INTRODUCING WINDOWS AZURE

DAVID CHAPPELL

OCTOBER 2010

SPONSORED BY MICROSOFT CORPORATION

CONTENTS

An Overview of Windows Azure **2**
Compute 4
Storage 5
Fabric Controller 7
Content Delivery Network 8
Connect 9
Using Windows Azure: Scenarios **11**
Creating a Scalable Web Application 11
Creating a Parallel Processing Application 12
Creating a Scalable Web Application with Background Processing 13
Creating a Web Application with Relational Data 14
Migrating an On-Premises Web Application with Relational Data 15
Using Cloud Storage from an On-Premises or Hosted Application 16
Understanding Windows Azure: A Closer Look **17**
Creating Windows Azure Applications 17
Examining the Compute Service 18
Examining the Storage Service 19
Blobs 19
Tables 20
Queues 21
Examining the Fabric Controller 23
Looking Ahead **24**
Conclusions **24**
For Further Reading **25**
About the Author **25**

AN OVERVIEW OF WINDOWS AZURE

Cloud computing is here. Running applications and storing data on machines in an Internet-accessible data center can offer plenty of advantages. Yet wherever they run, applications are built on some kind of platform. For on-premises applications, such as those running inside an organization's data center, this platform usually includes an operating system, some way to store data, and perhaps more. Applications running in the cloud need a similar foundation.

The goal of Windows Azure is to provide this. Part of the larger Windows Azure platform, Windows Azure is a foundation for running applications and storing data in the cloud. Figure 1 illustrates this idea.

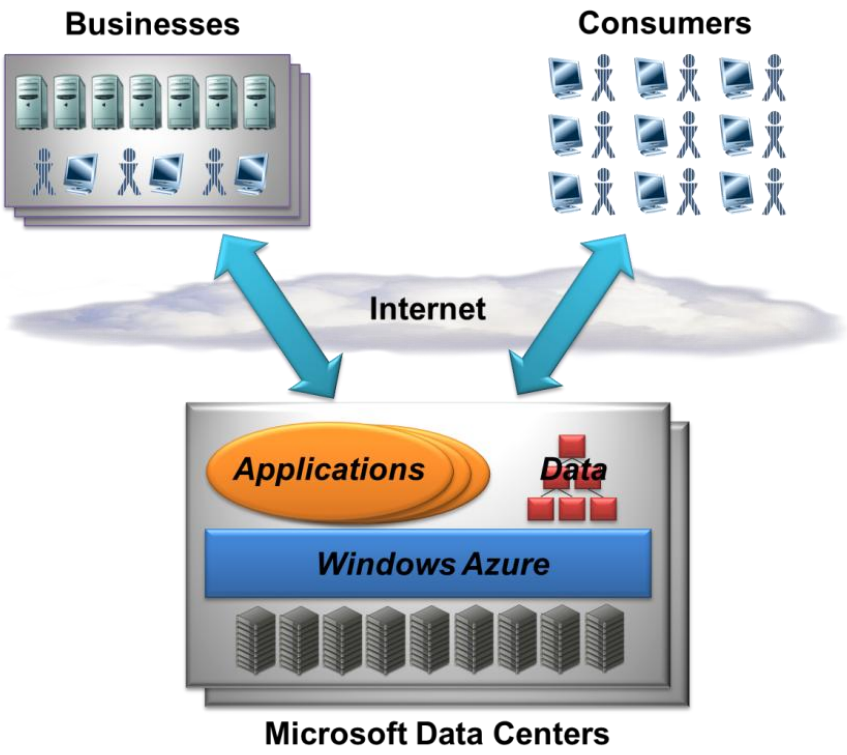

Figure 1: Windows Azure applications run in Microsoft data centers and are accessed via the Internet.

Rather than providing software that Microsoft customers can install and run themselves on their own computers, Windows Azure today is a service: Customers use it to run applications and store data on Internet-accessible machines owned by Microsoft. Those applications might provide services to businesses, to consumers, or both. Here are some examples of the kinds of applications that can be built on Windows Azure:

- An independent software vendor (ISV) could create an application that targets business users, an approach that's often referred to as *Software as a Service (SaaS).* Windows Azure was designed in part to support Microsoft's own SaaS applications, so ISVs can also use it as a foundation for a variety of business-oriented cloud software.

- An ISV might create a SaaS application that targets consumers rather than businesses. Because Windows Azure is intended to support very scalable software, a firm that plans to target a large consumer market might well choose it as a platform for a new application.

- Enterprises might use Windows Azure to build and run applications that are used by their own employees. While this situation probably won't require the enormous scale of a consumer-facing application, the reliability and manageability that Windows Azure offers could still make it an attractive choice.

To support cloud applications and data, Windows Azure has five components, as Figure 2 shows.

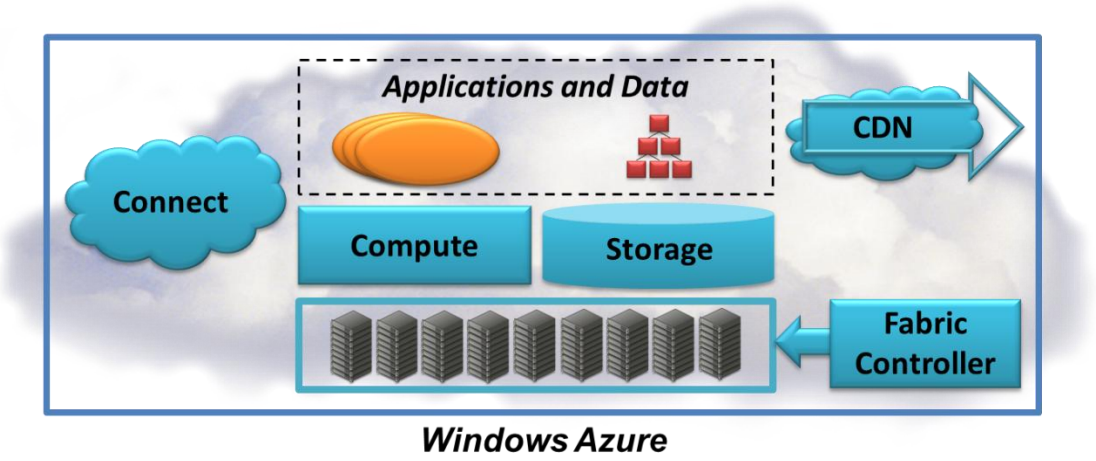

Figure 2: Windows Azure has five main parts: Compute, Storage, the Fabric Controller, the CDN, and Connect.

Those components are:

- Compute: runs applications in the cloud. Those applications largely see a Windows Server environment, although the Windows Azure programming model isn't exactly the same as the on-premises Windows Server model.

- Storage: stores binary and structured data in the cloud.

- Fabric Controller: deploys, manages, and monitors applications. The fabric controller also handles updates to system software throughout the platform.

- Content Delivery Network (CDN): speeds up global access to binary data in Windows Azure storage by maintaining cached copies of that data around the world.

- Connect: allows creating IP-level connections between on-premises computers and Windows Azure applications.

The rest of this section introduces each of these technologies.

COMPUTE

Windows Azure compute can run many different kinds of applications. Whatever an application does, however, it must be implemented as one or more *roles*. Windows Azure then typically runs multiple *instances* of each role, using built-in load balancing to spread requests across them. Figure 3 shows how this looks.

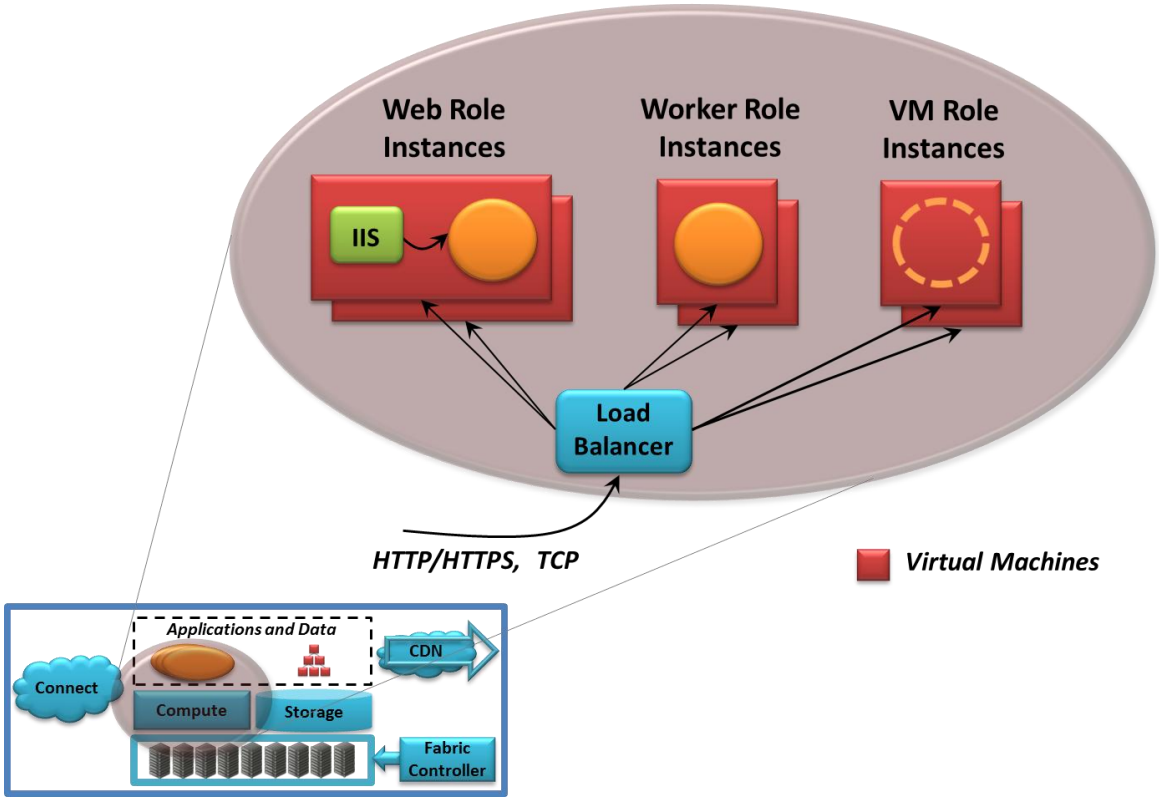

Figure 3: A running Windows Azure application consists of any combination of Web role instances, Worker role instances, and VM role instances.

