroductionPhasing'. This new entry will not be saved back to the queue service though until we call `SetMetaData` on line C. This connects to the service and uploads our metadata for the queue in the cloud.

You can remove existing properties from the bag if you no longer need them. Line B shows how easy it is for us to remove the 'BadKeyNoDoughnut' from use. Removing an item from the metadata collection must also be followed by a `SetMetaData` call to persist the changes to the cloud.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 16.5 - Displaying the metadata we set on a queue. You can attach up to 8KB of metadata to a queue. This can help in managing that queue, such as what the back off pace rate should be.

Now that we have created a queue, and set its metadata, let's look at how to delete a queue. It is good practice to clear a queue before you delete it. This removes all of the messages from the queue. The clear queue method is handy for resetting a system, or clearing out some poison messages that may have stopped up the flow.

Listing 16.7 – Clearing and deleting a queue is simple

```
CloudQueue q = Qsvc.GetQueueReference("newordersqueue");
q.Clear();
q.Delete();
q = null;
```

Deleting a queue is as simple as that, when using the client library. The equivalent REST call would be

Listing 16.8 - REST call to delete a queue

```
DELETE http://hsslog.queue.core.windows.net/newordersqueue?timeout=30
```

Being able to create and destroy queues in a single line of code make them simple objects to work with. In the past using a queue in your system would require days if not weeks of installing several queue servers (for redundancy purposes). They would also require a lot of care and feeding. Queues in the cloud are much easier to work with, and don't require any grooming or maintenance. The real power of queues however are the messages that flow through them, which we will delve into in the next section.

16.6 - Working with messages

Now that we know how to work with queues, let's look at how you can work with messages. As we mentioned above, queues are a FIFO structure, similar to a line at the movie theater. When we 'put' messages on the queue, the new message is placed onto the bottom or end of the queue. When we 'get' a message from the queue, it is from the top or front of the queue.

Listing 16.9 – Creating a new message, then adding it to a queue

```
CloudQueue q = Qsvc.GetQueueReference("newordersqueue");
CloudQueueMessage theNewMessage = new CloudQueueMessage("cart:31415");    A
q.AddMessage(theNewMessage);                                              B
```

A and B should be cueballs

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

To put add a message to the queue we need a reference to the queue, as we did in the samples above. Once you have the queue reference you can call the `AddMessage` method, B and pass in a `CloudQueueMessage` object. Creating the message is a simple affair; in this case we are simply passing in text that will be the content of the message. You can also pass in a byte array if you are passing some serialized binary data. Remember that the content of each message is limited to 8KB in size.

Figure 16.10 - An example of putting a message onto a queue with REST

```
POST /my-special-queue/messages?timeout=30 HTTP/1.1
x-ms-date: Fri, 07 Aug 2009 01:49:25 GMT
Authorization: SharedKey hsslog:3oJPdtrUK47gMONZ+DupUiSpHfwrmasdnT5nJpGpszg=
Host: hsslog.queue.core.windows.net
Content-Length: 80
Expect: 100-continue

<QueueMessage><MessageText>cart:31415</MessageText></QueueMessage>
```

In this example, we are adding a message with the order number that can be found in the related Azure table. The consumer will pick up the message, unwrap the content, and process the cart numbered 31415. I bet their shopping cart is filled with pie and pie related accessories.

Before we show you how you will get a message, we want to talk about peeking. Peeking is a way to get the content of a message from the queue, without taking the message off of the queue. This leaves the message still on the queue, so someone else can grab it. Many people peek at messages to determine if they want to process the message or not, or want to determine how they want to process the message.

Figure 16.11 - Peeking at messages in the queue can save time

```
CloudQueueMessage m = q.PeekMessage();                    A

private IEnumerable< CloudQueueMessage > mList;
mList = q.PeekMessages(10);                               B
```

A and B should be cueballs.

Line A shows how easy it is to peek at a message. Calling the `PeekMessage` method returns a single message, the one at the front of the queue. You can peek at more than one message by calling `PeekMessages`. You will need to provide the number of messages you want returned. In the example above, we asked for 10 messages.

Now that we have peeked at the messages, we are ready to get them. You don't have to peek a message before getting it, and many times you won't use peek at all.

```
private CloudQueueMessage currentMsg;
currentMsg = q.GetMessage();
```

If you already have a reference to your queue, then it is as simple as calling `GetMessage`. There is one override that lets you determine the visibility timeout of the get. We will discuss the lifecycle of a message in the next section.

```
string s = currentMsg.AsString;
```

Once you have a message, you can use the `AsString` or `CAsBytes` properties to access the contents of the message. This is the meat of the message, and the part you are most likely interested in.

Once we have processed a message, we will want to delete it. This takes it off of the queue. Deleting a message is as easy as getting it.

```
q.DeleteMessage(currentMsg);
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

To delete a message, we need to pass the message back into the `DeleteMessage` method of the queue object. You can also do it easily with REST. You can only delete one message at a time.

Listing 16.12 – Deleting a message with REST

```
DELETE http://hsslog.queue.core.windows.net/my-special[CA]
-queue/messages/f5104ff3-260c-48c8- [CA]
9c35-cd8ffe3d5ace?popreceipt=AgAAAAEAAAAAAAAA1vkQXwEXygE%3d&timeout=30
```

One part that we didn't dive into yet is the lifecycle of a message. What really happens when a message is retrieved? How does the queue keep the same message from being picked up by several consumers at the same time? These are important questions for a queue service. Never losing a message (known as durability) is critical to a queue.

16.7 - Message lifecycle

There is a property that is the part of every message called the visibility timeout. When a message is pulled from the queue, it is temporarily marked as invisible. The consumer is also given a receipt (called the pop receipt) that is unique to that `GetMessage` operation. The length of invisibility can be controlled by the consumer, but defaults to 30 seconds. The timeout cannot be longer than two hours however.

While a message is invisible, it will not be given out in response to new `GetMessage` operations.

FIGURE 16.6: Two consumers getting messages from a queue. A message is marked invisible when a getmessage operation is performed. Just like a cloak of invisibility. This effect does timeout after a period of time however.

#1 and #2 need to be cueballs in the next section

As an example, let's say a producer has placed four messages in the queue. We have two consumers that will be reading messages out of the queue. Consumer 1 gets a message, and that message is marked invisible (as seen in figure 16.14 in step #1). Seconds later, consumer 2 performs a get operation as well. Since message 1 is invisible, the queue responds with message 2 (step #2).

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Not long thereafter, consumer 1 finishes processing message 1, and performs a delete operation on the message. As part of the delete operation, the queue checks the pop receipt the consumer provides when it passes in the message. This is to make sure it is the most recent 'reader' of the message in question. The receipts match in this case, and the message is deleted. Consumer 1 then does an immediate read, and gets message 3. Consumer 1 fails to complete processing within the invisibility window, and fails to delete message 3 in time. It becomes visible again.

Just at that time, consumer 2 deletes message 2, and does a get. The queue responds with message three, because it is visible again. While consumer 2 is processing message 3, consumer 1 does finally finish processing message 3, and tries to delete. Only this time the pop receipt consumer 1 has doesn't match the most recently provided pop receipt, which was given to consumer 2 when message 3 was handed out for a second time.

Because the pop receipt doesn't match, an error is thrown, and the message is not deleted. You will likely see a `400 - Bad Request` error when this happens. The inner exception details will say it is because there aren't any messages with a matching pop receipt.

You can set the length of the visibility timeout when you get the message. This lets you determine the proper length of the timeout for each individual message. When you pick this number, you want to balance the expected processing time and how long it will take for the system to recover from an error in processing. If the timeout is too long, it will take a long time for the system to recover a lost message. If the timeout is too short, too many of your messages will be repeatedly reprocessed.

This leads us to an important aspect of queues in general, but specifically the Azure queue system. The guarantee is worded as promising that every message will be processed, *at least once*. You can see this 'at least once' business in the above scenario. Because consumer 1 failed to delete the message in time, the queue handed it out to another consumer. The queue has to assume the original consumer has failed in some way. This is very useful because it provides a way for your system to take a hit (a server going down), and keep on going. Cloud architecture has this concept of being able to plan on failure, and making that central to the structure of the system. Queues provide that capability quite nicely.

The downside is that it is very possible that a consumer doesn't crash, but just takes longer to process the message than intended. In this case you need to make sure that your processing code is either idempotent, or checks before processing each message to make sure it isn't a duplicate scenario. Because the message being re-processed is actually the same message, its id property is the same. This makes it easy to check a history table or perhaps the status of a related order before processing starts.

This little complexity might make you think about deleting a message as soon as you receive it. This is very dangerous and unwise. Because there will be failure along the way, and when that happens, that message will be lost forever. The goal of our messaging systems is to make sure we never lose a message. No matter how small or large, you never want to lose an order, or a set of instructions, or anything else you might be processing.

To avoid complexity, it is best to make sure your processing code is idempotent. Idempotent simply means that the process can be executed several times, and the system will result in the same state. Perhaps we are working with a piece of software that tracks dog food delivery. When the food is actually delivered to the physical address, the handheld computer sends a message to our queue in the cloud. The software uploads the physical signature of the recipient to blob storage, and submits an order delivered message to the queue. The message contains the time of delivery and the actual order number, which happens to also be the filename of the signature in blob storage.

When this message is processed, the consumer copies the signature image to permanent storage, with the proper filename, and makes the order as delivered in the package tracking system database.

If this message were to be processed several times, there would be no detriment to the system. The signature file would be overwritten with the same file. The order status is already set to delivered, so we would just be setting its status to delivered again. Using the same delivery time doesn't change the overall state of the system. This is the best way to handle the processing of queue messages, but it is not always possible. The next section will discuss some common queue processing patterns, and some of them deal with working around this issue.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

16.8- Patterns for message processing

As simple as queues are, they can prove valuable in a lot of complex scenarios. This section is going to focus on some common approaches developers tend to use with queues.

16.8.1 - shared counters

You might run into a scenario when a piece of work is broken into many small pieces, and you need to make sure all of those small pieces are completed before you move onto the next step in your process. Sometimes these pieces are subsets of the main problem.

This is called Single Instruction, Multiple Data. The same processing will be performed to each piece of data, but each piece is a subset of the whole. Consider working on an image. If I break the image into 100 pieces, and apply the same process to each piece, I need to know for sure that all 100 pieces are completed before I can stitch them back into the larger picture.

If we just break the image into 100 pieces, and throw them in the queue, it can be quite difficult to know for sure when all of the 100 units of work have been completed. This has to do with the visibility timeout, and the non-deterministic nature of queues. At any point I could check the length of the queue using `q.ApproximateMessageCount`.

Listing 16.13 – How many messages are waiting?

```
q.RetrieveApproximateMessageCount();
if (q.ApproximateMessageCount != null)
   int remainingMsg = q.ApproximateMessageCount;
```

You must call `RetrieveApproximateMessageCount` before you call the `ApproximateMessageCount` property. This fetches the information into the property from the queue in the cloud. This property returns an approximate count of the items in the queue, not an exact count. This is because of two reasons. The first is that the queue is running in triplicate, and an add operation might have completed on one instance, but not the other two. This would lead to an inconsistent result. The second reason is that you might get a zero back from your check, only to have an invisible message turn visible again when it's timeout expires. Then you would have a message in the queue you didn't know about.

The crux of the problem is that you need a deterministic way to know for sure that all 100 pieces have been processed. One way to do this is to use a shared counter, perhaps in an Azure table. When the processing starts, a table is made with a counter set to 0. As items are submitted into the queue by the producer, the counter is incremented. As the work is completed in the consumer, and the messages are deleted, the counter is decremented.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 16.7 – Using a shared counter is one way to deterministically track how many messages have been processed. This is a good approach if you have a specific of messages to process, and you need a precise count.

There is one small flaw this approach suffers from, and it is a flaw all shared counters have. It is possible that you run into a concurrency problem. If one process reads the counter, adds one to it and then writes it back to the table while another process is doing the same thing, they could end up overwriting each other, resulting in losing track of the count. In order to fix this you need to use locking on the counter. Another solution, which is used in eventually consistent scenarios, is to check that the counter hasn't changed as you write your update to make sure it wasn't changed while you weren't looking.

This approach will give you a simple count as to the progress of the work. What if you want to know which pieces are done or not? No problem. We can do this with a small change to the above approach.

Instead of writing to a shared counter on each put operation, create a new record in the shared table. This will result in one record per queue message. As they are processed and the queue messages are deleted, the corresponding row should be deleted in the table. Another option would to mark a property of the row in the table as complete or store a completed time and date for performance tracking.

In either of these ways, with the shared counter, or the shared message tracking table, you can know with a simple query if all of the work has been completed or not. You should think about wiring up a management portal that monitors the counter or table to show the progress to an administrator.

16.8.2 - work complete receipt

The above scenario works when you can control the producer, and you have a closed loop. What if you don't own the producer, or there are too many for you to make them also manage a counter? In this scenario you can use a return receipt, or a work complete receipt.

In this approach, as work is completed, a message is sent through a separate channel, perhaps another queue, back to the producer. This alerts the producer that the work is done. This is common in scenarios where the process takes a really long time to complete, and the producer wants an asynchronous notification when the work is done.

Instead of using a return queue, I have also had the consumer call a small notification web service on the producer side, with a simple message regarding the status of the work. This makes the consumer an active part of the process, and removes the need for the producer to monitor a queue, and become a consumer themselves.

16.8.3 - asymmetric queues vs. symmetric queues

Queues are decidedly one way. They are a way for one or more producers to communicate with one or more consumers, but not the other way around. Using one queue in this manner is an asymmetric queue. Generally, in an asymmetric queue, the producer finds out about the work being completed in a passive way. The new file happens to be in the right place when the user hits refresh, or the customer receives an email when the order is shipped, or any number of other scenarios.

As was discussed in the above section about work complete receipts, sometimes using symmetric queues can be useful. This makes the response from the consumer back to the producer an active one. Using a queue do this does helps decouple the two halves of the system, but it can lead to too much complexity. This also turns the original producer into a consumer in its own right, which can be hard if the original producer was a web site. Since web sites only respond to outside requests (a person performs a GET or POST with a web browser to view the catalog), it doesn't have a running process to proactively read the queue and respond to it.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 16.8 – A symmetric queue is when two queues are used together to allow for two way communication between a set of consumers and a set of producers. This keeps both sets of systems loosely coupled. The inbound queue is on the top, and the outbound queue is on the bottom.

#1, #2, #3, #4 should be cueballs.

In figure 16.x above, we can see two queues connecting the consumers and the producers together. The top queue is the inbound queue, sending messages to the consumers. The bottom queue is the outbound queue, bringing messages back to the consumers. A typical message pattern would be for the producers to send a message to the consumer by putting it in the inbound queue (#1). The consumer picks up the message (#2), does some critical work in your business like ordering pudding, and submits a confirmation message back to the producer by putting it in the outbound queue (#3). The producer finally receives the confirmation message back in step #4.

16.8.4 - truncated exponential back off

It is quite common for a worker role to have an infinite loop that polls the queue, processes the work, and pauses for a period of time if the queue is empty.

Listing 16.14 – The typical infinite loop to poll a queue

```
while (true)                                          A
{
   CloudQueueMessage msg = queue.GetMessage();        B
   if (msg != null)                                   C
   {
      string messageContent = msg.AsString;
      DoWork(messageContent);
      queue.DeleteMessage(msg);                       D
   }
   else
   {
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
        System.Threading.Thread.Sleep(1000);          E
    }
}
```

#A, #B, #C, #D, #E should be replaced with cueballs.

Listing 16.14 shows the typical infinite polling loop. #A starts the loop off, looping until true equals false, which shouldn't ever happen, and if it does we are all in for a world of hurt. Line #B grabs the first message off of the queue. C checks to see if this message is NULL. If there aren't any messages on the queue, the #B would return a NULL message. If there is a message, then we process it, then not deleting it (#D) until the real work is fully completed.

If there wasn't a new message retrieved, then the loop sleeps for a period of time (#E), and then tries again.

Sometimes there is a concern that the queue is polled too much. If this is a concern for you, you can dynamically change the wait time in the bottom of the loop. A common algorithm used in networking is an algorithm called truncated exponential back off. Simply put, each time a queue check doesn't return a message the loop delay is extended exponentially, until a certain ceiling is reached. If the check does return a message, the loop delay is decreased, either back to the lowest setting, or to the next lowest setting in the progression.

Figure 16.15 - Adjusting the delay in a polling loop

