INTRODUCING WINDOWS AZURE

DAVID CHAPPELL

MARCH 2009

SPONSORED BY MICROSOFT CORPORATION

CONTENTS

An Overview of Windows Azure 2

The Compute Service 3

The Storage Service 5

The Fabric 7

Using Windows Azure: Scenarios 8

Creating a Scalable Web Application 8

Creating a Parallel Processing Application 9

Creating a Scalable Web Application with Background Processing 11

Using Cloud Storage from an On-Premises or Hosted Application 12

Understanding Windows Azure: A Closer Look 13

Developing Windows Azure Applications 13

Examining the Compute Service 15

Examining the Storage Service 15

Blobs 16

Tables 16

Queues 18

Examining the Fabric 20

Conclusions 21

For Further Reading 21

About the Author 21

AN OVERVIEW OF WINDOWS AZURE

Cloud computing is here. Running applications on machines in an Internet-accessible data center can bring plenty of advantages. Yet wherever they run, applications are built on some kind of platform. For on-premises applications, 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 Microsoft's Windows Azure is to provide this. Part of the larger Azure Services Platform, Windows Azure is a platform for running Windows applications and storing data in the cloud. Figure 1 illustrates this idea.

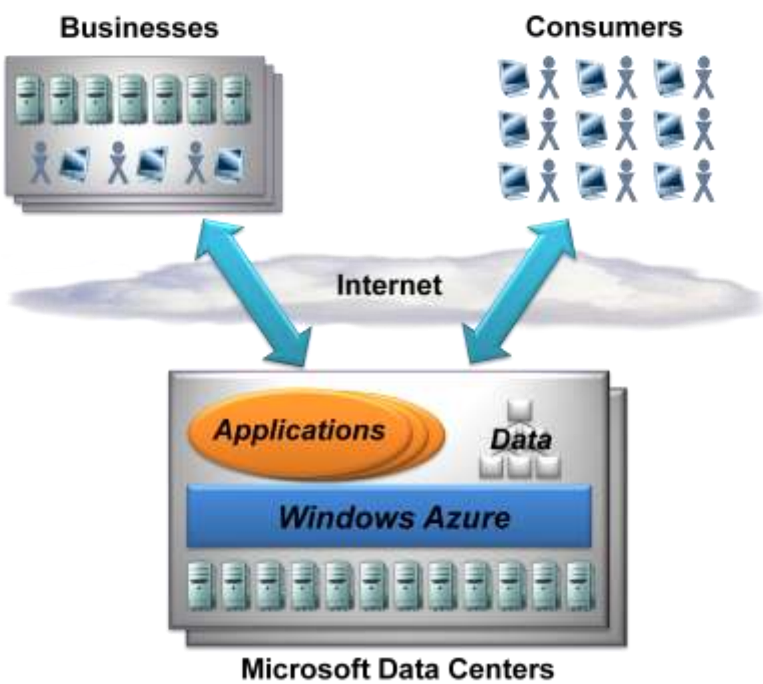

Figure 1: Windows Azure applications run in Microsoft data centers and are accessed via the Internet.

As the figure shows, Windows Azure runs on machines in Microsoft data centers. Rather than providing software that Microsoft customers can install and run themselves on their own computers, Windows Azure 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 might 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).* ISVs can use Windows Azure as a foundation for a variety of business-oriented SaaS applications.

- An ISV might create a SaaS application that targets consumers. Windows Azure is designed to support very scalable software, and so a firm that plans to target a large consumer market might well choose it as a foundation 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.

Whatever a Windows Azure application does, the platform itself provides the same fundamental components, as Figure 2 shows.