In the current version of Windows Azure, developers can choose from three kinds of roles:

- Web roles, intended primarily to make it easier to create Web-based applications. Each Web role instance has Internet Information Services (IIS) 7 pre-configured inside it, so creating applications using ASP.NET, Windows Communication Foundation (WCF), or other Web technologies is straightforward. Developers can also create applications in native code—using the .NET Framework isn't required. This means that they can install and run non-Microsoft technologies as well, including PHP and Java.

- Worker roles, designed to run a variety of Windows-based code. The biggest difference between a Web role and a Worker role is that Worker roles don't have IIS configured inside them, and so the code they run isn't hosted by IIS. A Worker role might run a simulation, for example, or handle video processing or do nearly anything else. It's common for an application to interact with users through a Web role, then hand tasks off to a Worker role for processing. Once again, a developer is free to use the .NET Framework or other software that runs on Windows, including non-Microsoft technologies.

- VM roles, each of which runs a user-provided Windows Server 2008 R2 image. Among other things, a VM role can sometimes be useful in moving an on-premises Windows Server application to Windows Azure.

To submit an application to Windows Azure, a developer can use the Windows Azure portal. Along with the application, she submits configuration information that tells the platform how many instances of each role to run. The Windows Azure fabric controller then creates a virtual machine (VM) for each instance, running the code for the appropriate role in that VM.

As Figure 3 shows, requests from the application's users can be made using protocols such as HTTP, HTTPS, and TCP. However these requests arrive, they're load balanced across all instances of a role. Because the load balancer doesn't allow creating an affinity with a particular role instance—there's no support for sticky sessions—there's no way to guarantee that multiple requests from the same user will be sent to the same instance of a role. This implies that Windows Azure role instances shouldn't maintain their state themselves between requests. Instead, any client-specific state should be written to Windows Azure storage, stored in SQL Azure (another component of the Windows Azure platform), or maintained externally in some other way.

A developer can use any combination of Web role instances, Worker role instances, and VM role instances to create a Windows Azure application. If the application's load increases, he can use the Windows Azure portal to request more instances for any of the roles in his application. If the load decreases, he can reduce the number of running instances. Windows Azure also exposes an API that lets all of these things be done programmatically—changing the number of running instances doesn't require manual intervention—but the platform doesn't automatically scale applications based on their load.

To create Windows Azure applications, a developer uses the same languages and tools as for any Windows application. She might write a Web role using ASP.NET and Visual Basic, for example, or use WCF and C#. Similarly, she might create a Worker role in one of these .NET languages, work directly in C++ without the .NET Framework, or use Java. And while Windows Azure provides add-ins for Visual Studio, using this development environment isn't required. A developer who has installed PHP, for example, might choose to use another tool to write applications.

To allow monitoring and debugging Windows Azure applications, each instance can call a logging API that writes information to a common application-wide log. A developer can also configure the system to collect performance counters for an application, measure its CPU usage, store crash dumps if it fails, and more. This information is kept in Windows Azure storage, and a developer is free to write code to examine it. For example, if a Worker role instance crashes three times within an hour, custom code might send an email to the application's administrator.

The ability to execute code is a fundamental part of a cloud platform, but it's not enough. Applications also need persistent storage. Meeting this need is the goal of the Windows Azure storage service, described next.

STORAGE

Applications work with data in many different ways. Accordingly, the Windows Azure storage service provides several options. Figure 4 shows the choices.

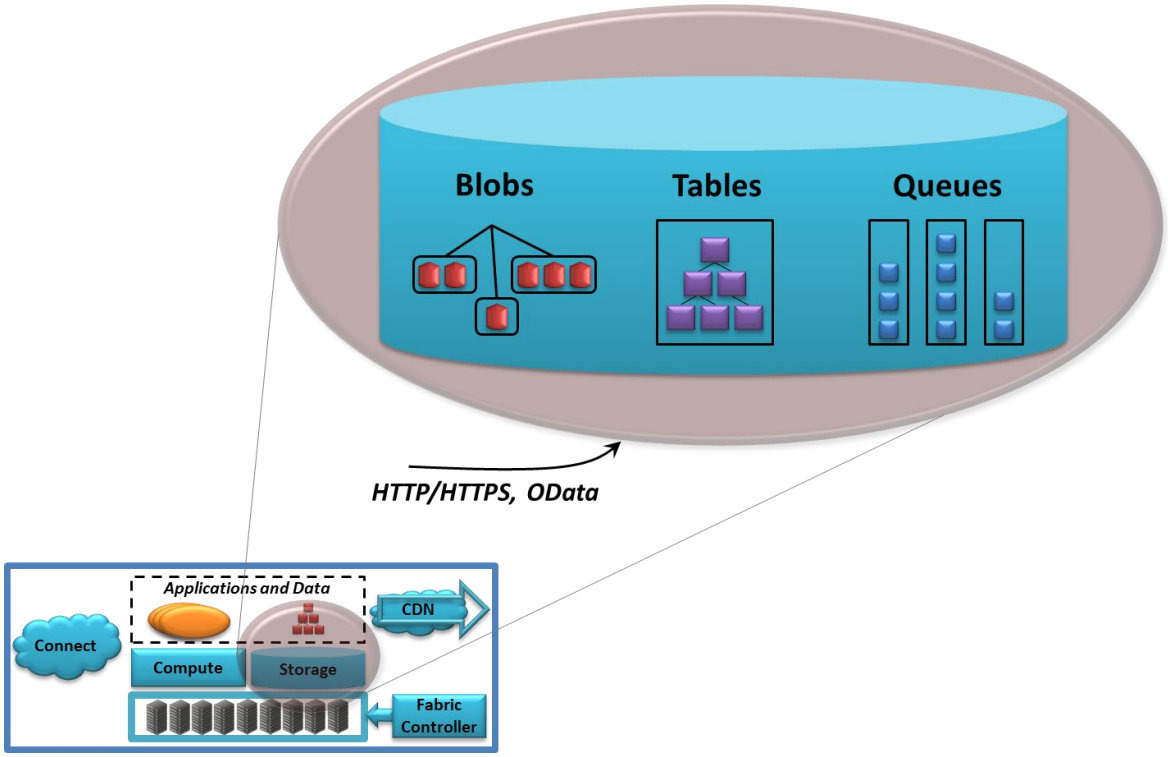

Figure 4: Windows Azure Storage provides blobs, tables, and queues.

The simplest way to store data in Windows Azure storage is to use blobs. A blob contains binary data, and as Figure 4 suggests, there's a simple hierarchy: Each *container* can contain one or more blobs. Blobs can be big—up to a terabyte—and they can also have associated metadata, such as information about where a JPEG photograph was taken or who the singer is for an MP3 file. Blobs also provide the underlying storage for *Windows Azure drives*, a mechanism that lets a Windows Azure role instance interact with persistent storage as if it were a local NTFS file system.

Blobs are just right for some situations, but they're too unstructured for others. To let applications work with data in a more fine-grained way, Windows Azure storage provides tables. Don't be misled by the name: These aren't relational tables. The data each one holds is actually stored in a group of *entities* that contain *properties*. And rather than using SQL, an application can query a table's data using the conventions defined by OData. This approach allows scale-out storage—scaling by spreading data across many machines—more effectively than would a standard relational database. In fact, a single Windows Azure table can contain billions of entities holding terabytes of data.

Blobs and tables are both focused on storing and accessing data. The third option in Windows Azure storage, queues, has a quite different purpose. A primary function of queues is to provide a way for Web role instances to communicate asynchronously with Worker role instances. For example, a user might submit a request to perform some compute-intensive task via a Web interface implemented by a Windows Azure Web role. The Web role instance that receives this request can write a message into a queue describing the work to be done. A Worker role instance that's waiting on this queue can then read the message and carry out the task it specifies. Any results can be returned via another queue or handled in some other way.

Regardless of how data is stored—in blobs, tables, or queues—all information held in Windows Azure storage is replicated three times. This replication allows fault tolerance, since losing a copy isn't fatal. The system provides strong consistency, however, so an application that immediately reads data it has just written is guaranteed to get back what it wrote. Windows Azure also keeps a backup copy of all data in another data center in the same part of the world. If the data center holding the main copy is unavailable or destroyed, this backup remains accessible.

Windows Azure storage can be accessed by a Windows Azure application, by an on-premises application, or by an application running at a hoster or on another cloud platform. In all of these cases, all three Windows Azure storage styles use the conventions of REST to identify and expose data, as Figure 4 suggests. Blobs, tables, and queues are all named using URIs and accessed via standard HTTP operations. A .NET client can use a Windows Azure-provided library to do this, but it's not required—an application can also make raw HTTP calls.