```
while (true)
{
      CloudQueueMessage msg = q.GetMessage();
      if (msg != null)
      {
         // do your work here on the message
         q.DeleteMessage(msg);

         if (useGradualDecrease)                                    A
              if (currentInterval > intervalFloor)                  A
                   currentInterval = currentInterval / 2;           A
              else                                                  A
                   currentInterval = intervalFloor;                 A
         else                                                       A
              currentInterval = intervalFloor;                      A

      }
      else
      {
           if (currentInterval < intervalCeiling)                   B
           currentInterval = currentInterval * 2;                   B

           Thread.Sleep(currentInterval * 1000);
      }
}
```

#A and #B should be cueballs in the next section

An example would be that the start value for the delay loop is 2 seconds. After an empty poll, the length is doubled to 4 seconds. Successive empty checks result in the time delay increasing to 8, 16, 32, 64, 128, 256, and 512 seconds. You can see the code to make this progression at #B. At 512 seconds, the counter reaches the maximum set for the system, and does not rise above 512 seconds. After some time a message appears. The system processes the message, and then checks the length of the queue. Since there was activity on the queue the delay setting is set to the next step down the ladder, to 256 seconds at line #A. The code also supports an immediate drop to the floor interval, if you want an aggressive increase in the rate of polling.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
Output
Show output from: Debug
'QueueBrowser.vshost.exe' (Managed (v4.0.21006)): Loaded 'C:\Wi
'QueueBrowser.vshost.exe' (Managed (v4.0.21006)): Loaded 'C:\Wi
QueueBrowser.vshost.exe Information: 0 : Interval '4'
Chance: '1 of 3'
QueueBrowser.vshost.exe Information: 0 : Dequeued '*****'
QueueBrowser.vshost.exe Information: 0 : Deleted '*****'
QueueBrowser.vshost.exe Information: 0 : Interval '2'
Chance: '2 of 3'
QueueBrowser.vshost.exe Information: 0 : Interval '4'
Chance: '2 of 3'
QueueBrowser.vshost.exe Information: 0 : Interval '8'
Chance: '2 of 3'
QueueBrowser.vshost.exe Information: 0 : Interval '16'
```

Figure 16.9 – The output from running our backoff polling algorithm. We can see the polling interval exponentially increase from 2 to 4 to 8 to 16 as the queue continues to be empty.

Figure 16.9 shows the backoff polling in action. The sample code will randomly put messages in the monitored queue. There is a one in three chance in doing so. The interval started at 2, and was immediately bumped to 4. Then a message was processed, so it was dialed back down to 2. Then there was a succession of empty calls, leading the code to rapidly increase the polling interval all the way to 16 seconds between checks.

16.8.5 - queue creation on startup

One advantage of the Azure storage system is the easy creation and deletion of queues. A common trend is to inspect the storage system on startup of the system to determine if the needed entities (blog accounts, tables, and queues) exist. If they don't exist, they are created on the spot. This makes it really easy to deploy a system, knowing that it will self-provision the storage resources it needs during startup.

A possible drawback to this approach is forgetting to manage the state of these resources carefully. If you are rolling out an upgrade to the system, and the initialization steps clear the work queues and other storage entities, it is possible that you could lose valuable data or work in progress. Make sure that you test both new deployments and upgrades to your system in a safe environment before relying on the self-provisioning code. The automatic provisioning may also impact performance on system startup, because it will be busy checking the infrastructure, and configuring things as needed.

16.8.6 - dynamic queues vs. static queues

Most queues in your applications will be static in nature. The design of your system will require whatever queues it needs, and these will be provisioned when the application is deployed, and left to run as is.

There is a capability you can take advantage of, that you normally can't do with other queuing technologies, and that is the dynamic creation of queues. Imagine that you can create a new custom queue for each new order that is being processed for by system. This helps your application dynamically scale very well, and also helps you separate different concerns in your system.

Perhaps you are building a system for a Value Added Network that manages the flow of purchase order messages from vendors and suppliers. All day long you are signing up new vendors and suppliers, and occasionally some stop using your service. A great way to automate the provisioning of the queues that are going to be needed for each customer that signs up is to dynamically create the queues and infrastructure needed as they sign up, and are approved as users of the system.

You want to pay careful attention to the state of each customer, and make sure that any leftover data is cleaned up, and entities are de-provisioned as customers stop using your service. You don't want to have to pay for unneeded infrastructure that is forgotten and left lying around.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Another scenario would be even more short term than the one above. Think back to the image processing scenario. Since each image needs a queue to manage the breakdown and processing of each image, you could dynamically create a queue for each new image job that is submitted. When the processing is complete, the queue can be torn down.

16.8.7- ordered delivery

Some scenarios just require guaranteed ordered delivery, such as some EDI scenarios. The queue, as it is, does not support ordered delivery. There are several approaches for adding this capability on top of the normal queue service.

The simplest approach hinges on the series of messages having a header message that declares the length of the series, or the existence of a trailer message that tells the system when the last message has been received.

The basic approach would be for a process to monitor the normal queue, and pull the messages off, and store them in a temporary Azure table. The table should have an integer property that stores the order of that message in the series. Once the defined number of messages are received, or when the trailer message is received, the messages can be properly ordered (usually by some element present in the message themselves) and then sent onto the final system that needed the messages ordered properly.

An optimization would be to have the process that is ordering the messages to start trickling them on to the final destination, when it knows that it has some of the messages already in order. For example, if messages arrive in the order 1, 2, 3, 5, 6, 4. Then the message collator can almost immediately send messages 1, 2, and 3. The collator would have to wait for message 4 to arrive until it could forward messages 4, 5, and 6.

16.8.8 - Long Queues

Most of the queues we have discussed until now have been in the form of an immediately serviced queue, where there are one or more consumers actively processing the messages in the queue.

There are times when it is important to have a long queue in play. This would be a queue that might receive messages all day long, without an active consumer. The messages would be processed in a batch manner later that evening when the consumer comes online. You might have an application in the field that produces messages into the cloud during the day. During the evening, a backend system comes online, processes all of the messages until the queue is emptied, and then goes back offline. This will be a useful scenario when the consuming system is not always available to process the messages you are holding in the queue.

16.8.9 - Dynamically scaling to meet queue demand

The promise of queues is that the processing of messages is decoupled from the production of those messages. By decoupling the backend, you are free to scale the backend to meet the demands of the number of messages in the queue.

In a traditional environment, the number of producers is fairly static. The pool of consumers can be scaled, but it requires all the work of buying an additional server, provisioning it, and deploying it to the pool. This can be time consuming, and you are likely to miss the spike in demand while you are waiting for hardware to be shipped from your vendor.

The promise of cloud computing is the true dynamic allocation of resources to your computing needs. A management tool can be deployed that monitors the length of the queue in question. The tool can then dynamically create additional consumers (by increasing the number of deployed worker role instances using the service management API) based on the length of the queue. The management tool should define a cap on the number of instances that can be created, and also rules as to when instances should be created or destroyed.

For example, you might define the minimum number of instances at zero, with a maximum of five instances. The rule of thumb will be one instance per 20 messages in the queue. The management tool will need to determine the length of the queue on a regular basis, perhaps every three minutes. Your rules should always allow for a little buffer in capacity. If you run too close to actual demand, the slight delay it takes (a few minutes) to bring on new instances can have a deleterious effect on the performance of your system.

Because Azure is billed based on the number of active instances, and not the actual use of the CPU, this can be a way to not only meet spikes in demand with grace, but also minimize the costs of the solution.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

16.9 - Queue Performance

Performance is critical to every part of Azure, and queues are no exception. Each queue, like the rest of the Azure storage services, exists as three instances. Each instance is protected by different fault and update domains. This strategy protects your queue from completely failing when a switch goes down, or a patch is rolled out.

As the demand for a queue increases, the storage fabric will start serving the requests out of a memory cache. This dramatically increases the performance of the queue, and reduces the latency of using the queue.

16.10 – Summary

Azure queues are a great way to break your system into pieces that still work together to get the work done, and they are easy to work with. They don't have to connect Azure web and worker roles together. They can be used to help cloud applications, mobile applications, and enterprise applications communicate together. Instead of a mobile application needing to punch through a firewall to submit a new repair ticket, it can submit the ticket into the cloud, where it can be picked up later by the on-premises system.

Queues are often the only onramp to a backend system. In this role they act as the service endpoint for the capability the backend system represents. We showed how simple it is to create and manage queues.

While queues are pivotal in leveraging the dynamic power of Azure with their ability to act as load balancers for worker role instances, the real power is in the messages in the system. Messages are the lifeblood of a decoupled system, allowing different components and subsystems to work together without being required to understand dirty implementation details of external parts.

We also explored several patterns that can be used in the design of your application. The two most common are to manage the polling of the queue with a back off polling algorithm, and to dynamically provision the proper amount of consumers based on the depth of the queue.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

17

Connecting in the cloud with AppFabric

This chapter covers:

- How to secure your services with ACS
- What the Service Bus is
- How to connect to your service from anywhere

Windows Azure Platform AppFabric (hereafter known as AppFabric) is an important piece of the Windows Azure puzzle. It is part of the larger Azure ecosystem and provides some fundamental features for working with hybrid applications. It provides two major pieces, how to secure REST services, and how to connect them together.

AppFabric is a really big topic, one that deserves its own book. Whenever you start talking about security you get into a really long conversation. The same thing goes for service busses. The reason is that both of these topics have a lot of local terminology the average developer is likely not familiar with.

Our goal in this chapter is to give you a good enough look at AppFabric to understand the core scenarios it can be used for, and understand enough to go dive into a detailed book on your own. We will be visiting the two key services (Access Control Service and Service Bus) with a nice example. You will need to know a little about WCF. Don't worry though, WCF is not scary.

17.1 The road AppFabric has travelled

AppFabric is arguable the most mature part of Windows Azure, at least if you measure by when it has been publicly available, if not broadly announced. AppFabric started life as BizTalk Services. It was seen as a complimentary cloud offering to BizTalk Server. BizTalk is a very high end enterprise grade messaging and integration platform, and indeed the services fit into that portfolio very well. Some joke that it was called BizTalk Services as a clever way to keep it a secret because BizTalk is one of the most underestimated products Microsoft has. Just talk to a BizTalk developer. They are very passionate about it and always want it to tell everyone about.

When Windows Azure was announced at PDC 2008 the services were renamed to '.NET Services.' Over the following year there was a broad push to get developers to work with the services and put the SDK through its paces. Out of that year of real world testing came a lot of changes. When Windows Azure went live in early 2010 the services were renamed to Windows Azure Platform AppFabric to fall more in line with another product, and tie it more closely to the Windows Azure platform. Some people were confused with the older .NET Services name, thinking it was just the runtime and base class library running in the cloud, which makes no sense whatsoever.

17.1.1 The sibling of Windows Server AppFabric

Do not confuse the AppFabric we will be covering in this chapter with the new Windows Server AppFabric product. They are currently related by name alone. Over time they will merge to become the same product, but they aren't there quite yet.

Windows Server AppFabric is essentially an extension to WAS and IIS that makes it easier to host WCF and WF based services in your own datacenter. It supplies tooling and simple infrastructure to provide a base level messaging infrastructure. It does not supply a local instance of the ACS or Service Bus service at this time.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Likewise, Windows Azure Platform AppFabric doesn't provide any of the features that Windows Server AppFabric does, at least today. In early CTPs of Windows Azure Platform AppFabric there was the ability to host WF workflows in the cloud, but this was removed as they moved towards a production release.

The AppFabric we are going to cover in this chapter makes two services available to you, Access Control Service and the Service Bus.

17.1.2 Two key services

AppFabric is a library of services that focus on helping you run your services in the cloud and connect them to the rest of the world. Not everything can run in the cloud, as we have covered several times already in this book. You could have software running on devices out in the field, a client side rich application that runs on your customer's computers or software that works with credit card information and can't be stored off premises.

The two services in AppFabric are geared to help with these scenarios.

- Access Control Service—Also called ACS, this service provides a way to easily provide claims-based access control for REST services. This means that it abstracts away authentication and the role based minutia of building an authorization system. Several of the parts of Azure use ACS for their access control, including the Service Bus service in AppFabric.
- Service Bus—This service provides a bus in the cloud for you to connect your services and clients together so they can be loosely coupled. A bus is simply a way to connect services together and route messages around. An advantage to the Service Bus is that you connect can connect it to anything, anywhere, without having to figure out the technology and magic that goes into making that possible.

As we go through both of these services we will cover some basic examples. All of these examples rely on WCF. The samples will run as a normal local application, not as Azure applications. We did it this way to show you how these services can work out of the cloud, but also to make the samples easier to use. Each sample has two pieces that need to run, a client and a service. You can run both simultaneously when you hit f5 in Visual Studio by changing the startup projects in the solution configuration.

17.2 Controlling access with ACS

Managing identity, authenticating, and authorization are hard. It takes a lot of work by the developers to get right. One wrong step and you leave a gaping hole a bad guy can take advantage of and land your company on the front page of the newspaper. Security is always a high priority on any project. But security is always loaded with special terms and more complexity than we want to deal with. It can be just hard.

17.2.1 Identity in the cloud

Move all of these concerns out of your own network and into the cloud. These concerns become even bigger issues. The application is no longer sitting right next to the source of authentication; the identity boundaries have been broken.

The short answer is to just build yet another identity store (a place to store user names and passwords) and give every use yet another username and password to remember. They won't remember them, they will either write it down on a sticky note on their monitor, or they will call you once a month to remember what it is. This is a bad experience for the user, and it is dangerous for the owner of the application.

We call this the identity fishbowl. Your identity (who you are and what you can do) is more easily maintained on your network, but the second you go off premises you leave your fishbowl. The application in the cloud doesn't have any way to connect to your Active Directory to authenticate you. The way to bridge these worlds is with open standards and the concept of federated identity.

There are some other challenges as well. Go beyond the scenario above where we have an internal user trying to access our application in the cloud. What if the user doesn't work for us, but is a customer or vendor of some sort. What if I need to provision 100 user accounts for that new customer, or 10,000 accounts? This is a lot of work for you, it gives your end user yet another identity to remember, and exposes you to a risk of not shutting down an account when it needs to. You are on the hook for managing those accounts, not the customer or vendor.

What we want is an easy and secure way for your internal users and external users (customers, vendors, etc.) to be able to access your application using the identities they already have. This approach also has to have low

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

impact to my service code. I don't want to have to fix my code every time we enroll a new customer, or find a new protocol to support. Let's face it, we want to write applications, not become enterprise security ninjas. Well, most of us anyway.

ACS handles all of these concerns for us in a brilliantly elegant way. In order to really talk about ACS though we have to have a common understanding of some of the core concepts ACS is built on.

17.2.2 Working with actors

The security field is loaded with special vocabulary, terms and concepts that scare most developers. Even with the importance of security many developers just find some sample code and past it in. They either don't have the understanding needed to work with the code, or won't change it for fear of making a mistake.

There are several actors in our play that we need to define. The most important one is your service or application, the resource you are trying to protect. It is commonly called the 'protected resource' or the 'relying party' because it is relying on the security infrastructure.

The next actor is also easily defined, the client. This is the application that is trying to access the relying party. They are also called an 'issuer' at times. This side can get a little complicated when you have a second client that is being delegated through the first client to the protected resource.

And then finally we have the ACS service itself. In the security lingo ACS would be called the 'authorization server,' or 'trusted authority.' It provides the security infrastructure the client needs to use to authenticate and use the protected resource.

In our play, the director is the authorization server, the lead actor is the protected resources, and the client is the supporting actor.

17.2.3 Tokens communicate authorization

The three actors we discussed need some way to communicate, and they do this with passing tokens around. There are many formats for tokens, and there are many ways to pass them around. The messages they pass around usually include a set of claims, which we will get to in just a moment.