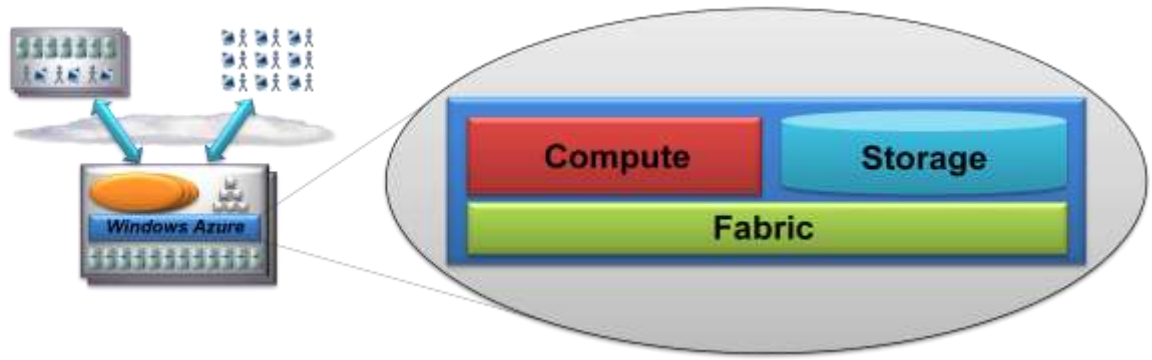

Figure 2: Windows Azure has three main parts: the Compute service, the Storage service, and the Fabric.

As their names suggest, the Compute service runs applications while the Storage service stores data. The third component, the Windows Azure Fabric, provides a common way to manage and monitor applications that use this cloud platform. The rest of this section introduces each of these three parts.

THE COMPUTE SERVICE

The Windows Azure Compute service can run many different kinds of applications. A primary goal of this platform, however, is to support applications that have a very large number of simultaneous users. (In fact, Microsoft has said that it will build its own SaaS applications on Windows Azure, which sets the bar high.) Reaching this goal by scaling *up*—running on bigger and bigger machines—isn't possible. Instead, Windows Azure is designed to support applications that scale *out*, running multiple copies of the same code across many commodity servers.

To allow this, a Windows Azure application can have multiple *instances*, each executing in its own virtual machine (VM). These VMs run 64-bit Windows Server 2008, and they're provided by a hypervisor (based on Hyper-V) that's been modified for use in Microsoft's cloud. To run an application, a developer accesses the Windows Azure portal through her Web browser, signing in with a Windows Live ID. She then chooses whether to create a *hosting* account for running applications, a *storage* account for storing data, or both. Once the developer has a hosting account, she can upload her application, specifying how many instances the application needs. Windows Azure then creates the necessary VMs and runs the application.

It's important to note that a developer can't supply her own VM image for Windows Azure to run. Instead, the platform itself provides and maintains its own copy of Windows. Developers focus solely on creating applications that run on Windows Azure.

In the initial incarnation of Windows Azure, known as the Community Technology Preview (CTP), two different instance types are available for developers to use: Web role instances and Worker role instances. Figure 3 illustrates this idea.

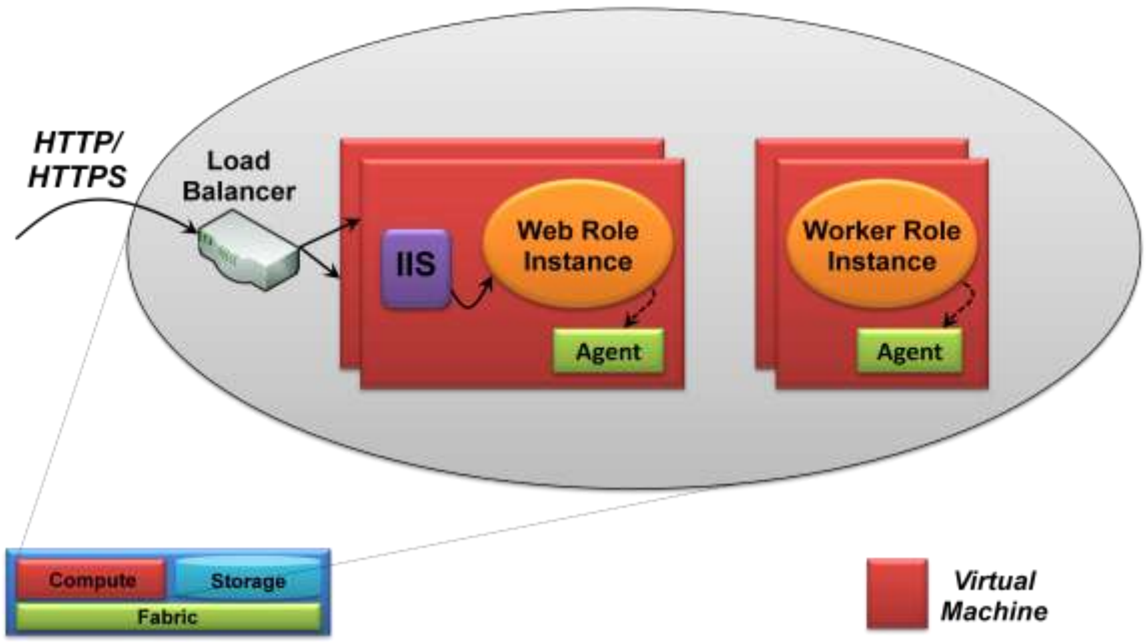

Figure 3: In the CTP version, Windows Azure applications can consist of Web role instances and/or Worker role instances, each of which runs in its own style of virtual machine.