Creating Windows Azure applications that use blobs, tables, and queues can certainly be useful. Applications that rely on relational storage can instead use SQL Azure, another component of the Windows Azure platform. Applications running on Windows Azure (or in other places) can use this technology to get familiar SQL-based access to relational storage in the cloud.

FABRIC CONTROLLER

All Windows Azure applications and all of the data in Windows Azure storage reside in some Microsoft data center. Within that data center, the set of machines dedicated to Windows Azure and the software that runs on them are managed by the fabric controller. Figure 5 illustrates this idea.

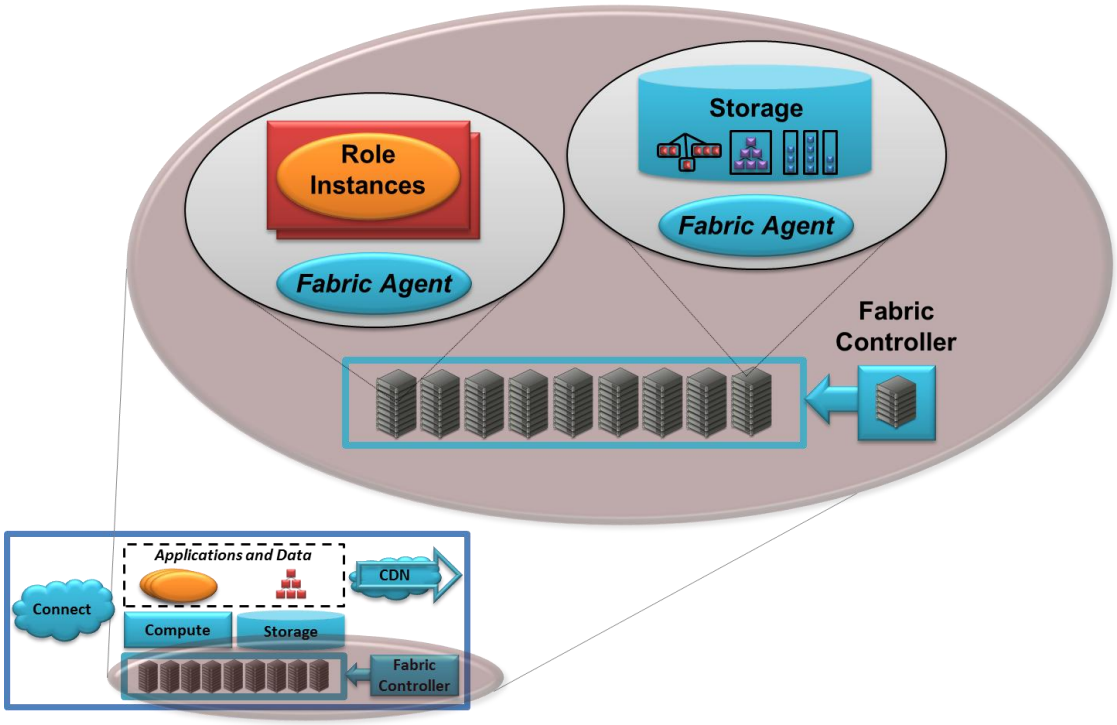

Figure 5: The fabric controller interacts with Windows Azure applications via a fabric agent.

The fabric controller is itself a distributed application that's replicated across a group of machines. It owns all of the resources in its environment: computers, switches, load balancers, and more. Because it can communicate with a *fabric agent* on every computer, it's also aware of every Windows Azure application in this fabric. (Interestingly, the fabric controller sees Windows Azure storage as just another application, and so the details of data management and replication aren't visible to the controller.)

This broad knowledge lets the fabric controller do a number of useful things. It monitors all running applications, for example, giving it an up-to-the-minute picture of what's happening. It also decides where new applications should run, choosing physical servers to optimize hardware utilization. To do this, the fabric controller depends on the configuration information that's uploaded with each Windows Azure application. This file provides an XML-based description of what the application needs: how many Web role instances, how many Worker role instances, and more. When the fabric controller deploys a new application, it uses this configuration file to determine how many VMs to create.

Once it's created those VMs, the fabric controller then monitors each of them. If an application requires five Web role instances and one of them dies, for example, the fabric controller will automatically start a new one. Similarly, if the machine a VM is running on dies, the fabric controller will start a new instance of the role on another machine, resetting the load balancer as necessary to point to this new VM.

Windows Azure today gives developers five VM sizes to choose from. The options are:

- Extra-small, with a single-core 1.0 GHz CPU, 768MB of memory, and 20GB of instance storage.
- Small, with a single-core 1.6 GHz CPU, 1.75 GB of memory, and 225 GB of instance storage.
- Medium, with a dual-core 1.6 GHz CPU, 3.5 GB of memory, and 490 GB of instance storage.
- Large, with a four-core 1.6 GHz CPU, 7 GB of memory, and 1,000 GB of instance storage.
- Extra-large, with an eight-core 1.6 GHz CPU, 14 GB of memory, and 2,040 GB of instance storage.

An extra-small instance shares a processor core with other extra-small instances. For all of the other sizes, however, each instance has one or more dedicated cores. This means that application performance is predictable, with no arbitrary limit on how long an instance can execute. A Web role instance, for example, can take as long as it needs to handle a request from a user, or a Worker role instance might compute the value of pi to a million digits.

For Web and Worker roles (although not for VM roles), the fabric controller also manages the operating system in each instance. This includes things such as applying OS patches and updating other system software. This lets developers focus solely on creating applications—they don't need to worry about managing the platform itself. It's important to understand, however, that the fabric controller always assumes that at least two instances of every role are running. This lets it shut down one of them to update its software without taking down the entire application. For this and other reasons, running a single instance of any Windows Azure role is usually a bad idea.

CONTENT DELIVERY NETWORK

One common use of blobs is to store information that will be accessed from many different places. Think of an application that serves up videos, for example, to Flash, Silverlight, or HTML 5 clients around the

world. To improve performance in situations like this, Windows Azure provides a content delivery network. The CDN stores copies of a blob at sites closer to the clients that use it. Figure 6 illustrates this idea.

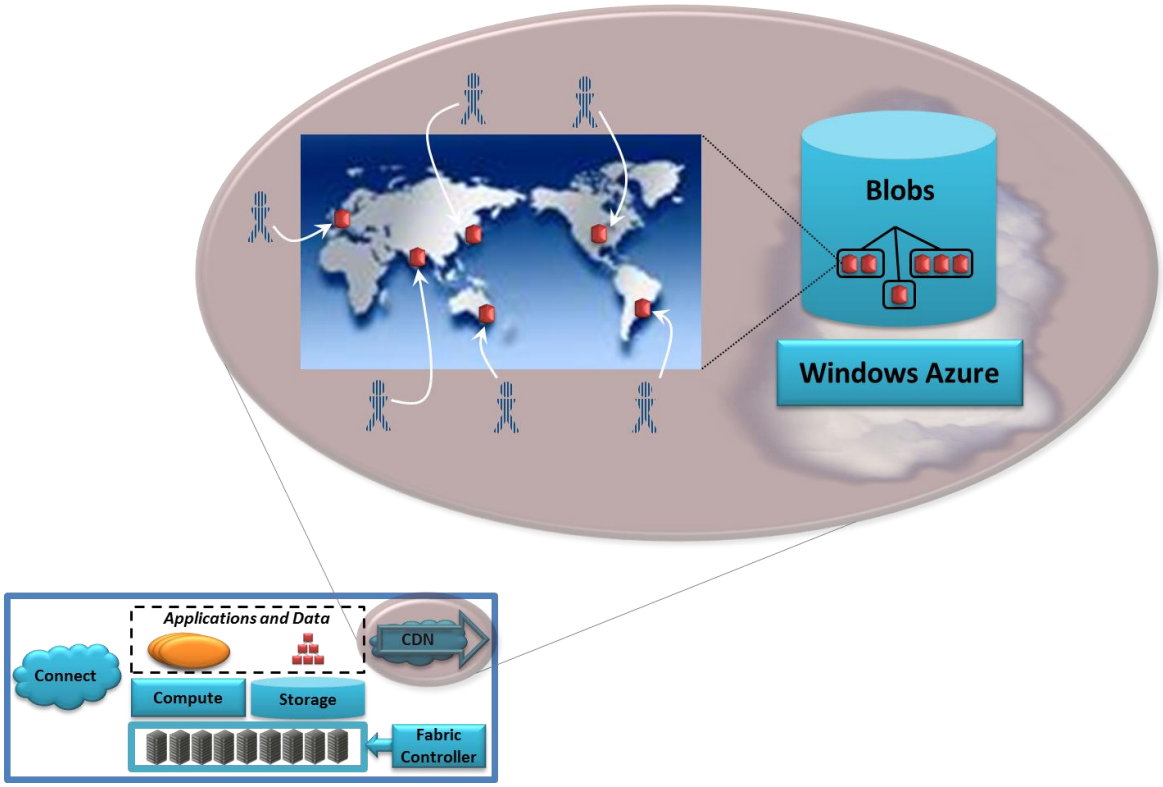

Figure 6: The Windows Azure CDN caches copies of blobs around the world, letting users access that information more quickly.

Don't take the figure too literally—the Windows Azure CDN actually has many more global caching locations than it shows—but the concept is correct. The first time a particular blob is accessed by a user, the CDN stores a copy of that blob at a location that's geographically close to that user. The next time this blob is accessed, its contents will be delivered from the cache rather than from the more remote original.