You likely use a token every day, your driver's license. You can use it to prove your identity or to prove a claim. A bar might demand you satisfy (aka prove) your claim that you are over 21. You never have to give them your birth certificate; you just give them your driver's license. You proved who you were, and when you were born, when you applied for your license. The department of motor vehicles validated your credentials and provided you with a token that proves that claim of your age on the license. The department is the trusted source, the bar with the beer is the protected resource, and your license is the token. You are the client. The following figure shows this relationship between you, the bar and the Department of Motor Vehicles. You first authenticate to the DMV. Once they are satisfied they give you a secure token that provides data for some claims. You can then use this token to get a frosty beverage at any bar, even if not everybody knows your name.

Figure 17.1 - A bar doesn't require you prove your birthdate. You provide them a valid and secure token from a trusted authority.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

You had to prove your birthdate to the authority (#1), and they gave you a token (#2). You then use that token to get into the bar (#3).

Some token formats are proprietary and some are an open standard. Microsoft has worked with Google and Yahoo! to develop a new simple open standard that can be used across the cloud and the web. This standard is called OAuth, short for open authorization. There are other open standards that are popular, such as SAML (Security Assertion Markup Language), but they tend to be more complex and geared for SOAP instead of REST. OAuth's goal is to provide a very simple token and protocol that makes it easy to secure REST based services.

You can read the actual OAuth specification by visiting http://groups.google.com/group/oauth-wrap-wg. ACS is based on OAuth, but knows how to read other types of tokens as well. There are two halves to OAuth we will look at. The first is WRAP, or Web Request Authorization Protocol. This is the protocol used to make authorization requests, and how those requests and tokens are moved around on the wire.

The other half is the definition of the token itself, called SWT. SWT means Simple Web Token, and it is exactly that. It is easy to read and understand. The goal was to build protocols that any platform could leverage. All tokens coming from ACS will be SWT tokens. This is a sample token. Notice that it is URL encoded. Later on we will look at code that will let us shred this token and understand the separate parts. If you have ever seen a SAML token, this is far simpler and easier to work with. There isn't even any XML!

```
CustomerId%3d31415%26Issuer%3dhttps%253a%252f%252fstringreversalinc.a
ccesscontrol.windows.net%252f%26Audience%3dhttp%253a%252f%252flocalhost%252fproc
essstring%26ExpiresOn%3d1266231958%26HMACSHA256%3dI5g66yaiECux9IQ8y7Ffm2S1p%252b
AXF73HWfzSNPyPLOE%253d
```

The working group's website includes the specification for these protocols and formats. They are surprisingly easy to read, but they do lack an interesting plot. They are much easier to read than the WS-* and SAML/STS specifications if that means anything. You can read them all in about 30 minutes.

17.2.4 Making claims about who you are

With all these tokens flying around we need something to put in them. While there are other pieces of data stored in tokens, the real reason they exist is to deliver what is called a set of claims.

A claim set is just a list of claims made about the client or user by the authorization server. What a claim represents is completely open and can be defined by the systems using them. The claim must serialize to plain text, there isn't any fancy XML in OAuth, just a name value collection of claims.

A claim set might include a user's name, their birthdate, their customer level, a list of roles or groups they belong to, or anything else your service (the protected resource) needs. Your service will use these claims to make security and behavior decisions. The first decision is whether the user is allowed to access your service. Once you have let them in you can use the claims to determine what they can do. Perhaps if their role claim includes 'manager' you will let them apply discounts to existing orders. If the role claim is 'staff' then maybe they can only create normal orders. What you ask for in claims and what you do with them once you get them is up to you.

When we started using claims we moved away from the traditional RBAC (role based access control) and towards CBAC (claims based access control). The concepts are the same; it is just how we get the data of the user's identity and how we make decisions on how to use that data.

As you implement ACS it is possible to add the use of SWT tokens to your system without ripping out the old way of managing identity. This is useful if you are trying to transition to the new platform without breaking what already exists.

17.3 Example: a return to our string reversing service

Let's think back, way back to chapter 15 when we talked about the amazing string reversal company, String Reversal Inc., and our service that used a new and innovative way to reverse strings. In the time it has taken you to read the intervening chapters the company has grown and prospered. String reversal user groups and industry conferences are springing up all over the world.

But our emerging company is running into some trouble. Every time they add a customer they have to do a lot of work just to provision that customer in their system. This overhead is getting in the way of them and their rapid

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

expansion. It has also led to a few problems. While you were in the middle of reading the chapter on queues one of the company's customers, Maine Reversal, was forced to fire its employee Newton Fernbottom for insider string reversing, a horrible, horrible crime. Because that customer didn't notify us that Newton was let go we never disabled his account. Newton ran home and starting a competing firm, Downeast Reversing, using Maine Reversal's account with our company.

What we need to do now is provide a better way to authorize customers to use our service, and we want to minimize the amount of code we need to change. Because anything can be a client, and anything can be a protected resource with ACS we are going to move our sample service to a normal local WCF service, just to make it easier to focus on how ACS works. Everything ACS does can easily be applied to both services and clients running in Azure.

17.3.1 Putting ACS in place

Our first step in upgrading our service is to support a simple scenario where our customers will have a shared secret (similar to a username and password) to access our service. Whoever they give that secret to will be able to use the service. They will be able to change the secret when they need to, just like changing your password every 30 days.

Our first step is to login to the Azure portal and create an AppFabric namespace. This namespace is a lot like a container in blob storage. It holds our settings for how we are using the ACS service for our needs. We could have several namespaces if we wanted to, perhaps to isolate different services and how they are configured. To create the namespace we will use the portal. Besides creating the namespace the portal doesn't do much for us with regards to ACS. There are other tools for that.

Figure 17.2 - To start using AppFabric you first have to create a namespace. This acts like a container for the entire configuration of ACS and the Service Bus. This namespace has to be globally unique.

To create a namespace, login to your Azure portal and choose 'AppFabric' on the left hand side. You will then see a list of your existing namespaces and a button to create a new one. These namespaces are also used by the Service Bus later on. You simply need to provide a globally unique name for your namespace. In figure 17.*** you can see we have selected 'StringReversalInc' for our namespace. Once you click create AppFabric will provision its systems with your namespace.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 17.3 - Once you create a namespace AppFabric will provision that namespace with ACS, Service Bus and management endpoints. Security keys will be generated for you as well.

ACS has configured both a Service Bus and an ACS service for you with your namespace. Their service endpoints for both services will be displayed to you. Notice your namespace is the hostname of the service endpoints.

A management key will be created for you as well. This 32 byte symmetric key is what you will use when accessing the AppFabric management service to perform operations on your namespace. We won't explore the management service in this chapter but you should check it out. The keys listed here should not be shared outside your organization, or published in a book where anyone can get a hold of them.

17.3.2 Reviewing our String Reversal Service

We have adjusted our string reversal service from chapter 15 to be a simple REST service that runs locally. We have removed the entire worker role and Azure related code to do this. ACS is about securing REST based services, and our old service was using a TCP based binding. We have changed it to use REST by using the `WebServiceHost` and the `WebHttpBinding`.

Our first code listing shows how we are building our simple little service. This code will start up the service, and wait for calls to the service.

Listing 17.1 - Our simple REST service

```
using System.ServiceModel;
using System.ServiceModel.Web;

public class svcProcessString
{
  public static void Main(string[] args)
  {
    Console.WriteLine("Starting string reversal servicehost...");
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
      WebServiceHost serviceHost = new
[CA]WebServiceHost(typeof(ReverseStringTools));                          #A

      WebHttpBinding binding = new WebHttpBinding(WebHttpSecurityMode.None);

      serviceHost.AddServiceEndpoint(typeof(IReverseString), binding, new
[CA]Uri("http://localhost/processstring"));

      try
      {
        serviceHost.Open();                                              #B
        Console.WriteLine("String reversal servicehost started.");
      }
      catch (Exception ex)
      {
        Console.WriteLine("Could not start string reverser servicehost. {0}",   [CA]ex.Message);
      }

      Console.ReadLine();
    }
}
```

#A Create a service host to run the service code.
#B Start the service host with the configuration we setup.

If you run this sample string reversal service you will be able to make all of the requests to the service you want. The sample code incudes a simple client to call the service. Our next few steps are going to center around adding code to the service so that it can read and use SWT tokens. Once that is done we will upgrade our client to be able to fetch a token from ACS and use it during a request to our service.

17.3.3 Accepting Tokens from ACS

We need to upgrade our service to know how to receive and work with ACS tokens. This code is fairly trivial, and much of it is supplied in the ACS SDK. In order to take these next steps you will have to install the ACS SDK. You can find the SDK on the Azure portal. It also includes several tools that we will look at in the next section.

Exactly how we get the token might change depending on your situation, and exactly where we process the token might change, but in general the steps are the same. The first step is to grab the token from the incoming message. This will usually be in the header as an authorization entry. In some situations it can also be in the URL or in the body of the message, depending on the capabilities of the client. Exactly how you grab this header will differ based on how you are receiving the message. In WCF it is best to do this in a custom `ServiceAuthorization` class. This class is added to our WCF pipeline when we set up the channel. Every message will flow through this class, and we can make a decision if we let it through or deny it access. In our normal WCF service we need to use the `WebOperationContext`.

```
string authorizationHeader =
[CA]WebOperationContext.Current.IncomingRequest.Headers[HttpRequestHeader.Authorization];
```

This code will just get us the raw header. We now need to do a few things to make sure this token is valid, and then we can use it to make decisions. The SDK has all the sample code you need to build a class called `TokenValidator`. I have taken that code and refactored it a little to make it easier to use. The validator will do a series of checks for us, and if they all pass it will return true. If the validation fails we will deny the access.

```
validator = new [CA]ACSTokenValidator("dqSsz5enDOFjUvUnrUe5p1ozEkp1ccAfUFyuYpawGW0=",[CA]
"StringReversalInc", "http://localhost/stringservice");

if (!validator.ValidateAuthorizationHeader(authorizationHeader))
  DenyAccess();
```

To initialize our validator we need to pass in three pieces of information. We need to pass in our signing key, the ACS namespace to check against, and the URL of the service the message was sent to. We are passing in our key, the namespace we setup called "StringReversalInc", and the URL of the service we are protecting, http://localhost/stringservice.

We then call the ValidateAuthorizationHeader on the header we pulled off of the message. If this returns false we will deny access by calling a simple little method, `DenyAccess`, that setups the deny message.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
private static void DenyAccess()
{
  WebOperationContext.Current.OutgoingResponse.StatusCode =
[CA]HttpStatusCode.Unauthorized;
                WebOperationContext.Current.OutgoingRequest.Headers.Add("WWW-
[CA]Authenticate", "WRAP");
  }
```

That's all we need to receive the header. Of course most of the work is making sure it is a valid header, and something we can trust. This is the same as the bouncer at the bar looking at your driver's license to make sure it is a real license and has not been tampered with.

17.3.4 Checking the token

We have put all of our token checking logic into the class ACSTokenValidator. We discussed just above how to new up a validator. When we pass in the header will verify several aspects of it to make sure it is valid. All of these tests test for the negative. If the test passes we have a bad token and we return false. Table 17.1 summarizes the checks that we do in the code.

Table 17.1 Validation checks on a token.

Check to make	Why
`string.IsNullOrEmpty(authHeader)`	Make sure we received a header.
`!authHeader.StartsWith("WRAP ")`	By defection the header should always start with WRAP.
`nameValuePair[0] != "access_token"`	There should be two pieces to the header. The first should be equal to "access_token".
`!nameValuePair[1].StartsWith("\"") || !nameValuePair[1].EndsWith("\"")`	The second piece must start and end with a slash.
`!Validate(GetTokenFromHeader(authHeader))`	Grab the actual token part of the header and make sure it is valid.
`IsHMACValid(token, signingKey)`	Make sure the token has been signed properly. It this is correct we know who sent it.
`this.IsExpired(token)`	We want to make sure the token hasn't expired. Tokens are vulnerable to replay attacks, so this is important.
`this.IsIssuerTrusted(token)`	Do we recognize the sender as a trusted source? We will cover this shortly.
`this.IsAudienceTrusted(token)`	The audience is the intended destination. Is this a destination we are protecting?

If we pass all of these checks then we know we have a secure token from a trusted source, and that it is meant for us. This is the minimum you will want to do to allow the message through to the service. You may also want to crack open the claim set in the token to look at what claims have been sent, and make decisions on those claims. In our sample we have mapped in some claims. One is the customer id number, and the other is their service level. This might be used to determine how long the strings they submit can be to use our service. They would have to pay more to reverse longer strings.

That is all we have to do to enable our service to consumer and use ACS tokens for authorization. The next step is to look at how we configure a client to add the authorization header to their request.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

17.3.5 Sending a token as a client

We are going to build a simple command line client. This will make it easier to focus on the service aspects of the code. Feel free to make it sexy by using Silverlight or WPF. For our client we are going to share our contract through sharing a project reference. Many projects do this, especially when you own both ends of the conversation. You should either share the contract through a project reference, or through a shared assembly.

In this case our biggest customer, Maine Reversal, will be building this client. We have setup a trusted relationship with them by swapping keys, and configuring them in ACS. We will look at how to do this in the next section. In our example MainReversal will not be sending in any custom claims of their own, just their issuer identity. This process is essentially giving them a secure username and password.

We have created a helpful utility class that encapsulates how to fetch an ACS header from the AppFabric service. Again, this is mostly from the SDK with some tweaks we wanted to make. Hey, why write new code when they give us code that works? To call this method we want to pass in the service namespace ("StringReversalInc"), the issuer name (that is the client's name, "MaineReversal" in this case), the signing key that MaineReversal uses, and the URL of the intended destination, the 'audience' in security parlance.

```
string Token = GetTokenFromACS("StringReversalInc", "MaineReversal",
"ltSsoI5l+8DzLSmvsVOhOmflAsKHBYrGeCR8KtCI1eE=", "http://localhost/processstring");
```

The `GetTokenFromACS` performs all the work for us. It uses the `WebClient` class to create the request to ACS. If everything goes well the ACS service will respond with a token we can put in our authorization header on our request to the string reversal service. Our sample continues with showing how we can request a token from the ACS service.

Listing 17.2 - How a client gets a token from ACS

```
private static string GetTokenFromACS(string serviceNamespace, string [CA]issuerName, string
issuerKey, string scope)
{
  WebClient client = new WebClient();
  client.BaseAddress = [CA]string.Format("https://{0}.accesscontrol.windows.net",
serviceNamespace);                                          #1

  NameValueCollection values = new NameValueCollection();    #2
  values.Add("wrap_name", issuerName);
  values.Add("wrap_password", issuerKey);
  values.Add("wrap_scope", scope);

  byte[] responseBytes = client.UploadValues("WRAPv0.9", "POST", values);

  string response = Encoding.UTF8.GetString(responseBytes);

  return response                                           #3
         .Split('&')
         .Single(value => value.StartsWith("wrap_access_token=",
[CA]StringComparison.OrdinalIgnoreCase))
         .Split('=')[1];
}
```

#1 The ACS service address
#2 Sends our authorization data
#3 Gives the token to the caller

Replace #1-3 with cueballs in code and text

We have to provide the `GetTokenFromACS` method with the based address for our request. #1 This is a combination of the ACS service address, `accesscontrol.windows.net`, and the namespace for our ACS account, StringReversalInc. To make the call we need to provide three pieces of data. The client has to provide the issuer name, which is their 'name' in the ACS configuration, their signing key, and the namespace of the service they are trying to reach.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

At this point ACS will check their credentials. The issuer name is basically their username, and the signing key is their password. If everything checks out ACS will respond with a valid token for the client to attach to their request to the service.

17.3.6 Attaching the token

Attaching the token to the header of the request is fairly simple on most platforms. You can put the token information in the URL or the message body. Doing either isn't as good as using an authorization header, so only do this if your system doesn't support an authorization header.