As its name suggests, a Web role instance can accept incoming HTTP or HTTPS requests. To allow this, it runs in a VM that includes Internet Information Services (IIS) 7. Developers can create Web role instances using ASP.NET, WCF, or another .NET technology that works with IIS. Developers can also create applications in native code—using the .NET Framework isn't required. (This means that developers can upload and run other technologies as well, such as PHP.) And as Figure 3 shows, Windows Azure provides built-in hardware load balancing to spread requests across Web role instances that are part of the same application.

By running multiple instances of an application, Windows Azure helps that application scale. To accomplish this, however, Web role instances must be stateless. Any client-specific state should be written to Windows Azure storage or passed back to the client after each request. Also, because the Windows Azure load balancer doesn't allow creating an affinity with a particular Web role instance, there's no way to guarantee that multiple requests from the same user will be sent to the same instance.

Worker role instances aren't quite the same as their Web role cousins. For example, they can't accept requests from the outside world. Their VMs don't run IIS, and a Worker application can't accept any incoming network connections. Instead, a Worker role instance initiates its own requests for input. It can read messages from a queue, for instance, as described later, and it can open connections with the outside world. Given this more self-directed nature, Worker role instances can be viewed as akin to a batch job or a Windows service.

A developer can use only Web role instances, only Worker role instances, or a combination of the two to create a Windows Azure application. If the application's load increases, he can use the Windows Azure portal to request more Web role instances, more Worker role instances, or more of both for his application. If the load decreases, he can reduce the number of running instances. To shut down the application completely, the developer can shut down all of the application's Web role and Worker role instances.

The VMs that run both Web role and Worker role instances also run a Windows Azure *agent*, as Figure 3 shows. This agent exposes a relatively simple API that lets an instance interact with the Windows Azure fabric. For example, an instance can use the agent to write to a Windows Azure-maintained log, send alerts to its owner via the Windows Azure fabric, and do a few more things.

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 with WCF and C#. Similarly, she might create a Worker role in one of these .NET languages or directly in C++ without the .NET Framework. 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.

Both Web role instances and Worker role instances are free to access their VM's local file system. This storage isn't persistent, however: When the instance is shut down, the VM and its local storage go away. Yet applications commonly need persistent storage that holds on to information even when they're not running. Meeting this need is the goal of the Windows Azure Storage service, described next.

THE STORAGE SERVICE

Applications work with data in many different ways. Accordingly, the Windows Azure Storage service provides several options. Figure 4 shows what's in the CTP version of this technology.