For example, suppose Windows Azure is used to provide videos of a day's sporting events to a far-flung audience. The first user who accesses a particular video won't get the benefit of the CDN, since that blob isn't yet cached in a closer location. All other users in the same geography will see better performance, however, since using the cached copy lets the video load more quickly.

CONNECT

Running applications in the Microsoft cloud is useful. But the applications and data we use inside our organizations aren't going away any time soon. Given this, effectively connecting on-premises environments with Windows Azure is important.

Windows Azure Connect is designed to help do this. By providing IP-level connectivity between a Windows Azure application and machines running outside the Microsoft cloud, it can make this combination easier to use. Figure 7 illustrates the idea.

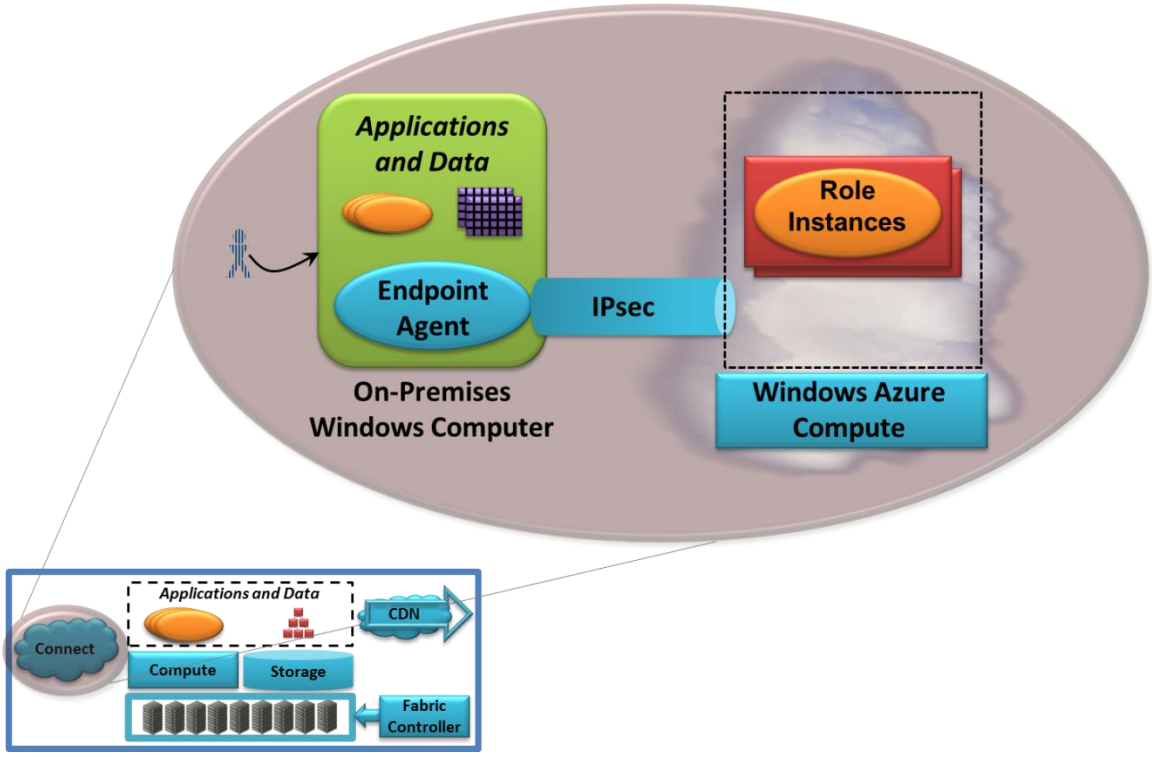

Figure 7: Windows Azure Connect allows IP-level communication between an on-premises Windows machine and a Windows Azure application.

As the figure shows, using Windows Azure Connect requires installing an *endpoint agent* on each on-premises computer that's connecting to a Windows Azure application. (Because the technology relies on IP v6, the endpoint agent is available today only for Windows Server 2008, Windows Server 2008 R2, Windows Vista, and Windows 7.) The Windows Azure application also needs to be configured to work with Windows Azure Connect. Once this is done, the agent can use IPsec to interact with a particular role in that application.

Note that this isn't a full-fledged virtual private network (VPN). While Microsoft has announced plans to offer this eventually, Windows Azure Connect is a simpler solution. Setting it up doesn't require contacting your network administrator, for instance. All that's required is the ability to install the endpoint agent on the local machine. This approach also hides the potential complexity of configuring IPsec—it's done for you by Windows Azure Connect.

Once the technology is set up, roles in a Windows Azure application appear to be on the same IP network as the on-premises machine. This allows things such as the following:

- A Windows Azure application can access an on-premises database directly. For example, suppose an organization moves an existing Windows Server application built using ASP.NET into a Windows Azure Web role. If the database this application uses needs to stay on premises, a Windows Azure Connect

connection can let the application—now running on Windows Azure—access the on-premises database just as it always has. Not even the connection string needs to be changed.

- A Windows Azure application can be domain-joined to the on-premises environment. Doing this allows single sign-on to the cloud application by on-premises users. It also lets the application use existing Active Directory accounts and groups for access control.

Making the cloud fit well with today's on-premises environment is important. By allowing direct IP-level connectivity, Windows Azure Connect makes this easier for Windows Azure applications.

USING WINDOWS AZURE: SCENARIOS

Understanding the components of Windows Azure is important, but it's not enough. The best way to get a feeling for what this platform can do is to walk through examples of how you might use it. Accordingly, this section looks at several Windows Azure scenarios: creating a scalable Web application, creating a parallel processing application, creating a Web application with background processing, creating a Web application with relational data, migrating an on-premises Web application with relational data, and using cloud storage from an on-premises or hosted application.

CREATING A SCALABLE WEB APPLICATION

Suppose an organization wishes to create an Internet-accessible Web application. The usual choice today is to run that application in a data center within the organization or at a hoster. In many cases, however, a cloud platform such as Windows Azure is a better choice. For example, if the application needs to handle a large number of simultaneous users, building it on a platform expressly designed to support this makes sense. The intrinsic support for scalable applications and data that Windows Azure provides can handle much larger loads than more conventional Web technologies.

Or suppose the application's load will vary significantly, with occasional spikes in the midst of long periods of lower usage. An online ticketing site might display this pattern, for example, as might a news video site with occasional hot stories, an application that's used mostly at certain times of day, and others. Running this kind of application in a conventional data center requires always having enough machines on hand to handle the peaks, even though most of those systems go unused most of the time. If the application is instead built on Windows Azure, the organization running it can expand the number of instances it's using only when needed, then shrink back to a smaller number. Since Windows Azure charging is usage-based—you pay per hour for each instance--this is likely to be cheaper than maintaining lots of mostly unused machines.

To create a highly scalable Web application on Windows Azure, a developer might use Web roles and tables. Figure 8 shows a simple illustration of how this looks.

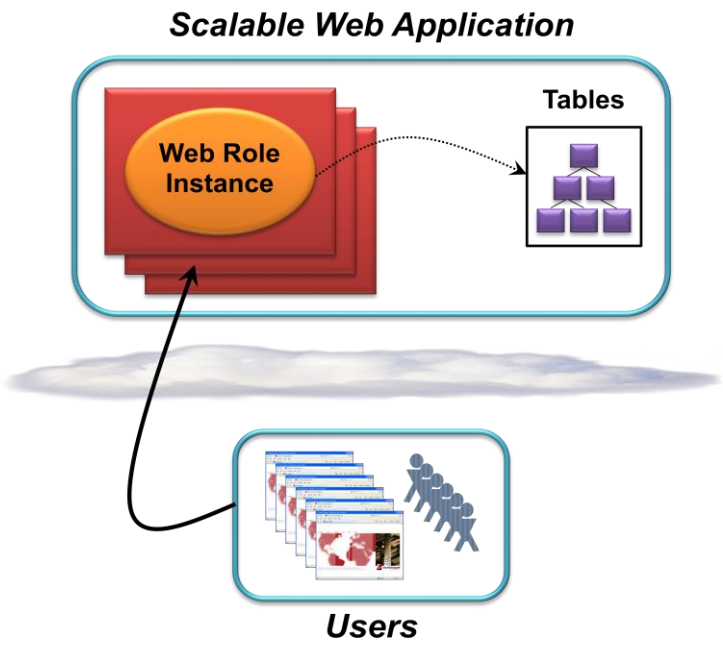

Figure 8: A scalable Web application can use Web role instances and tables.

In the example shown here, the clients are browsers, and so the application logic can be implemented using ASP.NET or another Web technology. It's also possible to create a scalable Web application that exposes RESTful and/or SOAP-based Web services using WCF, then call those services from, say, a Silverlight client. In either case, the developer specifies how many instances of the Web role should run, and the Windows Azure fabric controller creates this number of VMs. As described earlier, the fabric controller also monitors these instances, making sure that the requested number is always available. For data storage, the application uses Windows Azure storage tables, which provide scale-out storage capable of handling very large amounts of data.

CREATING A PARALLEL PROCESSING APPLICATION

Scalable Web applications are useful, but they're not the only situation where Windows Azure makes sense. Think about an organization that occasionally needs lots of computing power for a parallel processing application. There are plenty of examples of this: financial modeling at a bank, rendering at a film special effects house, new drug development in a pharmaceutical company, and more. While it's possible to maintain a large cluster of machines to meet this occasional need, it's also expensive. Windows Azure can instead provide these resources as needed, offering an on-demand compute cluster.

A developer can use Worker roles to create this kind of application. And while it's not the only choice, parallel applications commonly use large datasets, which might be stored in Windows Azure blobs. Figure 9 illustrates this kind of application.