```
string authorizationHeader = [CA]
string.Format("WRAP access_token=\"{0}\"",[CA] httpUtility.UrlDecode(Token));
                WebOperationContext.Current.OutgoingRequest.Headers[CA]
.Add("authorization", authorizationHeader);
```

To attach the token to the header we need UrlDecode it, and then wrap it with the WRAP leading text. This tells the destination service the token is a WRAP token. It will be stripped off by the server once validated. Then we simply add the header to the outgoing request using the `WebOperationContext` class.

Once this is done nothing else needs to be done by the client. Your client should be robust enough to handle any errors in the ACS service call, or the ACS token request is denied.

In order for this to work we had to setup some configuration in the ACS service. We had to setup a trusted relationship with the issuer, and provide some rules.

17.3.7 Configuring our ACS namespace

The ACS needs to be configured for your service. You have already learned how to define a namespace for you service. The namespace is a container for the rest of the ACS configuration. You can chain namespaces together, which is the key mechanism for providing simple delegation.

Each namespace has four components. The components are issuers, scopes, rules, and token policies. These elements work together to help secure your REST service. You have two options in how you configure your service. Both options are provided in the ACS SDK, and run locally. Both of these tools call into the management service, which you can as a third option, but will required more work on your part. Beyond setting up the namespace there aren't any management tools on the ACS portal.

The first option is to use ACM.exe at the command line. ACM can be found in the tools folder where you installed the ACS SDK. ACM is most useful for when you are automating a process or want to script the configuration. The second tool is the Azure Configuration Browser. This is shipped in the SDK as well, but in a source code form. You need to load the solution and compile it to use it. This is really useful because you can then extend the tool to meet your needs, and it is a lot easier to use than the command line tool.

The Configuration Browser does have a few limitations. First, it is really ugly, but that is ok. The second is that, at this time, it cannot update your cloud configuration; it can only deploy a complete configuration. This means that anytime you make a change you have to delete the configuration in the cloud and then completely redeploy the new configuration. An advantage is that you can store you configuration locally in an xml file, which aids in backup and configuration management.

You will need to provide your service name and your management key with either tool. You can put your settings in the app.config file for the ACM.exe application, which saves you from having to type them in as part of your commands every single time.

ISSUERS

Issuers are an actor in the ACS system and represent a consumer of your service. When you create an issuer you need to provide both a display name and an official issuer name. Once the issuer is created a signing key will be generated for you. This is the key the issuer must sign their requests with when they ask the ACS service for a token.

To do this from a command line you would use the following.

```
acm create issuer -name:MaineReversal -issuername:MaineReversal -autogeneratekey
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In the management browser you will need to right click on the issuers node and choose create. The next screen shot shows how we setup our first client, Maine Reversal.

Figure 17.4 - Creating an Issuer is easy with the ACS Configuration Management Browser. You will need to provide both a display name and an official name. You can use the tool to automatically create the signing keys.

Setting up an issuer in the system is akin to creating a user with a username (the issuer name) and a password (the signing key.) These will be the credentials they will use to request an ACS token.

TOKEN POLICY

A token policy is defines how you want your tokens to be created. Because token based systems can be vulnerable to token replay attacks you will first want to set a lifetime timeout for the token. This is expressed in seconds. When the token is created it will be stamped with the time when the token will expire. Once it is expired the token will be considered invalid and will not be accepted by a service.

The command at the command line is

```
acm create tokenpolicy -name:StringReversalInc -autogeneratekey
```

To create a token policy in the management browser right click on the token policy node, and choose create. Here we are creating a policy for our string service.

Figure 17.5 - You will need to create a token policy. This will determine the lifetime of your tokens, and the key that will be used to sign your ACS tokens.

The second piece of data you will need for your token policy is the signing key, which can be generated for you. This is the key that will be used to sign the tokens generated for you by the ACS.

SCOPE

Scope is a container that is tied to a service URI. It will bring together the rules that you want applied to your service, as well as the token policy you want used for that service.

To create one at the command line level you will need the id of the token policy you want to assign to the scope. You can get the tokenpolicyid from the output of the `create tokenpolicy` command above.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
acm create scope -name:myservicescope -appliesto:http://localhost/myservice -
tokenpolicyid:tp_4cb597317c2f42cba0407a91c2553324
```

When you are using the management browser you will not need to provide the token policy id, you will simply be able to choose it from the drop down. We can now associate our policy to our namespace by creating a scope.

Figure 17.6 - It is easy to create a scope. A scope acts as a container for a set of rules for your service. It also associates a token policy with the service. You will need to define the URI for the service the scope applies to.

There are several advanced uses for scope that we don't time to go into in this chapter. These include managing a large number of service endpoints with a few scopes, and changing namespaces together for delegation.

RULES

Rules are the heart of the ACS system. When a token is created by ACS the rules for the related scope are executed. This is the process that allows you to change the request from the consumer to something your application expects. They can also be used to create claims out of thin air, and put them in the resulting token.

One example would be if you want to place a claim in the token that represents the consumer's customerid to make it easier for your service to identify the account the request is related to. You could create a rule that says 'If this is for issuer MaineReversal, then add a claim called customerid with a value of 31415.' The next screen shot shows how we create this rule.

Figure 17.7 - Creating a rule to insert a claim that includes the customer's customerid. In this case we are relying on the 'issuer' of the inbound request to know which customer it is.

Another rule we could use would assign a new role value based on mappings we have worked out with the customer. Perhaps their system tracks their users with a role of ServiceManager. Perhaps this is a group the user belongs to at Maine Reversal. Our system doesn't know that role name, and we don't want to add all of our customer's role types to our system, it would get very complex very quickly. In this next rule we create the roles claim with the manager value.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 17.8 - In this figure we are creating a rule that substitutes the inbound roles claim for a new one. This is because our system doesn't have a role called ServiceManager. Using this rule we can map that role value to one our system does know, manager.

What we can do is create a rule that finds a claim called role with a value of SalesManager, and replace it with a claim called roles that has a value of manager. In this way we have moved the customer configuration and mapping out of our service and into the authorization service where it belongs.

Creating a rule at the command line is a little more complex than using the management tool.

```
acm create rule -name:MaineReversalMap[CA]
     -scopeid:scp_e7875331c2b880607d5709493eb2751bb7e47044 [CA]
     -inclaimissuerid:iss_6337bf129f06f607dac6a0f6be75a3c287b7c7fa [CA]
       -inclaimtype:roles -inclaimvalue:ServiceManager [CA]
       -outclaimtype:roles -outclaimvalue:manager
```

To find the id's of the scope and issuer you can use the command `acm getall scope` and `acm getall issuer`.

17.3.8 Putting it all together

We have come a long way in stopping illicit use of our service. Now we can control who uses it, and how they use it. We updated our service to consume tokens. We updated the client to submit them with service requests, and we prepared the ACS service with our configuration.

How does this all work? In the simple scenario we have in front of us, the client requests an access token from ACS. It provides its secret key and issuer name. ACS validates this and creates a token, using the scope rules we setup to create claims in the new token. The client then attaches the token to the message in the authorization header. Figure 17.9 should look familiar; it is basically the same as the DMV example (see figure 17.1), but shows the technical actors and how they interact together.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 17.9 - How the three actors work together to access a REST service in a secure way. The service configures ACS. The Client asks for a token. ACS creates a token, based on rules. Then the Client submits this token with their service request.

When our service finally receives the message we will grab the token from the header and verify it. We want to make sure that it is valid, and hasn't been tampered with. Once we can trust it we can make decisions on what to do. In our case we can take the customerid value and verify they are still a paying customer, and if so, respond to their request. We can stop using the token at this time and just respond like normal, or we can shred the token and use the claims `throughout` our application.

If we were protecting an ASP.NET website instead of a REST based WCF service we could `take` those claims and put them in a class the implements `IPrincipal`, and use it to make role decisions throughout our code.

We just finished a quick lap around ACS. ACS' sibling is the Service Bus, which will let us connect anything to anywhere, with just a little bit of WCF and cloud magic.

17.4 Connecting with the Service Bus

The second major piece of Windows Azure Platform AppFabric is the Service Bus. As adoption for a service oriented architecture increases, developers are seeking better ways of connecting their services together. At the simplest level, the Service Bus does this for any service out there. It makes it easy for services to connect to each other and for consumers to connect to services.

In this section we are going to look into what the Service Bus is, why we would use a bus, and most importantly how can connect our services to it. You will see how easy it is to use the Service Bus.

17.4.1 What is a Service Bus

Enterprise Service Buses have been around for years, and they have grown out of the SOA movement. As services became popular, and as the population of services at companies increased companies found it harder and harder to maintain the infrastructure. The services and clients became so tightly coupled that the infrastructure became very brittle. This was the exact thing services wanted to avoid. ESB's evolved to help fix these problems.

ESB's have several common characteristics, all geared for building a more dynamic and flexible service environment. The first service they provide is a service registry. Developers and dynamic clients needed ways to find services available to them, and find out the contract and usage information they needed to consume them. With this registry they needed to provide a way to name the services. This was an effort to provide a namespace

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

around services so there wasn't conflict in both the service names, but the message types defined. ESB's tend to provide some infrastructure for security, a way to allow or deny people access to a service, and what they are allowed to do on that service. Most importantly there was the 'bus' part of ESB. The bus provides a way for the messages to move around from client to service, and back. The important part of the bus was the instrumentation in the end points that allowed IT to manage the endpoint. They could track the SLA of the endpoint, performance, and track faults on the service. Another common feature of an ESB is service orchestration. Orchestration is the concept of composing several services together into a bigger service that achieved some business process. A common model for ESBs is below. Figure 17.10 is very similar to the typical n-tier architecture model. Each tier relies on the abstractions provided by the layer below it.

Figure 17.10 - Many ESBs support a three tier concept. The lowest tier is comprised of funcational services, very discrete services. These services are composed together in the service orchestration layer, and are exposed to provide a comprehensive business service.

The orchestration became not only a way to have lower level services work together, but it also provided a layer of indirection on top of those services. In the orchestration layer you can route messages based on content, policy, or even service version. This is important as you connect services together, and as they mature.

17.4.2 Why an ESB is a good idea in the cloud

The problem for ESB's is that they usually only connect internal services and internal clients together. It is hard to publish a service you don't control to your own bus. External dependencies end up getting wrapped in your own service. While this avoids the first problem introduces a new problem, which is yet more code.

If I wanted to expose a service to the several vendors or if I wanted a field application to connect to an internal service I have to play all sorts of firewall tricks. I have to open ports, provision DNS, and many other things that make IT managers have nightmares.

Another challenge is the effort it takes to make sure that an outside application can always connect and use your service. To go one step farther, it is an even bigger challenge to connect two outside clients together. The problem comes down to the variety of firewalls, NATs, proxies, and other network shenanigans that make point to

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

point communication difficult. Figure 17.11 might represent how many layers are between your local software and the services it is calling across the Internet.

Figure 17.11 - Modern networks provide a great number of barriers to easy point to point communication. Many computers these days have a local firewall, with one or more firewalls on their network. You also have proxies, NATs and other devices in the way making it hard to connect in an old fashioned direct way.

Take your instance messaging client for example; almost all of them operate on the same principal. When the client starts up, and the user logs in, the client creates an outbound, bidirectional connection to the chat service somewhere. This is always allowed (unless the firewall is configured to explicitly block that type of client) across the network, no matter where you are. An outbound connection, especially over port 80 (where http lives) is rarely a problem. Inbound connections on the other hand almost always are a problem.

Both clients have these outbound connections, and they are used for signaling and commanding. If client A wants to chat with client B a message is sent up to the service. The service uses the registry to figure out where client B's inbound connection is in the server farm, and sends the request to chat down client B's link. If client B accepts the invitation to chat then a new connection is setup between the two clients with a predetermined rendezvous port. In this sense the two clients are bouncing messages off a satellite in order to always connect, because a direct connection, especially inbound wouldn't be possible.

This strategy gets the traffic through a multitude of firewalls, on the PC, on the servers, on the network, on both sides of the conversation. There is also NATting (network address translation) going on. This is very common. A network will use private IP addresses internally (usually in the 10.*.*.* range), and only translate that to an IP address that works on the Internet if the traffic needs to go outside the network. It is quite common for all traffic coming from one company or office to have the same source IP address, even if it is hundreds of actual computers. The NAT device keeps a list of which internal addresses are communicating with the outside world. This list uses the TCP session id (which is buried in each network message) to route inbound traffic back to the actual computer that asked for it.

The 'bounce it off a satellite' approach bypasses this problem by having both clients dialing out to the service. You can visualize how all of this works with the following figure.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 17.12 - Any client can communicate with any other client (perhaps a service) from anywhere, on any network by using the relay bindings with the Service Bus. Each client first registers with the bus so it knows where they are connected. Then when A wants to connect with B they can use the registry to find each other.

The Service Bus is here to give you all of that easy messaging goodness without all of the work. Imagine if Skype or Yahoo Messenger could just write a cool application that helped people communicate, instead of spending all of that hard work and time figuring out how to always connect with someone, no matter where they are.

The first step in connecting though is knowing who you can connect with, and where they are. To do this we need to register our service on the Service Bus.

17.5 Example: listening for messages on the bus

Previously in this chapter, as we were configuring ACS we had to create a namespace. That namespace held configuration for the services we wanted to protect with ACS. That same namespace will be used to register and manage our service on the bus.

Almost all of the magic of the Service Bus is embedded in the new WCF service bindings that they ship. All of these bindings start with a protocol identifier of 'sb'. You are probably familiar with 'http' and others. The binding names almost always have the word `relay` in them, to indicate you will be relaying messages through the Service Bus to their destination.

17.5.1 Connecting the service to the bus

Changing our application to listen for messages from the bus instead of the http endpoint we had is very easy. We need to change the binding and address information to point to the bus.

We have moved the configuration of our service from code to the app.config file to make these changes easier. There are two pieces we need to setup. We need to configure the address and binding information, and we need to configure the service for authentication to the bus.

In our configuration we simply change the address to our namespace on the bus. We are going to change it from `http://localhost/processstring` to `sb://stringreversalinc.servicebus.windows.net/processstring`. This change tells WCF to use the

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

service bus relay bindings, and that the service we are publishing should be registered in the `stringreversalinc` namespace.

```
<endpoint address=
"sb://stringreversalinc.servicebus.windows.net/processstring"
behaviorConfiguration="sharedSecretClientCredentials"
binding="netTcpRelayBinding"
contract="StringReversalLibrary.Contract.IReverseString" />
```

Our service also has to authenticate to the bus when it starts up and registers with the bus. We use ACS to do this. In our example we are using the simple shared secret we did earlier in this chapter.

```
<behavior name="sharedSecretClientCredentials">
   <transportClientEndpointBehavior credentialType="SharedSecret">
      <clientCredentials>
         <sharedSecret issuerName="owner"[CA]
issuerSecret="MZuYNde3ZOzUJxVKo62kmWoFSlzEZEaKai5Fktlt3pQ=" />
      </clientCredentials>
   </transportClientEndpointBehavior>
</behavior>
```

This shared secret behavior we attach to our service will authenticate to the bus with our issuer name and signing key automatically.

Once we make these changes, when you run the service it will start u, authenticate, and start listening for messages coming from the bus. We will need to perform very similar actions on the client.

17.5.2 Connecting to the service

In the previous section, we looked at what we had to do to connect a service to the bus. We essentially just had to change the bindings to point to the bus, and update the address. We also had to add some authentication so that the bus knew we were allowed to use our namespace.

We are going to follow the same steps for the client we have. We need to change the client binding so it is sending messages to the bus. We will name our endpoint SBRelayEndpoint, with the same address our service used.

```
<client>
   <endpoint
      name="SBRelayEndpoint"
      address=
"sb://stringreversalinc.servicebus.windows.net/processstring"
      binding="netTcpRelayBinding"
      contract="StringReversalLibrary.Contract.IReverseString"
      behaviorConfiguration="sharedSecretClientCredentials"
   />
</client>
```

Our client is going to have to authenticate to the bus as well. We are going to configure it to use a shared secret. We will use the same Maine Reversal issuer we created in the ACS section of the chapter.