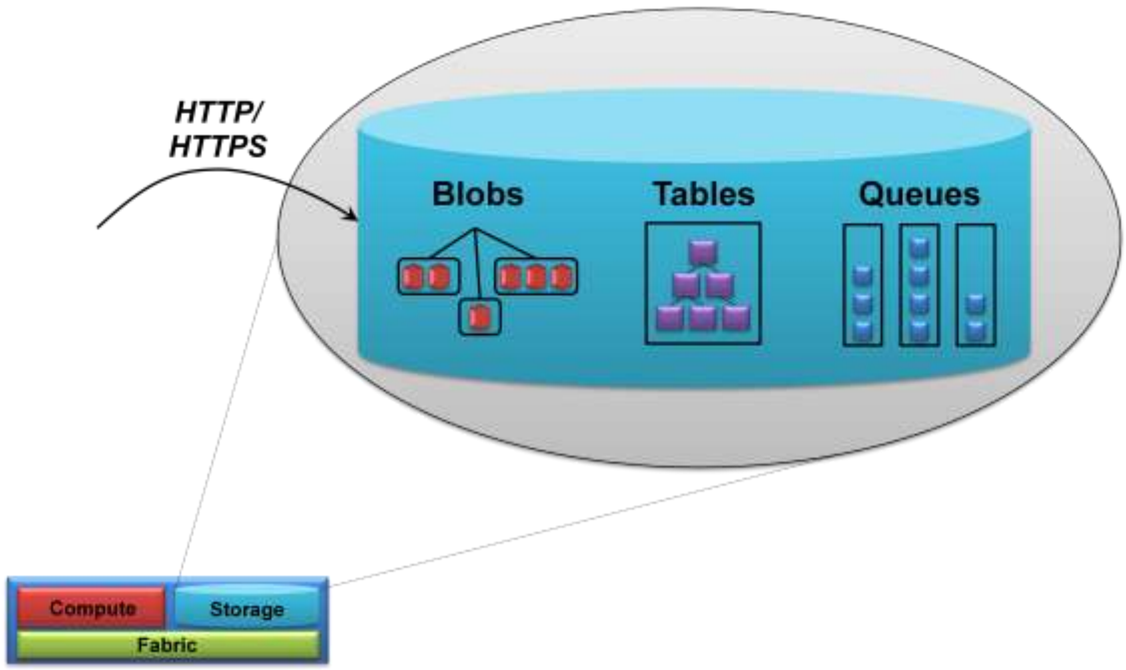

Figure 4: Windows Azure Storage provides blobs, tables, and queues.

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: A storage account can have one or more *containers*, each of which holds one or more blobs. Blobs can be big—up to 50 gigabytes each—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 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. In fact, even though they're called "tables", the data they hold is actually stored in a simple hierarchy of *entities* that contain *properties*. And rather than using SQL, an application accesses a table's data using the conventions defined by ADO.NET Data Services. The reason for this apparently idiosyncratic approach is that it allows *scale-out storage*—scaling by spreading data spread across many machines—much 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 with Worker role instances. For example, a user might submit a request to perform some compute-intensive task via a Web page 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 just wrote.

Windows Azure storage can be accessed by a Windows Azure application, by an application running on-premises within some organization, or by an application running at a hoster. 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. In other words, blobs, tables, and queues are all named using URIs and accessed via standard HTTP operations. A .NET client might use the ADO.NET Data Services libraries to do this, but it's not required—an application can also make raw HTTP calls.

THE FABRIC

All Windows Azure applications and all of the data in Windows Azure Storage live in some Microsoft data center. Within that data center, the set of machines dedicated to Windows Azure is organized into a fabric. Figure 5 shows how this looks.

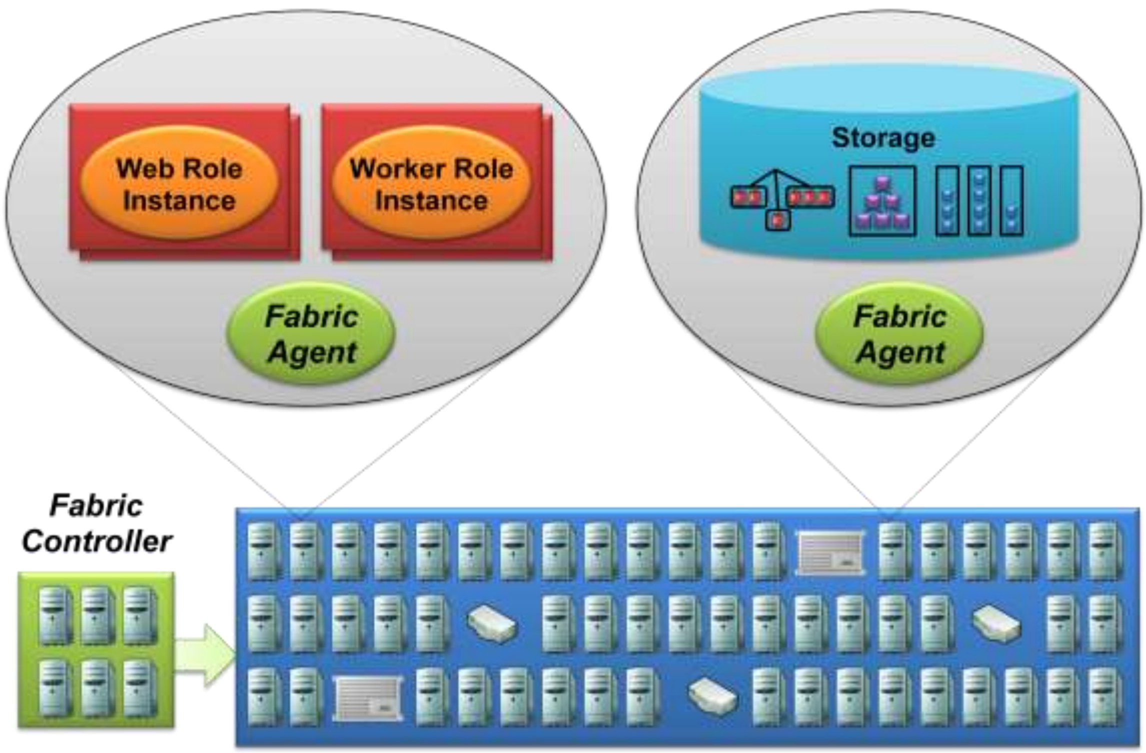

Figure 5: The fabric controller interacts with Windows Azure applications via the fabric agent.