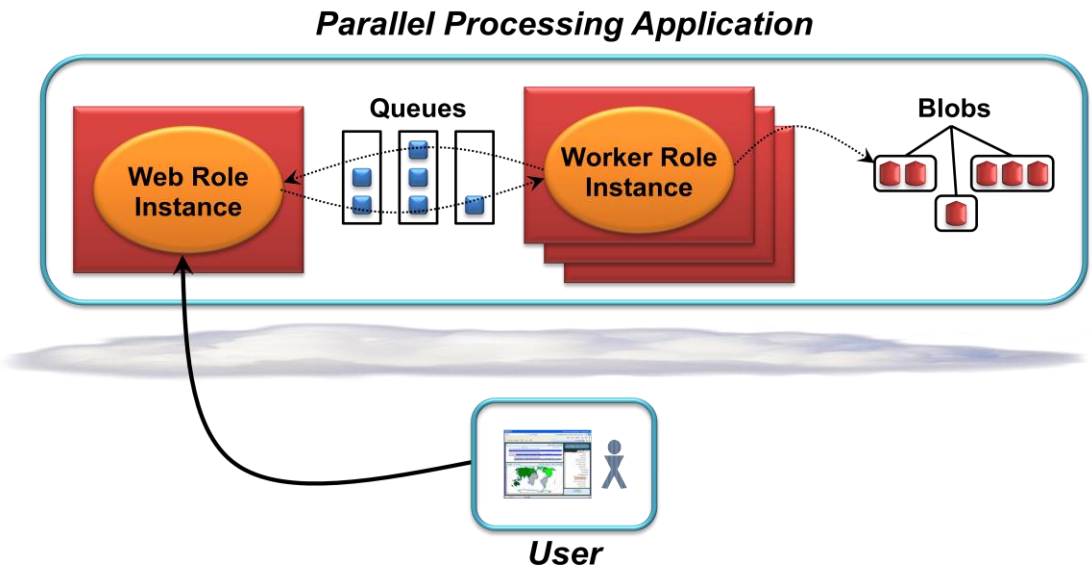

Figure 9: A parallel processing application might use a Web role instance, many Worker role instances, queues, and blobs.

In the scenario shown here, the parallel work is done by a number of Worker role instances running simultaneously, each using blob data. Since Windows Azure imposes no limit on how long an instance can run, each one can perform an arbitrary amount of work. To interact with the application, the user relies on a single Web role instance. Through this interface, the user might determine how many Worker instances should run, start and stop those instances, access results, and more. Communication between the Web role instance and the Worker role instances relies on Windows Azure storage queues.

Given the massive amount of processing power available in the cloud, this new approach is likely to transform high-performance computing. For example, Microsoft Windows HPC Server now allows creating a compute cluster using Windows Azure worker role instances along with or instead of on-premises physical servers. However it's done, exploiting this new source of computing power makes sense in quite a few situations.

CREATING A SCALABLE WEB APPLICATION WITH BACKGROUND PROCESSING

It's probably fair to say that a majority of applications built today provide a browser interface. Yet while applications that do nothing but accept and respond to browser requests are useful, they're also limiting. There are lots of situations where Web-accessible software needs to initiate work that runs in the background, independently from the request/response part of the application.

For example, think about a Web application for video sharing. It needs to accept browser requests, perhaps from a large number of simultaneous users. Some of those requests will upload new videos, each of which must be processed and stored for later access. Making the user wait while this processing is done wouldn't make sense. Instead, the part of the application that accepts browser requests should be able to initiate a background task that carries out this work.

Web roles and Worker roles can be used together to address this scenario. Figure 10 shows how this kind of application might look.

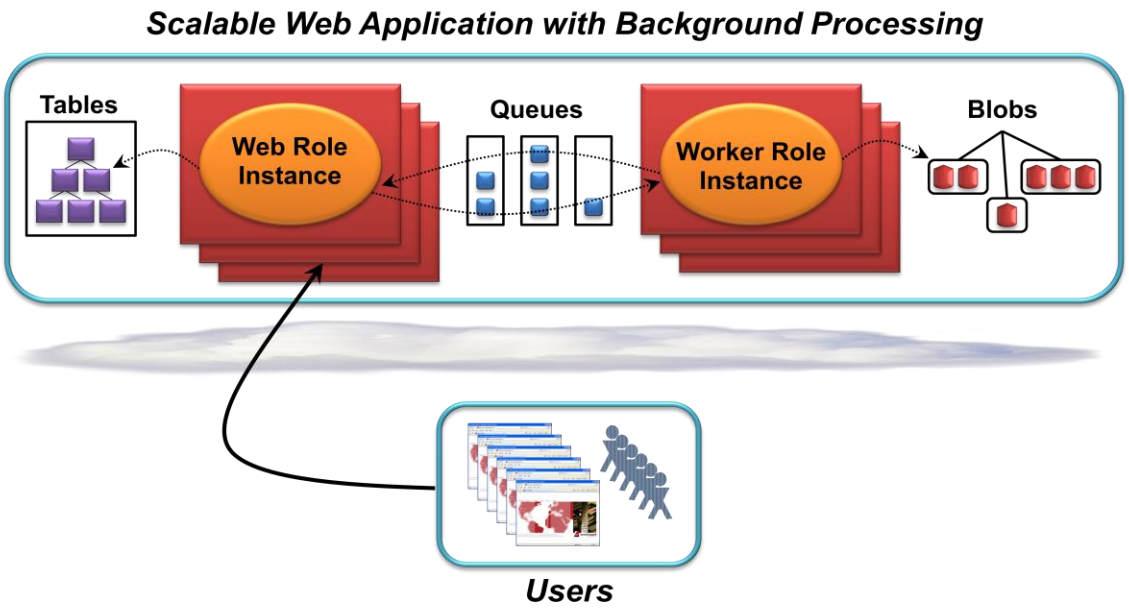

Figure 10: A scalable Web application with background processing might use many Windows Azure capabilities.

Like the scalable Web application shown earlier, this application uses some number of Web role instances to handle user requests. To support a large number of simultaneous users, it uses tables to store their profile information. For background processing, it relies on Worker role instances, passing them tasks via queues. In this example, those Worker role instances work on blob data, but other approaches are also possible.

This example shows how an application might combine many of the basic capabilities that Windows Azure exposes: Web role instances, Worker role instances, blobs, tables, and queues. And although it's not shown in the figure, a video-sharing application might also use the Windows Azure CDN to speed up access. While not every application needs all of these features, having them all available is essential for supporting more complex scenarios like this one.

CREATING A WEB APPLICATION WITH RELATIONAL DATA

Windows Azure blobs and tables are right for some situations. For many others, though, relational data is better. Suppose an enterprise wants to build and run an application for its own employees on Windows Azure, for instance. Maybe the application will have a short or unpredictable lifetime, and so allocating a server in the enterprise's data center isn't worthwhile. Or maybe the application needs to be up and running as quickly as possible, making the wait for internal IT to provision a server unacceptable. Or perhaps the organization believes that running the application on Windows Azure will be cheaper and simpler.

Whatever the reason, this application probably doesn't need the massive scale that Windows Azure tables allow. Instead, its creators might prefer to use the relational approach they already know, complete with familiar reporting tools. In a case like this, the application might use Windows Azure together with SQL Azure, as Figure 11 shows.

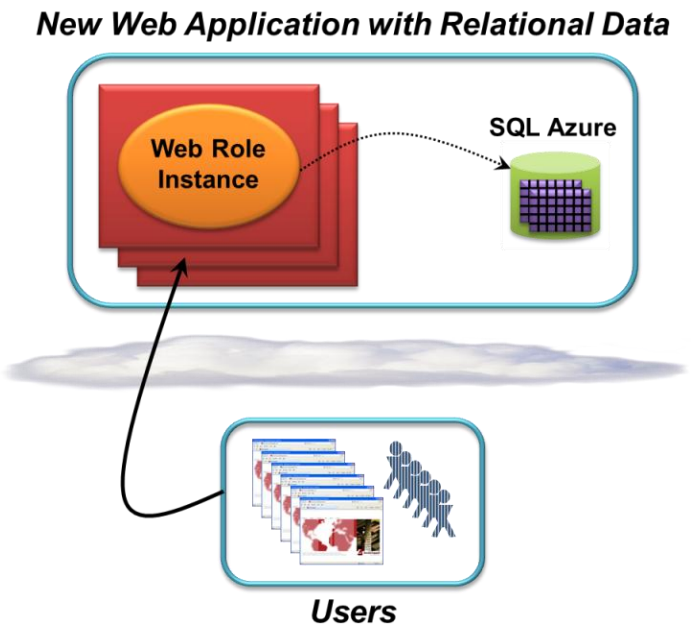

Figure 11: A Windows Azure application can use SQL Azure to work with relational data.

SQL Azure provides a large subset of SQL Server's functionality, including reporting, as a managed cloud service. Applications can create databases, run SQL queries, and more, but there's no need to administer the database system or the hardware it runs on—Microsoft takes care of this. A SQL Azure database can be accessed using the Tabular Data Stream (TDS) protocol, just as in the on-premises version of SQL Server. This lets a Windows Azure application access relational data using familiar mechanisms like Entity Framework and ADO.NET. And since SQL Azure is a cloud service, charging is usage-based.

Because Windows Azure and SQL Azure provide cloud analogs to their on-premises counterparts, it can be straightforward to move the code and data for this kind of application between the two worlds. Things aren't exactly the same—the Windows Azure application must be able to run multiple instances, for example—but the cloud and on-premises environment are quite similar. This portability is useful whenever it makes sense to create an application whose code and data can exist either on-premises or in the cloud.