```
<behavior name="sharedSecretClientCredentials">
   <transportClientEndpointBehavior credentialType="SharedSecret">
      <clientCredentials>
         <sharedSecret issuerName=" MaineReversal"[CA]
issuerSecret=" ltSsoI5l+8DzLSmvsVOhOmflAsKHBYrGeCR8KtCI1eE=" />
      </clientCredentials>
   </transportClientEndpointBehavior>
</behavior>
```

In this case we have configured it with the issuer name and signing key we created for Maine Reversal. Now they can call our service from anywhere and always be able to connect to it. This makes it really easy to provision new customers. In the old direct method we had to reconfigure the firewalls to let the new customer through. Now we can just point them to the Service Bus address and give them the credentials they will need.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The binding we used in this example is based on TCP, which is one of the fastest bindings you can use in WCF. It adds the 'relay' capabilities to allow us to message through the bus instead of directly. There are other bindings available that support using the relay.

Now that we have covered what AppFabric can do today we will consider what its future might hold. The future of software is often tricky to track, and companies are reluctant to set expectations that might not hold when the new version ships. Everything we are going to discuss is based on what Microsoft has discussed publicly. While it still might change, it is informative to understand where they expect the product to go over the next few years.

17.6 The future of AppFabric

We normally don't want to talk about futures in a book because it's entirely possible that priorities shift and the features never ship. The team has been open on what they consider their next steps. Keep in mind that any of this could change so do not base important strategy on unofficial announcements.

- Extending OAuth for Web Identity

  The ACS team wants to extend its use of OAuth and SWT to include common web identity platforms. These are identities on the web that people commonly have, instead of traditional enterprise identities from Active Directory at work. It will be possible for a user to choose to log into your site not just with an ACS provisioned issuer id, or a SAML token, but also with their Google, Yahoo! and Live ID accounts. This is accomplished by ACS federating with those directories with the updated OAuth protocols. This will also include FaceBook and any OpenID provider. This is very exciting for people who are building consumer centric applications.

- Support for WS-*

  The other big feature that have publicly discussed is the support for all WS-* protocols. Right now you can only protect REST services with ACS and they want to extend support for SOAP based services as well. Once they have support for WS-* they will have support for WS-Federation and WS-Trust which makes it easier to federate enterprise identities. This change will also give them support for CardSpace which is Windows identity selector for claims based authentication systems.

17.7 Summary

This chapter gave you a basic understanding of what Windows Azure Platform AppFabric is. We looked at how it is a cousin of Windows Server AppFabric, and that maybe over time they will merge.

The first component of AppFabric we looked at was ACS. ACS's primary concern is securing REST based services, and it does this with the open standards OAuth and SWT. ACS makes it easy to federate in other token protocols like SAML, and makes it easy to transform tokens from other parties into a shape that your application wants to consume. You can configure ACS with either the command line application ACM, or the ACS Configuration Management Browser. Both applications use the underlying management REST service, which you can also use directly if you want.

We looked at how a client first authenticated with ACS to prove who they are, and then ACS gives them a token to use with the service to gain access. This removes the concern of authentication from the application, which in turn simplifies the codebase, and makes it easier to adjust to new rules as they evolve.

The second component of AppFabric we looked at was the Service Bus. The Service Bus is a migration of a common Enterprise Service Bus into the cloud. Their goal is to make it easy for consumer and services to communicate with each other no matter where they are or how they are connected.

We adjusted out sample to use a net TCP binding, but to also relay the messages through the cloud to bypass any crazy firewalls or proxies that might be in the way. We just had to make a few adjustments on the service and the client to make this possible. Both sides do have to authenticate to ACS before they connect to the bus however, and this is the mechanism on how you would secure your service when it is connected to the bus.

While this was a simple chapter to help you get the feel for AppFabric, we hope that you are comfortable enough with the basics to know when you might be in a situation when it might make sense to leverage. You should now be prepared enough to explore on your own and maybe dive deep with a dedicated book.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Our next chapter is going to zoom out from all of this detail and help us understand what our applications are doing by using the diagnostics manager. The diagnostics manager will help us know what is happening, and the service management API will help us do something about it.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

18

Running a healthy service in the cloud

This chapter covers:

- Getting to know the Windows Azure Diagnostics platform
- Using logging to know what is happening with your service
- Using the service management APIs
- Making your service self-aware by combining logging and service management

Building and deploying and application to Azure is just the first step in a hopefully long application lifecycle. For the rest of its life the application will be in an operation mode, being cared for by loving IT professionals and support developers. The focus shifts from writing quality code to running the application, and keeping it healthy. There are a lot of tools and techniques out there to manage your infrastructure.

What 'healthy' means can actually be different for every application. It might be a measure of how many simultaneous users there are, or how many transactions per second are processed, or how fast a response can be returned to the service caller. In many cases it won't be just one metric, but a series of metrics. You have to decide what you are going to measure to determine a healthy state, and what those measurements must be to be considered acceptable. You must make sure these metrics are reasonable, and actionable. A measure that demands the site be 'as fast as possible' is not really measurable, and it is nearly impossible to test for and fix an issue phrased like that. Better to define the metric as an average response time for a standard request.

Keeping your application healthy means instrumenting and gathering diagnostics data to make that possible. In this chapter we are going to discuss how you perform diagnostics in the cloud, and then what tools Azure gives us to remediate any issues or under-supply conditions in our system.

18.1 Diagnostics in the cloud

At some point you might need to debug your code, or you will want to judge how healthy your application is while it is running in the cloud. I don't know about you, but the more experienced I get with writing code, the more I know my code sucks. I have drastically reduced the amount of debugging I need to do with the use of TDD, but I still need to fire up the debugger once in a while.

Debugging locally with the SDK is easy, but once we move to the cloud we can't debug at all, so we need to look at how we can log the behavior of the system. When logging, we can either use the infrastructure that Azure provides for us, or we can us our own logging framework. Logging, like in traditional environments is going to be our primary mechanism for collecting information on what is happening with our application.

18.1.1 Using Azure Diagnostics to find what is wrong

Logs are very handy. They help you find where the problem is, and can act as the flight data recorder for you system. They come in handy when your system has completely burned down, fallen over, and sank into the swamp. They also come in handy when the worst hasn't happened, and you just want to know a little bit more about the behavior of the system as it is running. Logs can be used to analyze how your system is performing, and

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

to understand better how it is behaving. This information can be critical when you are trying to determine when to scale the system, or how to improve the efficiency of your code.

The drawback with logging is that hindsight is 20/20. It is obvious, after the crash, that you should have enabled logging or that you should have logged a particular segment of code. As you write your application, it is important to consider instrumentation as an aspect of your design.

Logging is much more than just remote debugging 1980s style. It's about gathering a broad set of data at runtime that you can use for a variety of purposes, debugging them is one of those purposes.

18.1.2 Challenges with troubleshooting in the cloud

When you are trying to diagnose a traditional on-premises system you have easy access to the machine and the log sources on it. You can usually connect to the machine with remote desktop and get hands on. You can parse through log files, both those created by Windows, and those created by your application. You can monitor the health of the system by using performance monitor, and tap into any source of information on the server. During troubleshooting it is common to leverage several tools on the server itself to slice and dice the mountain of data to figure out what has gone wrong.

You simply can't do this in the cloud. You can't login to the server directly, and you have no way of running remote analysis tools. But the bigger challenge in the cloud is the dynamic nature of your infrastructure. On-premises you have access to a static pool of servers. You know which server was doing what at all times. In the cloud you don't have this nature. Workloads can be moved around; servers can be created and destroyed at will. And you aren't just trying to diagnose the application on one server, but across a multitude of servers, collating and connecting information from all of the different sources. The number of servers used in cloud applications can swamp most diagnostics analysis tools. The shear amount of data available can clog up bottlenecks in your system.

For example, a typical web user, as they browse your web site and decide to check out, can be bounced from web role to web role because of the load balancer. How do you truly find out the load on your system or the cause for the slow response while they were checking out? You have to have access to all of the data we are used to on terrestrial servers, and we need the data collated for us.

We also need close control over the diagnostic data producers. We need an easy way to dial the level of information from 'debug' to 'critical.' While we are testing our systems we want all the data possible, and the additional load it places on the system is acceptable. When we are in production we only want to know about the most critical of issues, and we want to minimize the impact on system performance.

These are the reasons why the Windows Azure Diagnostics platform sits on top of what is already present in Windows. They have extended and plugged into the existing platform, making it easy for you to learn, and easy to find the information you need.

18.2 Diagnostics in the cloud is just like normal, almost

With these challenges of diagnostics at cloud scale, it is amazing that the solution is so simple and elegant. Microsoft chose to keep everything that is xxx in place. Every API, tool, log, and data source stays the way it is. This keeps the data sources known and well documented. The diagnostics team provides a small process that is started on your instances called MonAgentHost.exe.

The MonAgentHost.exe process is started automatically, and it acts as your agent on the box. It knows how to tap into all of the sources, and it knows how to merge the data and move it to the proper locations for you to analyze. It can be configured on the fly without having to restart the host it is running on. This is critical. You don’t want to have to take down a web role instance just to dial up the amount of diagnostic information you are collecting. This allows you to control data collection across all of your instances with a simple API. You can see all of the moving parts in figure, 18.1.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 18.1 The MonAgentHost.exe process gathers, buffers, and transfer many different sources of diagnostics data on your behalf. It is the agent we will be focusing on in this section.

As the developer, you are always in control of what is being collected, and when it is actually collected. You can communicate with MonAgentHost by submitting a configuration change to the process. When you do this, the process will reload, and start executing your new commands.

18.2.1 Managing event sources

The local diagnostics agent knows how to access and find any of the normal Windows diagnostics sources, and then moves and collates the data into Windows Azure storage. The agent can even handle full dumps in the case of an unhandled exception in one of your processes.

You must configure the agent to have access to a cloud storage account. This is where it will place all of your data. Depending on the source of the data it will either place the information in blob storage (such as the case of a traditional log file), or it will the information in a table. Some information is stored in a table because of the nature of the data collection activity. Consider when you are collecting data from Performance Monitor. This data is normally stored in a special file with a `.blg` extension. While this file could be created and stored in blob storage, you will still have the hurdle of merging several of these files to make any sense of the data. And the information isn't easily viewed in notepad. You generally want to query that data. For example, you might want to find out what the CPU and memory pressure on the server were for a given time, when a particular request failed to process.

Table 18.1 shows what the most common sources of diagnostics information is, and where the agent will store the data once it is collected. We will discuss how to configure the sources, logs and the (tantalizingly named) “arbitrary files” in later sections.

Table 18.1 Diagnostic data sources

Data Source	Default	Destination	Configuration
Trace Logs	Enabled	Azure Table	web.config trace listener
Diagnostic Infrastructure Logs	Enabled	Azure Table	web.config trace listener
IIS Logs	Enabled	Blob	web.config trace listener
Performance Counters	Disabled	Azure Table	PerformanceCounterConfiguration class
Windows Event Logs	Disabled	Azure Table	WindowsEventLogsBufferConfiguration class
IIS Failed Request	Disabled	Blob	web.config traceFailedRequests

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Logs			
Crash Dumps	Disabled	Blob	CrashDumps class
Arbitrary Files	Disabled	Blog	DirectoryConfiguration class

The agent doesn't just take the files and upload to storage. The agent can also configure the underlying sources to meet your needs. You can use the agent to start collecting performance data, and then turn the source off when you don't need it anymore. This is all done through configuration.

18.2.2 It's not just for diagnostics

We have been focusing pretty heavily on the debugging or diagnostics nature of the Windows Azure Diagnostics platform. Diagnostics is the primary goal of the platform, but you should think of it as a pump of information about what your application is doing. Now that you no longer have to manage infrastructure, you can focus you attention on managing the application much more than you have in the past.

Consider some of the possibilities in business that you might need to provide for, and as you continue to read this chapter, think about how the diagnostics tools can make some of these scenarios possible.

There are the obvious scenarios of troubleshooting performance and finding out how to tune the system. This is a common process where you drive a load on the system, and monitor all of the characteristics of the system to find out how it responds. This is a good way to find the limits of your code, and to perform A/B tests on your changes. A/B tests are when you test two possible options to see which leads to the better outcome.

Other scenarios are not technical in nature at all. Perhaps your system is a multi-tenant system and you need to find out how much work each customer does. In a medical imaging system, I want to know how many images are being analyzed, and charge them a flat fee per image. I can use the diagnostics system to safely log a 'new image' event, and then once a day move that to Azure storage to feed into my billing system.

Maybe in this same scenario you need a rock solid audit so you know exactly who has accessed each medical record so you can comply with industry and government regulations. The diagnostics system provides a clean way to handle these scenarios.

An even more common scenario might be in the analysis of the visitors and their behaviors on your application. Some advanced ecommerce platforms know how their customers shop. With the mountains of data collected over the years they can predict that 80% of customers in a certain scenario will complete the purchase or not. Armed with this data they can respond to the user's behavior and provide some way to increase the likelihood they will make a purchase. Perhaps this is a timely invitation to a one on one chat with a trained customer service person to help them through the process. The diagnostics engine can help your application monitor the key aspects of the user and the checkout process, providing feedback back into the ecommerce system to improve business. This is the 21st century version of a sales person in a store asking if they can help you find anything.

To achieve all of these feats of science with the diagnostic agent, we need to learn how to configure and use it properly.

18.3 Configuring the Diagnostic Agent

If you are writing your code in Visual Studio, the default Azure project templates will include code that automatically starts the diagnostics agent, inserts a listener for the agent in the web.config, and configures the agent with a default configuration.

You can see this code in the OnStart() method in the WebRole.cs file.

```
public override bool OnStart()
{
   DiagnosticMonitor.Start("DiagnosticsConnectionString");     #1
   RoleEnvironment.Changing += RoleEnvironmentChanging;
   return base.OnStart();
}
```

#1 Start the diagnostic agent.

Replace #1 in code and text with a cueball

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

#1 This starts the agent with the default configuration, all in one line. The line also points to a connection string in the service configuration that will provide access to the Azure storage account you want the data to be transferred to. If you are running in the development fabric on your desktop computer you can configure it with the well-known development storage connection string `UseDevelopmentStorage=true.` This well-known string provides all of the data necessary to connect with the local instance of development storage.

You will also need to create a trace listener for the diagnostics agent. This will allow you to write to the Azure trace log in your code. You can do this by adding the following lines in your web.config. If you are using a standard template this is probably already included.

```
<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>
```

Once the trace listener is setup you can use the trace methods to send information to any trace listeners. When you do, you will want to set a category for the log entry. This will help you filter and find the right data later on. You should differentiate between critical data that you will always want and verbose logging data that you will only want when you are debugging an issue. You can use any string for the trace category you want. But be careful and stick to a standard set of categories for your project. If the categories vary too much, for example having both 'critical,' 'crit,' and 'important,' it will be too hard to find the data you are looking for. To standardize on log levels you can use the enumerated type `LogLevel` in `Microsoft.WindowsAzure.Diagnostics`. To write to the trace log you would use a line like the following.