MIGRATING AN ON-PREMISES WEB APPLICATION WITH RELATIONAL DATA

Suppose that rather than creating a new Web application for Windows Azure, an organization wishes to move an existing Windows Server application onto this cloud platform. One way to approach this is by using the Windows Azure VM role. The picture looks much like the previous case, as Figure 12 shows.

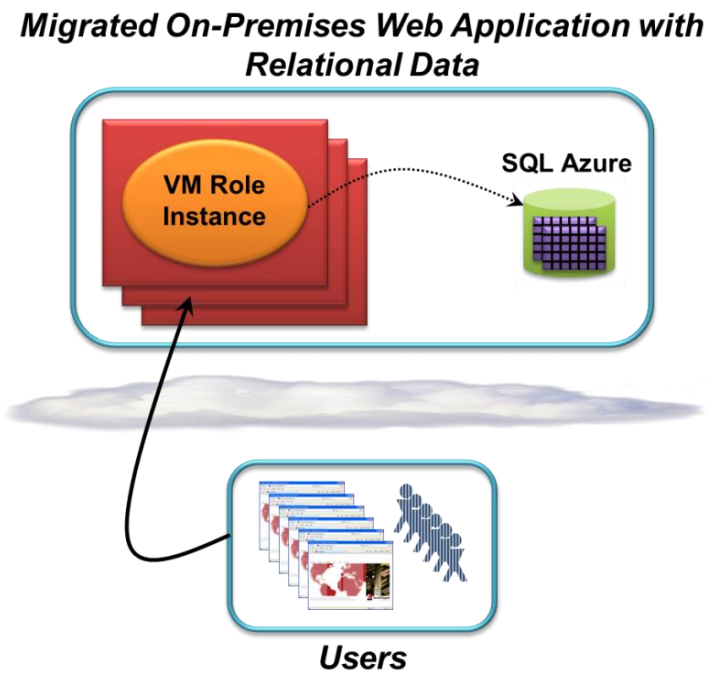

Figure 12: Some on-premises applications can be migrated to Windows Azure using the VM role and SQL Azure.

To use a VM role, an organization creates a virtual hard disk (VHD) from an on-premises machine running Windows Server 2008 R2. This image can then be uploaded to Windows Azure and executed within a VM role. As shown in the figure, the application might access relational data in SQL Azure. Another option is to leave its data on premises, accessing it directly via Windows Azure Connect as described earlier.

VM role can be useful. It's important to understand, however, that moving a Windows Server application to Windows Azure may well require more than just packaging it into a VHD and running it in a VM role. First, recall that the Windows Azure fabric controller assumes that at least two instances of every role are always running. (Qualifying for the Windows Azure Service Level Agreement—the SLA—requires this, in fact.) Remember too that Windows Azure load balances all user requests across a role's instances. If the application being migrated is built to work in this way—maybe it's already running in a load-balanced Web farm, for example—it can run well on Windows Azure without significant changes. If the application expects to run only a single instance, however, it will probably require some redesign to run successfully on Windows Azure.

USING CLOUD STORAGE FROM AN ON-PREMISES OR HOSTED APPLICATION

While Windows Azure provides a range of capabilities, an application sometimes needs only one of them. For example, think about an on-premises or hosted application that needs to store a significant amount of data. An enterprise might wish to archive old email, for example, saving money on storage while still keeping the mail accessible. A news Web site running at a hoster might need a scalable, globally accessible place to store large amounts of text, graphics, video, and profile information about its users. A photo sharing site might want to offload the challenges of storing its information onto a reliable third party. All of these situations can be addressed by Windows Azure storage, as Figure 13 shows.

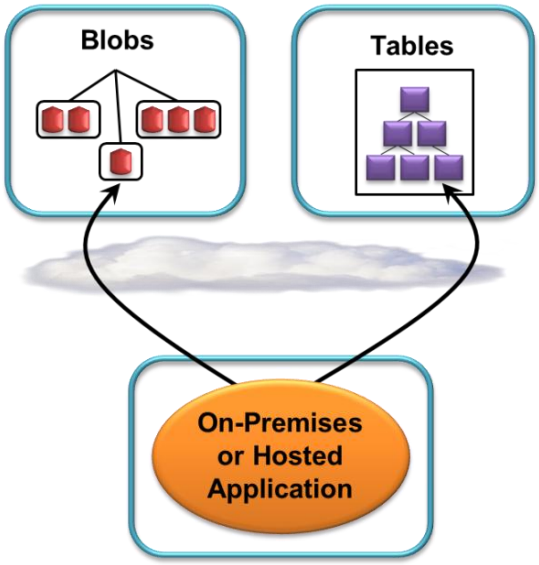

Figure 13: An on-premises or hosted application can use Windows Azure blobs or tables to store its data in the cloud.

As mentioned earlier, an on-premises or hosted application can directly access Windows Azure storage. While this access is likely to be slower than working with local storage, it's also likely to be cheaper, more scalable, and more reliable. For some applications, this tradeoff is definitely worth making. And even though it's not shown in the figure, applications can use SQL Azure in this same way.

UNDERSTANDING WINDOWS AZURE: A CLOSER LOOK

Understanding Windows Azure requires knowing the basics of the platform, then seeing typical scenarios in which those basics can be applied. There's much more to this technology, however. This section takes a deeper look at some of its more interesting aspects.

CREATING WINDOWS AZURE APPLICATIONS

For developers, building a Windows Azure application looks much like building a traditional Windows application. Since the platform supports both .NET applications and applications built using unmanaged code, a developer can use whatever best fits the problem. To make life easier, Visual Studio provides project templates for creating Windows Azure applications. It's also possible to directly upload applications from Visual Studio to Windows Azure.

One obvious difference between the cloud and on-premises worlds is that Windows Azure applications don't run locally. This difference has the potential to make development more challenging. To help mitigate this, Microsoft provides the *development fabric*, a version of the Windows Azure environment that runs on a developer's machine.

The development fabric runs on a single desktop or server machine. It emulates the functionality of Windows Azure in the cloud, complete with Web roles, Worker roles, VM roles, and all three Windows Azure storage options. A developer can build a Windows Azure application, deploy it to the development fabric, and run it in much the same way as with the real thing. He can determine how many instances of

each role should run, for example, use queues to communicate between these instances, and do almost everything else that's possible using Windows Azure itself. Once the application has been developed and tested locally, the developer can upload the code and its configuration information, then run it.

However it's created, a Windows Azure application is typically made available in the cloud via a two-step process. A developer first uploads the application to the platform's staging area. When the developer is ready to make the application live, she uses the Windows Azure portal to request that it be put into production. This switch between staging and production can be done with no downtime, which lets a running application be upgraded to a new version without disturbing its users.

The staged application has a DNS name of the form *<GUID>.*cloudapp.net, where *<GUID>* represents a globally unique identifier assigned by Windows Azure. For production, the developer chooses a DNS name in the same domain, such as myazureservice.cloudapp.net. To use a custom domain rather than the Microsoft cloudapp.net domain, the application's owner can create a DNS alias using a standard CNAME.

Once the application is accessible from the outside world, its users are likely to need some way to identify themselves. To do this, Windows Azure lets developers use any HTTP-based authentication mechanism they like. An ASP.NET application might use a membership provider to store its own user ID and password, for example, or it might use some other method, such as the Microsoft Windows Live ID service. Windows Azure applications are also free to use Windows Identity Foundation (WIF) to implement claims-based identity. The choice is entirely up to the application's creator.

EXAMINING THE COMPUTE SERVICE

Like most technologies, Windows Azure compute has evolved since its first release. Originally, for instance, code in Web and Worker roles could only run in user mode. Today, though, both provide an elevated privileges option, allowing applications to run in admin mode. This can be useful with applications that need to install a COM component, for example, something that was problematic in the first Windows Azure release.

Yet every running instance of a Web or Worker role starts from a clean slate: The underlying operating system in its VM is a standard image defined by Windows Azure. This means that any software installations the role performs must be done every time a new instance is created. This isn't a problem for simple installations, such as adding a single COM component. But suppose the instance needs to install a variety of software to carry out its purpose. Doing this every time a new role instance is created is likely to be too slow.

Avoiding this delay is a primary purpose of VM roles. Rather than requiring the installation to happen each time an instance is created, all of the required software can instead be included in a VHD, with that VHD then used to create a VM role instance. Doing this can be significantly faster than using Web or Worker roles with elevated privileges. It can also be the right solution when the installation process requires manual intervention, something Windows Azure doesn't allow.

Another change from the original version of Windows Azure is that the platform now supports access via the Remote Desktop Protocol (RDP). This is useful in debugging, for example, letting a developer directly access a specific instance. Don't expect to use this for virtual desktop infrastructure (VDI), however—Windows Azure (today, at least) isn't designed to support this kind of scenario.

Other important aspects of Windows Azure compute have been available since the technology's first release. For example, Windows Azure lets a developer indicate which data center an application should run in and where its data should be stored. He can also specify that a particular group of applications and data (including data in SQL Azure) should all live in the same data center. Microsoft is initially providing Windows Azure data centers in the United States, Europe, and Asia, with more to follow.

EXAMINING THE STORAGE SERVICE

To use Windows Azure storage, a developer must first create a *storage account*. To control access to the information in this account, Windows Azure gives its creator a secret key. Each request an application makes to information in this storage account—blobs, tables, and queues—carries a signature created with this secret key. In other words, authorization is at the account level (although blobs do have another option, described later). Windows Azure storage doesn't provide access control lists or any other more fine-grained way to control who's allowed to access the data it holds.