```
using System.Diagnostics;

System.Diagnostics.Trace.WriteLine("Information", string.Format("Page [CA]loaded on {0}",
System.DateTime.Now));

System.Diagnostics.Trace.WriteLine("Critical", "Failed to connect to [CA]database.");
```

Only people with access to your trace information, using the diagnostics API will be able to see the log output. With that being said we do not recommend exposing sensitive or personal information in the log. Instead of listing the person's social security number, refer to it in an indirect manner. Perhaps by logging the primary key in the `customer` table. That way if you really need the social security number you can look it up easily, but it won't be left out in plain text for someone to see.

Another benefit of using trace is that the trace output will appear in the DevFabric UI, like in figure 18.2.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 18.2 When writing to the Trace channel, the entries are stored by the Windows Azure Diagnostics trace listener to the Azure log, which can then be gathered and stored in your storage account for analysis. The trace output is also displayed in the DevFabric UI during development.

At a simple level this is all you need to start the agent and start collection of the most common of data. The basic diagnostics setup is almost done for you out of the box because of the default configuration that it comes with.

18.3.1 Default Configuration

When the diagnostics agent is first started it will start with a default configuration. The default configuration collects the Windows Azure trace, diagnostic infrastructure logs, and IIS 7.0 logs automatically. These are the most common sources you will likely care about in most situations.

When working with the configuration of the agent you are likely to follow a common flow. You will grab the current running configuration (or a default configuration) from the agent, adjust it to your purposes, and then re-start the agent.

Figure 18.3 Use the GAS process to configure and work with the diagnostics agent. Grab the config, adjust the config, and then start the agent. Sometimes you will grab the default config, and sometimes the running config.

In the next few sections we will look at some of the common configuration scenarios, including how to filter the log for the data you are interested in before it is uploaded to storage.

By default the agent will store about 4GB of data locally, and age out data automatically when the limit is reached. You can change these settings if you want, but most people will leave them as is, and just transfer the data to storage for long term keeping.

While the agent will age out data locally to the role instance, the data retention of data once moved to Azure storage is up to you. You can keep it there forever, dump it periodically, or download it to a local disk. Once it has

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

been transferred to your account the diagnostics agent doesn't touch your data again. The data will just keep piling up if you let it.

18.3.2 Diagnostic Host configuration

You can change the configuration for the agent with code running in the role that is collecting data, code in another role, or code that is running outside of Azure (perhaps a management station in your datacenter.)

CHANGING THE CONFIGURATION IN A ROLE

There will be times when you want to change the configuration of the diagnostics agent from within the role the agent is running in. This is most likely going to be during an `OnStart` event, while an instance for your role is starting up. You could change the configuration at any time, but changing it during startup is the most likely.

Listing 18.1 - Changing the configuration in role at runtime

```
using Microsoft.WindowsAzure.Diagnostics;

public override bool OnStart()
{
   var config = DiagnosticMonitor.GetDefaultInitialConfiguration(); #1

   config.PerformanceCounters.DataSources.Add(
    new PerformanceCounterConfiguration()
    {
      CounterSpecifier = @"\Memory\Available MBytes", #2
      SampleRate = TimeSpan.FromSeconds(5.0)                    #3
    });

config.PerformanceCounters.ScheduledTransferPeriod =[CA] TimeSpan.FromMinutes(1.0);

   DiagnosticMonitor.Start("DiagnosticsConnectionString", config);

   return base.OnStart();
}
```

#1 Grabs the default configuration
#2 Tracks the amount of available memory
#3 [write a short inline annotation here]

Replace #1, #2, #3 with cueballs in code and text

In the sample in listing 18.1 we are changing the configuration during the `OnStart` method for the role instance. #1 The first step to change the configuration is to grab the default configuration from the diagnostics agent manager. This is a static method and it gives you a common baseline to start building up the configuration you want running.

In our sample we are adding a performance counter called `\Memory\Available MBytes`, #2. The `CounterSpecifier` property is the path to the performance counter. You can easily find these if you use the Performance Monitor like in figure 18.4.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 18.4 Finding the counter specifier using PerfMon. Browse to the proper category on the left, and then choose a counter and click add. You join the category name and counter name with slashes just like a folder path. In this example we have '\Memory\Available Bytes'.

Browse to the counter you want to track and you can find the specifier (which is like a file path) in the corner. We want the agent to sample that performance counter every second, #3. To do this we are using the `PerformanceCounterConfiguration` class. There is a configuration class for each data source the agent has access to. For each piece of data you want collected you need to create the right type of configuration object, and add it to the matching configuration collection, in this case the `PerformanceCounterConfiguration` collection.

We also want to aggressively upload our performance counter data to Azure storage. Normally this value might be set to 20 minutes, or longer, but in this case we will set it to once a minute. Each data source will have its own data sources collection, and its own transfer configuration. This will allow you to transfer different data sources at different intervals. For example, you could transfer the IIS logs once a day, and transfer the performance counters every five minutes.

Finally, we start the diagnostics agent. We need to provide it the connection string to the storage account we want the data uploaded to, and the configuration object we just built. The connection string defaults to a value of `DiagnosticsConnectionString`, which is an entry in the cloud service configuration file. When you are playing with this on your development machine you would set the value to `UseDevelopmentStorage=true`, but in production you would set it to be a connection string to your storage account in the cloud.

The data will be uploaded to different destinations depending on the data source. In the case of performance counters the data will be uploaded to an Azure table called WADPerformanceCountersTable.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Timestamp	EventTickCount	DeploymentId	Role	RoleInstance	CounterName	CounterValue
2010-02-03 03:05:24	634007630871710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3452
2010-02-03 03:05:24	634007630921730000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3459
2010-02-03 03:05:24	634007630971710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3459
2010-02-03 03:07:04	634007631021710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3460
2010-02-03 03:07:04	634007631071700000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3462
2010-02-03 03:07:04	634007631121720000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3462
2010-02-03 03:07:04	634007631171710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3462
2010-02-03 03:07:04	634007631221710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3460
2010-02-03 03:07:04	634007631271710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3432
2010-02-03 03:07:04	634007631321870000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3426
2010-02-03 03:07:04	634007631371710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3419
2010-02-03 03:07:04	634007631421710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3422
2010-02-03 03:07:04	634007631471710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3424
2010-02-03 03:07:04	634007631521710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3425
2010-02-03 03:07:04	634007631571700000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3426
2010-02-03 03:08:04	634007631621710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3426
2010-02-03 03:08:04	634007631671710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3425
2010-02-03 03:08:04	634007631721720000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3427
2010-02-03 03:08:04	634007631771700000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3427
2010-02-03 03:08:04	634007631821710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3427
2010-02-03 03:08:04	634007631871710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3421
2010-02-03 03:08:04	634007631921710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3420
2010-02-03 03:08:04	634007631971710000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3417
2010-02-03 03:08:04	634007632021700000	deployment(52)	NinjaWebSite	deployment(52).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3430

Figure 18.5 The performance counter data is stored in a table called WADPerformanceCountersTable. In our example we are tracking the amount of available memory. You can see that the available memory starts at 3,452 MB and slowly drops to 3,436 MB.

In figure 18.5 you can see the results of our performance counter configuration. The agent tracked the available memory every five seconds, and stored that in the table. The entries were uploaded from the role instance to the table every minute, based on our configuration. The high order of the tick count is used as a partition key so that querying by time is made fast and easy, which is the most likely dimension to be queried on.

The RoleInstance column will contain the name of the instance, to differentiate entries across the different role instances. In this case we only had one instance.

Tracking log data can generate a lot of data. To make all of this data easier to use the diagnostics agent supports the concept of filters.

FILTERING THE DATA UPLOADED

The amount of data collected by the diagnostics agent can get quite voluminous. While you might want to track a great deal of data, when you are trying to solve a particular problem you may only want a subset of data to look through. The diagnostic agent configuration provides for filtering of the results.

The agent will still collect all the data locally. The filter is only applied when the data is uploaded to Azure storage. This can narrow down the data you need to sift to, but it will also make your transfers faster, and cost less in storage. You can simply set this property to filter based on the log level of the records indicated.

```
transferOptions.LogLevelFilter = LogLevel.Error;
```

All of the log data will remain local to the agent, and it will only upload the entries that match or exceed the filter level you set.

CHANGE CONFIGURATION FROM OUTSIDE THE ROLE

Being able to change the configuration inside the role instance is nice, but is likely only done during the startup of the instance. Dynamic changes to the configuration are more likely to come from outside the role instance. The source would likely either be an 'overseer' role that is monitoring the first role, or a management application of sorts that is running on your desktop.

The agent's configuration is actually stored in a file local to the role instance running the agent. This file is polled, by default, every minute for any configuration changes. You can change the polling interval if you want

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

using the `DiagnosticMonitorConfiguration.ConfigurationChangePollInterval` property. This property can only be set from within the role the agent is running in.

To update the configuration remotely, either from another role, or from outside Azure we are going to use two classes. The `DeploymentDiagnosticManager` class is a factory that returns diagnostic managers for any role you have access to. We will use this manager to change the configuration remotely by using it to create `RoleInstanceDiagnosticManager` objects. Each `RoleInstanceDiagnosticManager` object represents a collection of diagnostic agents for a given role, one for each instance running in that role.

Once we have this object we can make changes to the configuration like we did in the prior section. The trick is that we have to change the configuration for each instance individually.

Listing 18.2 - Remotely changing the configuration of a role's diagnostic agent

```
using Microsoft.WindowsAzure.Diagnostics.Management;
using Microsoft.WindowsAzure.Diagnostics;
using Microsoft.WindowsAzure.ServiceRuntime;

DeploymentDiagnosticManager myDDM = new
[CA]DeploymentDiagnosticManager(RoleEnvironment.GetConfigurationSettingValue("DiagnosticsConne
ctionString"), txtDeploymentID.Text);

var myRoleInstanceDiagnosticManager =
myDDM.GetRoleInstanceDiagnosticManagersForRole("NinjaWebSite");
PerformanceCounterConfiguration CPUTime =                                          #1
new PerformanceCounterConfiguration()
{
CounterSpecifier = @"\Processor(_Total)\% Processor Time",
SampleRate = TimeSpan.FromSeconds(5.0)
};


         foreach (var instanceAgent in myRoleInstanceDiagnosticManager)         #2
         {
         DiagnosticMonitorConfiguration instanceConfiguration =
[CA]instanceAgent.GetCurrentConfiguration();
                         instanceConfiguration.PerformanceCounters.DataSources.Add(CPUTime);
#3

         instanceAgent.SetCurrentConfiguration(instanceConfiguration);
         }
```

#1 Create a new configuration
#2 Iterate over the instance agents
#3 Add the new performance counters

Replace #1,#2,#3with cueballs in code and text

Looking at listing 18.2 we can see how updating the configuration for a running role is done. We first need to get an instance of the `DeploymentDiagnosticManager` for our deployment. There is one object for your whole deployment, overseeing all the roles in your deployment. You will need to give it a connection string to your storage account for logging. This constructor doesn't take a configuration element like the `DiagnosticsMonitor` class does. You have to pass in a real connection string, or a real connection. We are grabbing the string out of the role configuration with call to `GetConfigurationSettingValue`.

From there we ask for a collection of `RoleInstanceDiagnosticManager` objects for the particular role we want to work with. In our case we are changing the configuration for the NinjaWebSite role. We will get one `RoleInstanceDiagnosticManager` object for each instance running the NinjaWebSite role.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

EventTickCount	DeploymentId	Role	RoleInstance	CounterName	CounterValue
634007737309650000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Memory\Available MBytes	3412
634007737359870000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Processor(_Total)\% Processor Time	26.455785
634007737359870000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Memory\Available MBytes	3410
634007737409610000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Processor(_Total)\% Processor Time	12.953047
634007737409610000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Memory\Available MBytes	3409
634007737459610000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Processor(_Total)\% Processor Time	21.378814
634007737459610000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Memory\Available MBytes	3418
634007737509620000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Processor(_Total)\% Processor Time	10.321144
634007737509620000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Memory\Available MBytes	3418
634007737559600000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Processor(_Total)\% Processor Time	9.487033
634007737559600000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Memory\Available MBytes	3418
634007737609600000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Processor(_Total)\% Processor Time	10.771193
634007737609600000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Memory\Available MBytes	3419
634007737659610000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Processor(_Total)\% Processor Time	9.679234
634007737659610000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Memory\Available MBytes	3419
634007737709630000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Processor(_Total)\% Processor Time	18.414919
634007737709630000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Memory\Available MBytes	3418
634007737759600000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Processor(_Total)\% Processor Time	22.143216
634007737759600000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.1	\Memory\Available MBytes	3411
634007737817950000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Processor(_Total)\% Processor Time	23.094753
634007737817950000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3404
634007737867950000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Processor(_Total)\% Processor Time	14.653984
634007737867950000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3404
634007737917950000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Processor(_Total)\% Processor Time	8.119289
634007737917950000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3403
634007737967950000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Processor(_Total)\% Processor Time	15.763015
634007737967950000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3395
634007738017950000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Processor(_Total)\% Processor Time	34.482345
634007738017950000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3395
634007738068060000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Processor(_Total)\% Processor Time	26.687914
634007738068060000	deployment(54)	NinjaWebSite	deployment(54).NinjaDoughnuts.NinjaWebSite.0	\Memory\Available MBytes	3394

Figure 18.6 You can change the configuration of the Windows Azure Diagnostic agent running in each instance quite easily. In this case we added the % Processor Time performance counter to the agent. You can do this remotely, even from outside of Azure.

#1 Next we turn to creating the new part of the configuration we want to add to the agent. In this case we will build another performance counter data source. It will track the percentage of CPU in use. #2 Then we will iterate over our collection from and add #3 the new performance counter `CPUTime` to the current configuration. This is different from when we are changing the configuration in role. In this case we want to add to the configuration, not completely replace it. Finally, we update the configuration for that instance. This will update the configuration file for the diagnostic agent. When the agent polls for a configuration change it will pick up the changes and recycle to load them.

In our sample we put this code in the role, but this code would work running from any application running outside of Azure as well. The only difference would be how you would provide the connection string to storage, and how you would provide the deployment id.

We have looked at the standard data sources for Windows Azure Diagnostics, but there is one hole remaining. What if I want to manage a diagnostic source that isn't on the official list? That is where the escape hatch called "Arbitrary Diagnostics Sources" comes into play.

18.3.3 The other data sources

Up until now we have looked at how to configure a performance counter data source and how to enable the trace listener in the web.config file. There are other data sources that are configured in very similar ways. We will now briefly look at each.

CRASH DUMPS

Crash dumps are the log file of last resort. They really aren't a log file, but a dump of the status of the computer when a horrible problem has happened. There are two types of crash dumps. The normal dump includes a copy of

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

all the memory on the machine while the dump called 'mini dump' holds most important information without a complete copy of everything.

Crash dump files are stored in the local data buffer and transferred with the common logs. You can choose which size dump you want by passing in true to the `EnableCollection` or `EnableCollectionToDirectory` method for a full dump and false for a mini dump.

IIS Failed Request Logs

The failed request logs for IIS are a new feature in IIS7.0. IIS tracks log data for requests as they come through but only keeps the data if certain configurable conditions are met. Perhaps if a response takes too long to complete the request log data would be kept. If the response is completed fast enough then the log buffer is flushed. You can configure how IIS manages this process in the `tracing` section of `system.webServer` part of your web.config. Once you configure failed request tracing for IIS in this way the logs will be collected with the rest of the data logs.

Windows Event Logs

Windows event logs can provide important clues to serious problems with your applications. Some applications create custom event log sources for their own logging. The diagnostic agent in Azure can collect these logs and transfer them to storage for you.

You need to subscribe to the event data that you want to receive using an XPath expression. Because of the security profile your processes run on you will not be able to read the security windows event log. If you add this log to your configuration nothing will be logged, and it won't work properly until you remove it.

The most likely expression you will use is listed below.

```
diagConfig.WindowsEventLog.DataSources.Add("System!*");
```

It will grab everything from the Windows System Event log, which is where you usually want to start your investigations.

18.3.4 Arbitrary Diagnostic Sources

Windows Azure Diagnostics covers a lot of the diagnostic sources you might normally use to troubleshoot an issue with your system. It covers IIS logs, performance counters, Windows event logs, and several other things. Overtime you are likely to come in contact with a source of your own devising, some log you are creating that you need to track as well. Perhaps this is custom billing data, or a compressed log of the images used in production, or it might just be the output of a third party logging framework you have chosen to use.

You can have the agent transfer anything you want. You just need to get that data into a file in a designated folder. When you configure the agent you tell it which custom directories you want it to monitor. In order to do this you will want to configure some local storage and then write your log files to it.

Each data source has a configuration class that you add to the agent's configuration, and custom log locations are not any different. We will use the `DirectoryConfiguration` class to tell the agent to monitor a folder. We can set how large that directory is allowed to become, as well as scheduled transfer characteristics.

```
DirectoryConfiguration specialLogsDC = new DirectoryConfiguration();

specialLogsDC.Path = [CA]RoleEnvironment.GetLocalResource("specialLogs").RootPath;          #1

specialLogsDC.Container = "speciallogs";                                  #2
                instanceConfiguration.Directories.DataSources.Add(specialLogsDC);
```

#1 Source directory.
#2 Destination storage container.

Windows Azure Diagnostics is a powerful tool you can use to help troubleshoot and diagnose problems. It isn't just for problems, as we have discussed. It can also be used to help monitor the behavior of the system, or the actions of your users. While WAD is wonderful, it is only a source of data. You still need to analyze the data to turn it into information, and then take action. Sometimes the action you need to take is to change the configuration of the service model you are running your application in. Perhaps you need to add some instances to respond to a spike in traffic.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

18.4 Transferring Diagnostic Data

The diagnostic agent does a great job of collecting all of the local data and storing it on the machine it is running on. But if the diagnostic information is never moved to Azure storage it won't be on any good to anyone. This is where transfers come into play.

There are two types of transfers, one of which you have already seen in play. It is called the scheduled transfer. This sets up a timer and transfers the related data on a regular basis to your storage account.

The second type of transfer is an on demand transfer. This is more likely going to be done when you have a special request of the data.

Each data source category has its own transfer schedule. For example, we can transfer performance counter data at a different rate than we transfer the IIS logs.

18.4.1 Scheduled Transfer

In our sample in listing 18.2 at #4 we see that we are setting up a scheduled transfer of the performance counter data to occur every minute. As we covered earlier, transferring every minute is quite aggressive, and is probably only reasonable in a testing or debugging environment.

In our next sample we are going to transfer the IIS logs on a daily basis to storage. The IIS logs are automatically captured by default by the diagnostic agent so we don't need to add them as a data source. We can set the transfer interval to once a day with this line.

```
instanceConfiguration.Logs.ScheduledTransferPeriod = TimeSpan.FromDays(1.0);
```

Any log files that are captured are sent to a container in blob storage, not to a table. Each transfer will result in one file in the container. There will be a container hierarchy similar to what you would see on the real server. This means that our IIS logs will be in a similar folder structure that we are used to.

If there are logs that are collected that you don't want transferred you can set the `ScheduledTransferPeriod` to zero. This will disable the transfer of any data for that data source. I typically do this for the Azure Diagnostics log themselves, at least until I am troubleshooting a problem with the diagnostic agent itself.

Next we are going to discuss how you can trigger a transfer on demand.

18.4.2 On Demand Transfer

An on demand transfer lets you configure a onetime transfer of the diagnostics data. This gives you the ability to pick and choose what is transferred and when. The most likely scenario is you want an immediate dump of logs when you see something critical is happening. You can setup an on demand transfer in much the same way you would a normal transfer. There are some differences though.

In our sample, in listing 18.3, we are causing the on demand transfer to happen from within one of the instances, but it is also possible for the transfer to be initiated from outside the role with an administrative application.

Listing 18.3 - Initiating an on demand transfer

```
DeploymentDiagnosticManager myDDM = new
[CA]DeploymentDiagnosticManager(RoleEnvironment.GetConfigurationSettingValu[CA]e("DiagnosticsC
onnectionString"), txtDeploymentID.Text);

var myRoleInstanceDiagnosticManager =
[CA]myDDM.GetRoleInstanceDiagnosticManagersForRole("NinjaWebSite");

DataBufferName datasourceToTransfer = DataBufferName.PerformanceCounters;      #1
OnDemandTransferOptions transferOptions = new OnDemandTransferOptions();

transferOptions.From = DateTime.UtcNow - TimeSpan.FromHours(1.0);              #2
transferOptions.To = DateTime.UtcNow;                                          #2
transferOptions.NotificationQueueName = "transfernotificationqueue";           #3

foreach (var instanceAgent in myRoleInstanceDiagnosticManager)
{
  Guid requestID = [CA]instanceAgent.BeginOnDemandTransfer(datasourceToTransfer,
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
transferOptions);                                                    #4
  System.Diagnostics.Trace.WriteLine("on demand started:" + [CA]requestID.ToString(),
"Information");
}
```

#1 Transfers performance counters
#2 Selects time filter for data to be sent
#3 Provides name for transfer-complete notification messages
#4 Starts the transfer for each instance in the role

Like with the remote configuration sample we need to get a reference to the `RoleInstanceDiagnosticManagersForRole` class. This will let us work with the configuration manager for each instance. #1 In our sample we are going to be transferring the performance counter data over to an Azure table. This transfer will include all counters you might have running from prior configuration changes.

The `OnDemandTransferOptions` class is used to configure how the transfer should happen. It has several parameters that you will want to set. #3 We are going to set a time filter, in essence telling the agent to only send over the performance counter data that has been generated in the last one hour. If you are transferring a log you can also specify a log level filter (informational, critical, etc.).

When we start the transfers, each instance will perform its own transfer. The transfer operation is an asynchronous operation; you just start it and walk away. In most cases you will want to know when the transfers have completed so that you can start analyzing the data. You can have the transfer agent notify you when the transfer for an instance is complete by passing in a #3 queue name. #4 As each transfer is started you will be given a unique id to track. You can see these messages in figure 18.7.

Figure 18.7 You can configure an on demand transfer to notify you when a transfer is complete by placing a message in a queue you specify. In this case the role had two instances, so two transfers were completed, one for each instance.

When each transfer is complete the agent will drop a small message onto the queue you designated with that same id. This lets you track which transfers have been completed.

A great place to use an on demand transfer would be in the `OnStop` method in your `RoleEntryPoint` override class. Whenever a role is being shutdown, either intentionally or otherwise this method will fire. If you do an on demand transfer in this method you will save your log files from being erased during a reboot or a move. This can help you troubleshoot instance failures.

Now that we know how to get data on what is happening with your service we need to look at the APIs that will help you do something about what you see happening.

18.5 Using the Service Management API

Once your application is up and running in Azure you are going to want to automate some of the management functions. This can include scaling your roles, changing configuration, and automating deployments. Almost anything you can do through the Azure portal you can do through the service management API.

The service management API is built like all of the other API's in Azure. It uses REST and XML under the hood wrapped in a pleasant .NET library. You can use the service management API directly with REST, but most people will either use the library, or use a tool that calls the APIs.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

All of the management API's we are about to discuss can be called from within or from outside Azure. All management calls are free, they incur no cost to call or execute. The Azure team has said that they monitor the use of the API's and can throttle back your calls if they are abused.

To start using the service management api, you will need to configure your account with certificates for api authentication. Once you have done that you will be able to send it commands. After we show you that, we are going to look at how you can work with your services and containers, how to automate a deployment to the cloud, and how you can use the management api to scale your service up or down.

18.5.1 What the API doesn't do

A little earlier we said the service management API can do almost as much as the portal. There are still a few things that you have to use the portal for.

- Access to billing data

  You can only access the billing data from the portal. The portal has several tools you can use to monitor your usage and billing in near real time. This allows you to estimate your charges as they occur. The final numbers will be crunched at the end of the month to generate you bill.

- Creating subscriptions

  You will also need to use the portal when creating a subscription (the basis for your Azure account), and any compute or storage services. Once the subscription, and services are created everything else can be done from the management API.

In order to make calls with the API you will need to sign them with a certificate, which we'll discuss next.

18.5.2 Setting up the management credentials

The service management API has a lot of power, so all of its calls and responses must be secure. All calls are transferred over HTTPS, using a signed certificate that you associate with your Azure account. Whether you are calling the REST by hand or using the .NET library you will need to attach a certificate your Azure account trusts.

You can use any x.509 v3 certificate that you want. Since you have control over which certificates to your account trusts, these can include certificates you self-sign. You can also use certificates that you have purchased from a certificate authority like VeriSign.

Your account can hold up to five certificates. This allows you to distribute those certificates to different people or processes, and then to eventually revoke them if you need to. All a person needs to use the management API on your services is that certificate, and your subscription number. We will look at how to revoke a certificate later in this section.

In order to setup our management certificates we need to have a certificate to upload. We are going to walk through the process of creating a certificate locally and then uploading it to our account.

Our goal is to create a .cer file which holds the public key for your certificate. You never share the private half of the key. This public key will be uploaded to Azure, and Azure will use it to verify that your private key was used to sign the management API request.

We need to use IIS7 to create a self-signed certificate. Open up the IIS manager and look for the **Features** view. Listed there will be a link for **Server Certificates**. Click **Create Self-Signed Certificate** from the **Actions** pane and follow the steps. You will to give the certificate a name. This name will be used whenever you are working with the certificate.

You can also use the Visual Studio Command prompt to create a self-signed certificate. Open up the command prompt, and make sure to run it as an administrator. Then enter the command below.

```
makecert -r –pe -n "CN=CompanyName" -a sha1 -len 2048 -ss My "filename.cer"
```

This will create a certificate that you can use in the local directory.

Importing a certificate is as easy as logging into your Azure portal and going to the Account tab. Once there choose 'Manage My API Certificates.' This screen will display what certificates you have uploaded, and will also let you upload a new certificate. Your certificate must be in a .cer file. If you have a different format you can easily convert it by double clicking the certificate in Windows and choosing the details tab, as in figure 18.8.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 18.8 We have imported two X509 certificates. One for 'Lazy Dev' and one for 'Awesome Dev.' Certificates are used to authenticate to Azure when using the Service Management API. Guess who's certificate we are going to revoke in the next example.

Since you can have up to five certificates in your account at a time you should take advantage of them. For each person or system that is using the management API should have its own certificate. If you do this you will have an easy way to revoke their access. To revoke a certificate just click the delete X icon next to the one you want to revoke.

You will need to attach your certificate to each request you send the API. When you do this the message is signed with your private key, which only you should have. Then when Azure receives your message it will check that the message came from you by opening it with the public key you uploaded in the .cer file.

Now that we are ready, let's look at just some of the things we can use the service management API for.

18.5.3 Listing your services and containers

You can save a lot of time if you learn how to automate your deployments instead of doing them by hand through the portal. We are going to start by showing some code you can use to get a list of the service and storage accounts you have created in Azure. This code will be fairly primitive and will be using the REST API directly. We will eventually start using a tool that will abstract away the raw REST so that we have something nicer to work with.

Listing 18.4 - Querying for the list of services with REST

```
var request = [CA](HttpWebRequest)WebRequest.Create(
[CA]"https://management.core.windows.net/7212af99-206f-dem0-9334-
[CA]380d0f841d0b/services/hostedservices");                          #1

request.Headers.Add("x-ms-version:2009-10-01");                      #2
request.ClientCertificates.Add([CA]
X509Certificate2.CreateFromCertFile(@"C:\[CA]
 \awesomedev.cer"));                                                 #3

var responseStream = request.GetResponse().GetResponseStream();

var services = XDocument.Parse(new [CA]StreamReader(responseStream).ReadToEnd());
```

#1 Ssubscription id musrt be in uri
#2 Specifies version of the management service to use

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

#3 Adds your certificate to the request

Replace #1, #2, #3 with cueballs in code and text

We are going to use the `WebRequest` class to work with the REST call we will be making. We simply need to pass in the URI of the call we want to make. In this case, `#1` since we want a list of the services we have created we will pass in a string that includes the root of the call, `https://management.core.windows.net/`, our subscription id (which can be found on your accounts tab in the portal). Every request into the management service will need this base address. Since we want a list of the hosted services we have in our account we will add `/services/hostedservices` to the end of our URL.

We will also need to attach a version header and our certificate for authentication. `#2` The version header tells Azure which version of the management service we intend to call. Right now the latest version to call is the one published in October of 2009, so we will use that one. The certificate is easily attached at line `#3`. In this case we are attaching it from the file we generated above. We could also have easily used the certificate that is in our secure certificate store.

When we run this code, the result we get back in raw XML format is shown below in listing 18.5. All three services that are running in my subscription are listed in `HostedService` elements.

Listing 18.5 - The raw XML that comes back from our request

```
<HostedServices xmlns="http://schemas.microsoft.com/windowsazure"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <HostedService>
    <Url>https://management.core.windows.net/7212af99-206f-dem0-9334-
380d0f841d0b/services/hostedservices/aiademo1</Url>              #A
    <ServiceName>aiademo1</ServiceName>                          #B
  </HostedService>
  <HostedService>
    <Url>https://management.core.windows.net/7212af99-206f-dem0-9334-
380d0f841d0b/services/hostedservices/aiademo2</Url>
    <ServiceName>aiademo2</ServiceName>
  </HostedService>
  <HostedService>
    <Url>https://management.core.windows.net/7212af99-206f-dem0-9334-
380d0f841d0b/services/hostedservices/aiademo3</Url>
    <ServiceName>aiademo3</ServiceName>
  </HostedService>
</HostedServices>
```

#A Management URL to the service.
#B Simple name for our service.

Once you have received the XML back you can use something like LINQ to XML to parse through the results. You will want the URL for each service for later when you are calling back to reference that particular service. We can use the same code to get a list of storage accounts in our Azure account by changing the address of the request from `/hostedservices` to `/storageservices`.

You will be able to use these endpoints to see the storage and service accounts you have. This part of the API is readonly. In order to create service or storage accounts you will have to use the portal.

Now that we can make simple queries of the management service, let's flip over to csmanage.exe and use it to deploy a simple web application to Azure.

18.5.4 Automating a deployment

While you can do everything you need through the naked REST API of the service management service, it is a lot easier to use something that provides a higher level of abstraction. REST is fine, but is too low level for me on a daily basis.

We are going to use csmanage.exe for the rest of this chapter. This small utility is provided by the Azure team and can be found at http://code.msdn.microsoft.com/windowsazuresamples.

Even though we are using a higher level tool we still need to provide the tool with the subscription id of our Azure account and the thumbprint for the certificate we will be using to manage our account. You can enter these into the configuration file for the tool, csmanage.exe.config. These are both easiest found on the Azure portal.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

The csmanage application can work with one of three resources online, and each command you send will include a reference to which resource you want to work with. You can work with hosted services, storage services, or affinity groups.

To get a simple list of the hosted services we have in our account we can execute the following command at a command prompt. This command executes the same query we did by hand in a section above.

```
csmanage.exe /list-hosted-services
```

When we run this command the application will connect to our account in the cloud and return a list of the services we have. In this sample I have just one hosted service.

Figure 18.9 We can use the csmanage command line tool to execute management commands against our account in the cloud. In this example we used a simple command to list the hosted services we have.

Our next task is to create a deployment of our application. The first step is to create the package for our code. You have probably done this a thousand times by now, but right click on the Azure project and choose Publish. When we are using the management service we can only deploy a service package that is in blog storage, we can't upload it as part of the actual deployment command.

We can however upload the package through any package we want. We just want to upload the cspkg file. The configuration file will be uploaded when we run the deployment command.

Listing 18.6 - Pushing a deployment to the cloud from the command line

```
csmanage.exe
    /create-deployment       /slot:staging
    /hosted-service:aiademo1
    /name:ninjas
    /label:build1234
    /config:ninja.cscfg                                                  #1
    /package:http://aiademostore.blob.core.windows.net/[CA]
deployments/NinjaDoughnuts.cspkg                                         #2
```

#1 Local path of configuration file to upload
#2 URL of the package to deploy

This is a lot to type in by hand. The most common use of csmanage will be within an automated deployment script. You could make the deployment completely hands off with enough script and PowerShell. When we execute the above command there will be a slight pause as the configuration is uploaded and the management service deploys our bits.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

```
C:\Windows\system32\cmd.exe
C:\Users\brprince\Desktop\csmanage-10-10-2009\csmanage-10-10-2009>csmanage.exe /
create-deployment /slot:staging /hosted-service:aiademo1 /name:ninjas /label:bui
ld1234 /config:ninja.cscfg /package:http://aiademostore.blob.core.windows.net/de
ployments/NinjaDoughnuts.cspkg
Using certificate: CN=AiA
Creating Deployment... Name: ninjas, Label: build1234
Operation ID: dd1ac36e0d46402e8af03a19da7e1ebd
HTTP Status Code: Accepted
StatusDescription: Accepted
Waiting for async operation to complete:
.........Done
Operation Status=Succeeded

C:\Users\brprince\Desktop\csmanage-10-10-2009\csmanage-10-10-2009>
```

Figure 18.10 The output from csmanage when we script out a deployment to staging. We are deploying our ninja doughnut application, which has nothing to do with doughnuts or ninjas.

We don't want to just upload and deploy our code we need to start it so that fabric controller starts up our site. The command is quite simple.