Blobs

Binary large objects—blobs—are often just what an application needs. Whether they hold video, audio, archived email messages, or anything else, they let applications store and access data in a very general way. To use blobs, a developer first creates one or more containers in some storage account. Each of these containers can then hold one or more blobs.

To identify a particular blob, an application can supply a URI of the form:

http://*<StorageAccount>*.blob.core.windows.net/*<Container>*/*<BlobName>*

<StorageAccount> is a unique identifier assigned when a new storage account is created, while *<Container>* and *<BlobName>* are the names of a specific container and a blob within that container.

Blobs come in two forms:

- Block blobs, each of which can contain up to 200 gigabytes of data. To make transferring them more efficient, a block blob is subdivided into blocks. If a failure occurs, retransmission can resume with the most recent block rather than sending the entire blob again. Once all of a blob's blocks have been uploaded, the entire blob can be committed at once.
- Page blobs, which can be as large at one terabyte each. A page blob is divided into 512-byte pages, and an application is free to read and write individual pages at random in the blob.

Whatever kind of blob they hold, containers can be marked as private or public. For blobs in a private container, both read and write requests must be signed using the key for the blob's storage account. For blobs in a public container, only write requests must be signed; any application is allowed to read the blob. This can be useful in situations such as making videos, photos, or other unstructured data generally available on the Internet. In fact, the Windows Azure CDN works only with data stored in public blob containers.

Another important aspect of blobs is the role they play in supporting Windows Azure drives. To understand what that role is, realize first that role instances are free to access their local file system. By

default, however, this storage isn't persistent: When the instance is shut down, the VM and its local storage go away. Mounting a Windows Azure drive for the instance, however, can make a page blob look like a local drive, complete with an NTFS file system. Writes to the drive can be written immediately to the underlying blob. When the instance isn't running, this data is stored persistently in the page blob, ready to be mounted again. Among the ways in which drives can be used are the following:

- A developer can upload a VHD containing an NTFS file system, then mount this VHD as a Windows Azure drive. This provides a straightforward way to move file system data between Windows Azure and an on-premises Windows Server system.
- A Windows Azure developer can install and run a MySQL database system in a Windows Azure role instance, using a Windows Azure drive as underlying storage.

Tables

A blob is easy to understand—it's just a slab of bytes—but tables are a bit more complex. Figure 14 illustrates how the parts of a table fit together.

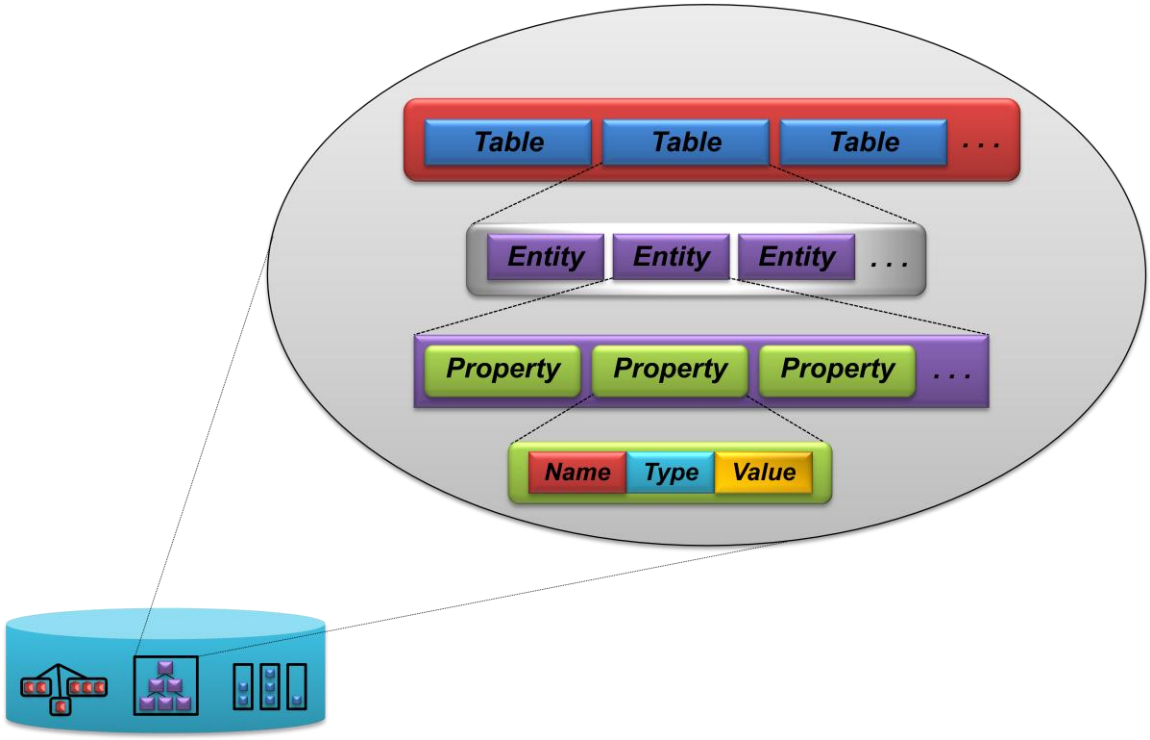

Figure 14: Tables provide entity-based storage.

As the figure shows, each table holds some number of entities. An entity contains zero or more properties, each with a name, a type, and a value. A variety of types are supported, including Binary, Bool, DateTime, Double, GUID, Int, Int64, and String. A property can take on different types at different times depending on the value stored in it, and there's no requirement that all properties in an entity have the same type—a developer is free to do what makes the most sense for his application.

Whatever it contains, an entity can be up to one megabyte in size, and it's always accessed as a unit. Reading an entity returns all of its properties, and writing one can replace all of its properties. It's also possible to update a group of entities within a single table atomically, ensuring that all of the updates succeed or all of them fail.

Windows Azure storage tables are different from relational tables in a number of ways. Most obviously, they're not tables in the usual sense. Also, they can't be accessed using ordinary ADO.NET, nor do they support SQL queries. And tables in Windows Azure storage enforce no schema—the properties in a single entity can be of different types, and those types can change over time. The obvious question is: Why? Why not just support ordinary relational tables with standard SQL queries?

The answer grows out of a primary Windows Azure goal: supporting massively scalable applications. Traditional relational databases can scale up, handling more and more users by running the DBMS on ever-larger machines. But to support truly large numbers of simultaneous users, storage needs to scale out, not up. To allow this, the storage mechanism needs to get simpler: Traditional relational tables with standard SQL don't work well for this purpose. What's needed is the kind of structure provided by Windows Azure tables.

Using tables requires some re-thinking on the part of developers, since familiar relational concepts can't be applied unchanged. Still, for creating very scalable applications, this approach makes sense. It frees developers from worrying about scale—just create new tables, add new entities, and Windows Azure takes care of the rest. It also eliminates much of the work required to maintain a DBMS, since Windows Azure does this for you. The goal is to let developers focus on their application rather than on the mechanics of storing and administering large amounts of data.

Like everything else in Windows Azure storage, tables are accessed RESTfully. A .NET application can use WCF Data Services to do this, hiding the underlying HTTP requests. Any application, .NET or otherwise, is also free to make these requests directly. For example, a query against a particular table is expressed as an HTTP GET against a URI formatted like this:

http://*<StorageAccount>*.table.core.windows.net/*<TableName>*?$filter=*<Query>*

Here, *<TableName>* specifies the table being queried, while *<Query>* contains the query to be executed against this table. If the query returns a large number of results, a developer can get a continuation token that can be passed in on the next query. Doing this repeatedly allows retrieving the complete result set in chunks.

Windows Azure tables aren't the right choice for every storage scenario, and using them requires developers to learn some new things. Still, for applications that need the scalability they provide, tables can be just right.

Queues

While tables and blobs are primarily intended to store and access data, the main goal of queues is to allow communication between different parts of a Windows Azure application. Like everything else in Windows Azure storage, queues are accessed RESTfully. Both Windows Azure applications and external applications reference a queue by using a URI formatted like this:

http://*<StorageAccount>*.queue.core.windows.net/*<QueueName>*

As already described, a common use of queues is to allow interaction between Web role instances and Worker role instances. Figure 15 shows how this looks.

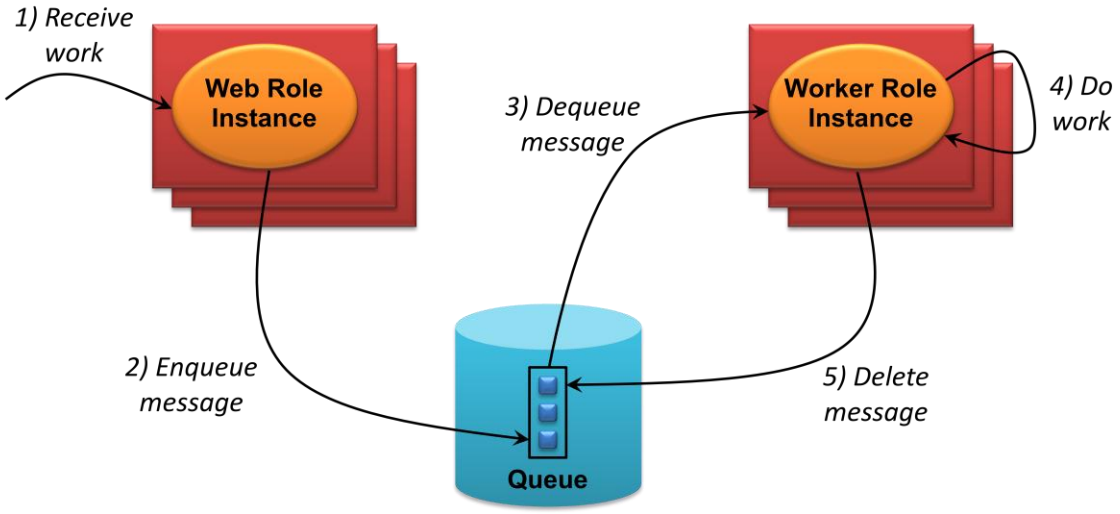

Figure 15: Messages are enqueued, dequeued, processed, then explicitly deleted from the queue.