```
csmanage /update-deployment /slot:staging /hosted-service:aiademo1 /status:running
```

We are telling the management API we want to update the deployment of our service in the staging slot to 'running'. We could use the same command to set the status to 'suspended'. Once you set the status to 'running' it can take a few minutes for the fabric controller to get everything up and running. You can check on the status of each instance of your roles by using the 'view-deployment' command.

```
csmanage /view-deployment /slot:staging /hosted-service:aiademo1
```

When you execute this command you will get a detailed view of each instance. In this case, both of our instances were 'busy'. If we just wait a few minutes they will flip to 'ready.'

```
C:\Windows\system32\cmd.exe
C:\Users\brprince\Desktop\csmanage-10-10-2009\csmanage-10-10-2009>csmanage /view
-deployment /slot:staging /hosted-service:aiademo1
Using certificate: CN=AiA
Getting Deployment
Name:ninjas
Label:build1234
Url:http://49dfc896de8a43d7bbdc47c52e3cf5a9.cloudapp.net/
Status:Running
DeploymentSlot:Staging
PrivateID:49dfc896de8a43d7bbdc47c52e3cf5a9
UpgradeDomainCount:2
RoleInstanceList contains 2 item(s).
    RoleName: NinjaWebSite
    Role InstanceName: NinjaWebSite_IN_0
    Role InstanceStatus: Busy
    RoleName: NinjaWebSite
    Role InstanceName: NinjaWebSite_IN_1
    Role InstanceStatus: Busy
Operation ID: 2f0de1c14b5146a69849a658f6ef718e
HTTP Status Code: OK
StatusDescription: OK

C:\Users\brprince\Desktop\csmanage-10-10-2009\csmanage-10-10-2009>
```

Figure 18.11 We can check the detailed status of each role instance by using the 'view-deployment' option. In this case our instances are busy because we have just deployed the package. In a moment this will ready 'ready.'

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Once our instance status read 'ready' we can use a command to perform a VIP swap. This is when we swap the virtual IPs for production and staging, performing a clean cutover from one environment to the other. This is the simplest way to deploy a new version. You can always to an update domain walk. This is more complicated but gives a rolling upgrade capability.

To perform the VIP swap execute the following command. You have to have a deployment in each slot to perform a VIP swap through the csmanage or REST interfaces. If you are deploying for the first time to an empty environment either deploy your first version into the production slot, or just rerun the deployment command and adjust the slot parameter to `production`.

```
csmanage /swap-deployment /production-slot:ninjasV0 /source-deployment:ninjas /hosted-
service:aiademo1
```

Be careful when typing this command in. The third parameter 'source-deployment' is different than what you expect. Since the second parameter is 'production-slot', you would expect the third to be 'staging-slot'. The naming is not terribly consistent. I also wonder why I need to define the slot names at all. I can only have one slot of each time anyway.

Figure 18.12 We have successfully swapped the staging and production slots by performing a VIP swap. You have to provide the names of the deployments in each slot.

Now that we have swapped out to production, and have fully tested it, we can tear down the staging slot. We need to suspend the state of the servers, and then delete the deployment.

```
csmanage /update-deployment /slot:staging /hosted-service:aiademo1 /status:suspended
```

```
csmanage /delete-deployment /slot:staging /hosted-service:aiademo1
```

We need to execute these two commands back to back. You cannot delete a service while it is running, so you have to wait for the first command to suspend the service finish.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Figure 18.13 After finally deploying and swapping our new version to production we are suspending the staging environment and then tearing it down to stop the billing. This takes two steps. First we must suspend the service, and then delete it.

We have successfully automated the deployment of a cloud service, started it up, moved it to production, and then finally stopped and tore down the old version of the service. We have done this without having to use the portal. We still had to use the portal to upload our management certificates and create the initial service container.

Now to what people really care about in the cloud, dynamically scaling the number of instances that are running our service.

18.5.5 Dynamic scaling and changing configuration

One of the golden promises of cloud computing is the dynamic allocation of resources to your service. It is really cool that I can deploy a service from nothing to 20 servers, but we also want to be able to change that from 20 servers to 30 servers if we are hit with a spike of some sort.

As we have discussed in chapters past your service is based on a service model that is defined in your service configuration file. This file defines how many instances per role your service defines. There are three ways to change this file.

The first is to edit it online in the portal. This is the simplest way to change it, but is also the most primitive. You can't wire this up into an automated system or into an enterprise configuration management system.

The second method can use configuration files generated by your enterprise configuration management change system. You can upload a new file (version them with different file names so you can keep track of them) into blog storage, and then point to that when changing the configuration from the portal.

The third option lets you upload a new configuration through REST, and that is the service management REST API. If you don't want to use REST you can use csmanage, like we have been doing in the past few sections.

The fabric controller will respond in different ways depending on how you have changed your configuration. If you have changed anything besides the instance count of a role then the fabric controller will tear down and restart any instances for the affected role. This will bring that part of the service to a screeching halt while the instances are restarted. The gap of outage isn't very long, but it is there.

If you only change the number of instances then none of the instances are affected, unless you are reducing the number of running instances. If you increased the instance count the fabric controller will create new instances, deploy your bits, configure them, and wire them into the network and load balancer like when have you started the role instances from scratch. This process can take several minutes. You will not see an immediate availability so be patient. You should take this slight delay into the logic you are using to detect when you need to add instances.

```
csmanage /change-deployment-config /config:ninja_daytime.cscfg /slot:production /hosted-service:aiademo1
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

When you are using the management API (and csmanage) you have to upload a locally stored configuration file. In figure *****18.x, you can see that it took quite a while for the management service to spin up the new instance. The total time it took in this instance was around four minutes.

Figure 18.14 We have upload a new configuration file for our service. Because we only changed the number of instances for each role we haven't had any downtime. The fabric controller has spun up new instances to meet our needs.

If you have reduced the number of instances then the fabric controller uses an undocumented algorithm to decide which instances will be shed. They will be unwired from the load balancer so that it doesn't receive any new traffic, and then it is torn down.

Many people wonder why Azure doesn't auto scale for them. There are a couple of reasons, but they all fall into the "My kid sent 100,000 text messages this month and my cell phone bill is over a million dollars" category.

The first challenge for Azure with auto scaling your application is that Azure doesn't know what it means for your system to be busy. Is it the depth of the queue? Which queue? Is it the number of hits on the site? Each system defines the status of busy different than any other system. There are too many moving parts for a vendor such as Microsoft to come up with a standard definition that will make all of its customers happy.

Another reason is that Microsoft could be accused of too aggressively scaling up, and not scaling down fast enough just to increase the charges on your account. We don't think they would actually do that, but the second someone thinks the algorithm isn't tuned to their liking they would get sued over overbilling customers.

Another scenario this approach protects against is a denial of service attack (DOS). In these attacks someone tries to flood your server will an unusually high number of requests. These requests overcome the processing power on your servers and the whole system grinds to a halt. If Azure auto scaled up in this scenario you would come in on Monday the day after an attack with five thousand instances running in production. While you would get enough mileage points on your credit card to fly to the Moon and back for free, you probably wouldn't be too happy about it.

Microsoft has given us the tools to manage our scaling ourselves. We can adjust the target number of instances at any time in a variety of ways. All we have to add is the logic we want to use to determine what 'busy' means for us, and what we do in a busy state, or the opposite, in a slow state.

We will look at some approaches for how to manage this in the next section.

18.6 Better Together For Scaling

Everyone has the expectation that they will be able to dynamically scale their service in Azure. This is possible but requires some lifting on the developer's part. Over time vendors will provide this as a service on top of Azure. We will want to provide some sort of control over the amount of resources allocated to our service.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

In this section, we are going to follow the same model that our homes use for heating and cooling. Our homes are driven by a sensor to detect a healthy condition (is the temperature in a pleasant range), a mechanism to change that temperature, and some simple rules that keeps the heater from running for 24 hours straight. We can take this approach and apply it to our cloud service. We will instrument it with the diagnostic engine, use the manage api to control the infrastructure, and provide some code to control how all of that works. We want to respond to events and keep our system healthy without coming in on Monday to find we have 1,500 instances running in the cloud. We'll start with the thermostat.

18.6.1 The Thermostat

A thermostat in your home is a simple example of a control system. Other examples include the cruise control in your car, the autopilot in a plane, and many manufacturing systems.

Each of these systems has three components.

The first is the system itself, the car, the plane, the furnace in the house. In Azure this would be the service you are running. The system needs to have inputs to be able to control important aspects of the system. In the furnace example, we can send it a 'turn on' signal or a 'turn off' signal. This will cause it to generate more heat, or to stop generating heat.

The second component is the measurement device. This is the device that measures the control aspects of the system. In our house example it is the thermometer in the thermostat. In Azure this might be any number of things. On a thermostat we use a simple dial to determine if the house is too cold or too hot. The hard part for our systems in Azure is deciding what busy is for our system. It isn't a simple dial, but likely an amalgam of several inputs. Perhaps the depth of a messaging queue, the number of pending requests in the IIS queue, and the running average of response time for each web request. Every measurement point we want to monitor to help make the busy decision needs to be something we can measure across all of our instances.

Figure 18.15 A system control system you might have in your house. The fire is your furnace, the system we want to control. The thermometer measures the temperature of the house, and provides feedback on the performance of the system to the controller. The controller is the electronics in your thermostat, and tells the furnace to turn or off based on the input from the thermostat.

Busy is sometimes represented as an absolute measure. An example would be to define a concrete amount of time a response is allowed to take under normal conditions. You are either beneath or above that allotted time. Some definitions for a busy state are relative in nature. Saying your system is busy whenever there are more than 50 messages in the queue won't work very long. You would want to measure 50 messages per instance maybe. Then if you have five instances, a scenario where there are 200 messages in the queue is ok. In this way we scale up the definition of busy as we scale up the amount of available resources.

The last component is the control logic itself. This is the piece that determines if anything needs to be done, and how it should be done.

18.6.2 The Control System

Your control has only a few things that it can do with regards to Azure to manage performance or scale. In general the only thing it can do is either add or remove instances of roles. That's about it.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

Of course there are plenty of scale patterns you can implement, and some that will help a dramatic scale failure from happening. You will want to look into shunting, bulkheads, and partitioning.

You will want to instrument your control system itself so you know what decisions it is making and how. You want to be able to figure out what wrong when you service lumbers out of control and eats a village.

The control system can run as a simple role in Azure or as an on-premises application. While you might assume an on-premise is a better solution, consider that you will need the input of the diagnostics logs, and that means downloading them all the time to make decisions. Having the control for the system running in the cloud puts the code near the data, which will make it faster.

18.6.3 Risks, and managing them

There is a lot of risk with implementing an auto scaling component for your service, it is not trivial. There are a lot of issues and complexities you will have to take into consideration. If things go haywire, they can go haywire really bad.

In one direction you could end up with a large Azure bill if your code goes crazy and spins up four hundred instances. On the other hand, if it fails to work properly you will end up not responding to a busy state at all, leading to lost orders and unhappy users. There is a fine balance to scaling, and you will want some protective measures in place.

In your logic make sure you have an absolute upper boundary in place. No matter what is happening in the feedback system the scaler will not go above this boundary. If it reaches tis ceiling it should call a human and ask for help. You should set this boundary at something that is high enough to handle expected spikes (the big spike at the end of month), plus 15% for a buffer. Along with a ceiling you will want a floor to the scale value set. You may want to make sure you always have two instances for reliability. Of course some applications are ok with just a single instance, and others are ok to be completely shut down if there isn't any load.

At some point the spike that caused the controller to create all of these instances will pass. And once it does you will want your controller to start shedding instances to bring the amount of resources deployed back into an acceptable range for the current load.

You will want to pick an algorithm that matches how your load tends to fall off. If it tends to fall very quickly, a steep spike, then you will want an aggressive back off strategy. In this case you might use a halving technique where each polling cycle the controller deems you have excessive it cuts the amounts of resources by half, going from 32 instances to 16 for example.

In other services you want a slower back off process, only dropping one instance every time the measure drops a certain percentage. You will want to do extensive testing on the behavior of your controller to make sure it is working under stressful situations the way you want it.

Another risk is with the controller flooding the channel with conflicting messages. It is possible that if the polling cycle is too fast, and the traffic too unpredictable that you will end up sending conflicting message through the channel to the service. If you send a message to add an instance and then immediately follow it with a message to shed an instance, you will end up thrashing your infrastructure. You also don't want to accidentally send the 'add an instance' message several times when you only want one net new instance started. To avoid this problem you will need to make sure your controller is stateful, tracking what commands it has sent, and whether those have been executed yet. You might want to even suspend all polling until the chosen action is completed.

No matter how clever you get with your controlling logic makes sure that you always include wetware somehow. The controller should always have a way to notify a human as to its behavior. If you run into the 'we accidentally started up 400 instances last night' problem, you will likely have a 'don't have a job on Monday' problem.

We strongly recommend that you initially build tools that help you watch performance and easily maintain it. Keep the decision to add or remove instances in human control, at least until you have an absolute understanding of how the system responds to stress, and how it responds to increases and decreases in resources.

The cloud does a great job of abstracting away the need to manage the platform, but we still need to manage the application.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579

18.6.4 Managing Service Health

Managing service health is critical to your system. Just because the system is running in the cloud doesn't mean there aren't failures, and that it doesn't need to be managed. You still need to manage a system in the cloud, and taking into consideration all of the aspects that you do when the system is running on-premises. The cloud doesn't fix problems in a bad system it just makes those problems more obvious.

As you are building your system, think about how you are going to manage disaster recovery, backups, and the ongoing health of the system. The system will be very reliable within the Azure datacenter where the fabric controller can monitor it. But that doesn't protect you against the worst scenario, if that datacenter gets wiped out. In some scenarios it might be ok to not manage the slight risk of a whole datacenter disappearing. On the other hand, many companies spend a lot of money running duplicate data centers that are not near one another just in case. You need to take this into account. If you need to reduce the risk of depending on one datacenter, then you have a few options. Just to be clear, while the fabric controller will manage the state of your system in a datacenter, it will not, at this time, manage it across data centers. I think that in the fullness of time, the fabric will be that powerful, but at time it is likely to be renamed to Microsoft SkyNet.

If the loss to your business is not great, and you can deal with a few hours downtime, you could simply plan on redeploying to another datacenter in the event of a catastrophe. You would need to keep a copy of the production bits, and service configuration handy. You would also need a backup of any data in the cloud. With these in hand, you could completely redeploy to another Azure data center in a matter of minutes to hours (depending on the amount of data that needs to be uploaded.)

If you need to minimize down time, it is possible to run two copies of you service in Azure. You set each copy with a different geographic affinity in the portal. This would force Azure to run each copy in a different datacenter. Perhaps the first copy is running in the Chicago datacenter, and the second in the Southwest datacenter. You could then use a DNS server that is geo aware, and have it route users to each system, based on their location. There is the matter of replicating your data across the two systems. One way would be to run them in complete separation, if your business processes can handle that. Then, once a night, you could merge the data sets with a background operation.

In any of these cases you need to be able to understand the health of your system. One way to do this is to use the management APIs to understand the load on your system, and report back the status in a recurring manner.

18.7 Summary

Although this chapter might have seemed like two disparate topics jammed together in one chapter to save space, at this point, you should understand how diagnostics, service management, and good service health instrumentation is important to running a healthy service in the cloud.

We dug deep into the diagnostics API, showing how you collect all sorts of data from many different instances, and merge it all into one place to make it easy to analyze the result. Each data source has its own configuration mechanism, and destination. You can tell each data source to transfer its data on a schedule, or you can force an on demand transfer.

The Azure platform also provides a rich service management API over REST. You can do almost anything over this API that you can do in the portal except for creating hosting and storage services. While REST is awesome, many people will likely opt to use csmanage.exe, especially when they are automating deployment with a build system.

Finally, we merged these two concepts together to discuss how an auto scaling feature might look, and why Microsoft doesn't provide one out of the box. Auto scaling is fraught with peril and you have to be very careful with the logic you use in the system controller.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=579