In a typical scenario, multiple Web role instances are running, each accepting work from users (step 1). To pass that work on to Worker role instances, a Web role instance writes a message into a queue (step 2). This message, which can be up to eight kilobytes, might contain a URI pointing to a blob or to an entity in a table, or something else—it's up to the application. Worker role instances read messages from this queue (step 3), then do the work the message requests (step 4). It's important to note, however, that reading a message from a queue doesn't actually delete the message. Instead, it makes the message invisible to other readers for a set period of time (which by default is 30 seconds). When the Worker role instance has completed the work this message requested, it must explicitly delete the message from the queue (step 5).

Separating Web role instances from Worker role instances makes sense. It frees the user from waiting for a long task to be processed, and it also makes scalability simpler: just add more instances of either. But why make instances explicitly delete messages? The answer is that it allows handling failures. If the Worker role instance that retrieves a message handles it successfully, it will delete the message while that message is still invisible, i.e., within its 30 second window. If a Worker role instance dequeues a message, however, then crashes before it completes the work that message specifies, it won't delete the message from the queue. When its visibility timeout expires, the message will reappear on the queue, then be read by another Worker role instance. The goal is to make sure that each message gets processed at least once.

As this description illustrates, Windows Azure storage queues don't have the same semantics as queues in Microsoft Message Queuing (MSMQ) or other more familiar technologies. For example, a conventional queuing system might offer first in, first out semantics, delivering each message exactly once. Windows Azure storage queues make no such promises. As just described, a message might be delivered multiple

times, and there's no guarantee to deliver messages in any particular order. Life is different in the cloud, and developers will need to adapt to those differences.

EXAMINING THE FABRIC CONTROLLER

To an application developer, compute and storage are the most important parts of Windows Azure. Yet neither one could function without the fabric controller. By knitting together a data center full of machines into a coherent whole, it provides the foundation for everything else.

As described earlier, the fabric controller owns all of the resources in a particular Windows Azure data center. It's also responsible for assigning applications to physical machines. Doing this intelligently is important. For example, suppose a developer requests five Web role instances and four Worker role instances for his application. A naïve assignment might place all of these instances on machines in the same rack serviced by the same network switch. If either the rack or the switch failed, the entire application would no longer be available. Given the high availability goals of Windows Azure, making an application dependent on single points of failure like these would not be a good thing.

To avoid this, the fabric controller groups the machines it owns into a number of *fault domains*. Each fault domain is a part of the data center where a single failure can shut down access to everything in that domain. Figure 16 illustrates this idea.

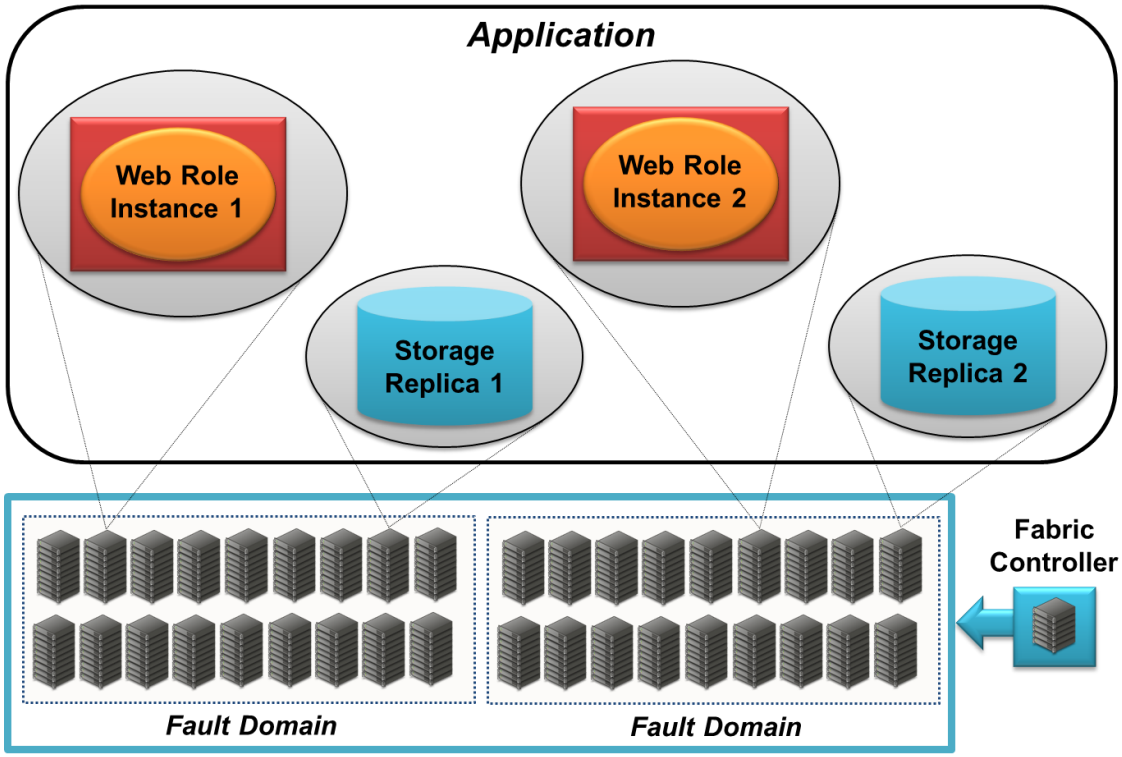

Figure 16: The fabric controller places different instances of an application in different fault domains.

In this simple example, the application is running just two Web role instances, and the data center is divided into two fault domains. When the fabric controller deploys this application, it places one Web role instance in each of the fault domains. This arrangement means that a single hardware failure in the data

center can't take down the entire application. Also, recall that the fabric controller sees Windows Azure storage as just another application—the controller doesn't handle data replication. Instead, the storage application does this itself, making sure that replicas of any blobs, tables, and queues used by this application are placed in different fault domains.

Keeping an application running in the face of hardware failures is useful, but it isn't enough. A truly reliable application—the kind that Windows Azure aims to support—shouldn't need to be shut down to be updated. One way to do this is by using the approach described earlier to switch from the existing version of an application to a new one. Another option is to rely on Windows Azure *update domains*. With this approach, the fabric controller assigns different instances of an application's roles to different update domains. To deploy a new version of the application, the fabric controller deploys new code one update domain at a time: It takes down that domain's instances of a role, updates the code for that role, then starts new instances. The goal is to keep the application running continuously, even while it's being updated. Users might notice the update—the application's response time will likely increase when some of its instances are shut down, for example, and different users will access different versions of the application in the middle of the update. Still, from the user's point of view, the application as a whole remains continuously available.

Don't confuse update domains, a property of an application, with fault domains, a property of the data center. Both have the same overarching purpose, however: helping the fabric controller keep Windows Azure applications running at all times.

LOOKING AHEAD

Microsoft has announced plans to add more to Windows Azure in 2011, including:

- The Windows Azure Platform Appliance, a combination of hardware and software that will allow hosters and enterprises to run Windows Azure in their own data centers.
- CDN dynamic content caching: Today, the Windows Azure CDN works only with blob data. This forthcoming functionality will let the CDN also cache content created dynamically by a Windows Azure application.
- VM role snapshotting: In its first release, the Windows Azure VM role doesn't save any changes made to the OS volume while it's running. Snapshotting will change this, providing a way to periodically save the state of this volume to persistent storage.
- Better Java support**:** While Windows Azure can run Java applications today, Microsoft plans to add more support for doing this. The coming improvements include better Java performance, stronger support for Eclipse-based tools, and a more complete set of Java libraries for Windows Azure.

As always, the goal of these additions is to make this cloud platform useful in a wider range of situations.

CONCLUSIONS

Running applications and storing data in the cloud is the right choice for many situations. The various parts of Windows Azure work together to make this possible. Together with the Windows Azure

development environment, SQL Azure, and the rest of the Windows Azure platform, they provide a bridge for Windows developers moving into this new world.

Today, cloud platforms are still a slightly exotic option for most organizations. As all of us build experience with Windows Azure and other cloud platforms, however, this new approach will begin to feel less strange. Over time, we should expect cloud-based applications—and the cloud platforms they run on—to play an increasingly important role in the software world.

FOR FURTHER READING

- Windows Azure Platform Home Page

  http://www.microsoft.com/windowsazure

- Introducing the Windows Azure Platform, David Chappell

  http://go.microsoft.com/fwlink/?LinkId=158011

- Windows Azure Blobs: Programming Blob Storage

  http://go.microsoft.com/fwlink/?LinkId=153400

- Windows Azure Tables: Programming Table Storage

  http://go.microsoft.com/fwlink/?LinkId=153401

- Windows Azure Queues: Programming Queue Storage

  http://go.microsoft.com/fwlink/?LinkId=153402

ABOUT THE AUTHOR

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technologies.