As the figure shows, the Windows Azure Fabric consists of a (large) group of machines, all of which are managed by software called the *fabric controller*. The fabric controller is replicated across a group of five to seven machines, and it owns all of the resources in the fabric: 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 many useful things. It monitors all running applications, for example, giving it an up-to-the-minute picture of what's happening in the fabric. It manages operating systems, taking care of things like patching the version of Windows Server 2008 that runs in Windows Azure VMs. It also decides where new applications should run, choosing physical servers to optimize hardware utilization.

To do this, the fabric controller depends on a configuration file that is 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 receives this new application, it uses this configuration file to determine how many Web role and Worker role VMs to create.

Once it's created these 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 restart a new one. Similarly, if the machine a VM is running on dies, the fabric controller will start a new instance of the Web or Worker role in a new VM on another machine, resetting the load balancer as necessary to point to this new machine.

While this might change over time, the fabric controller in the Windows Azure CTP maintains a one-to-one relationship between a VM and a physical processor core. Because of this, performance is predictable—each application instance has its own dedicated processor core. It also means that there's no arbitrary limit on how long an application instance can execute. A Web role instance, for example, can take as long as it needs to handle a request from a user, while a Worker role instance can compute the value of pi to a million digits if necessary. Developers are free to do what they think is best.

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 this platform is to walk through examples of how it can be used. Accordingly, this section looks at four core scenarios for using Windows Azure: creating a scalable Web application, creating a parallel processing application, creating a Web application with background processing, 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 scale-out applications and scale-out 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 news video sites with occasional hot stories, sites that are 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, this is likely to be cheaper than maintaining lots of mostly unused machines.

To create a scalable Web application on Windows Azure, a developer can use Web roles and tables. Figure 6 shows a simple illustration of how this looks.

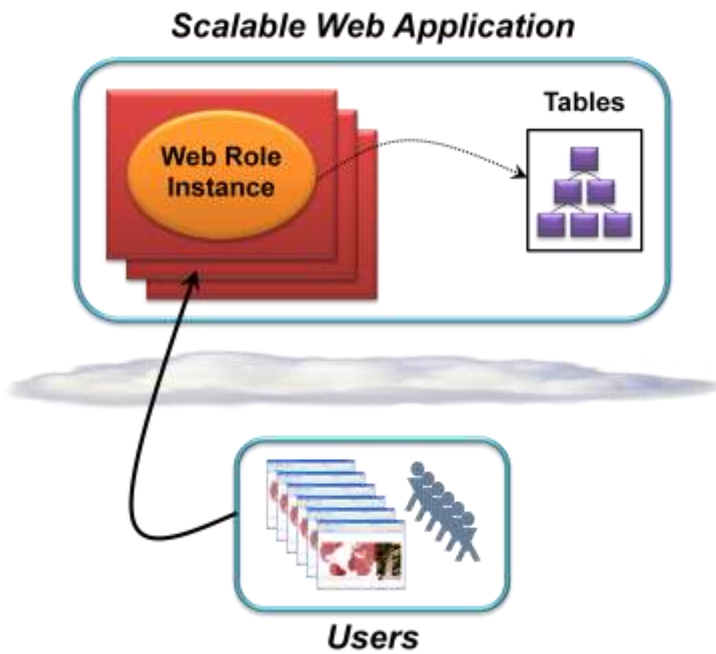

Figure 6: A scalable Web application can use Web role instances and tables.

In the example shown here, the clients are browsers, and so the application logic might 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. In either case, the developer specifies how many instances of the application 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: rendering at a film special effects house, new drug development in a pharmaceutical company, financial modeling at a bank, 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 something like an on-demand supercomputer.

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 binary datasets. In Windows Azure, this means using blobs. Figure 7 shows a simple illustration of how this kind of application might look.

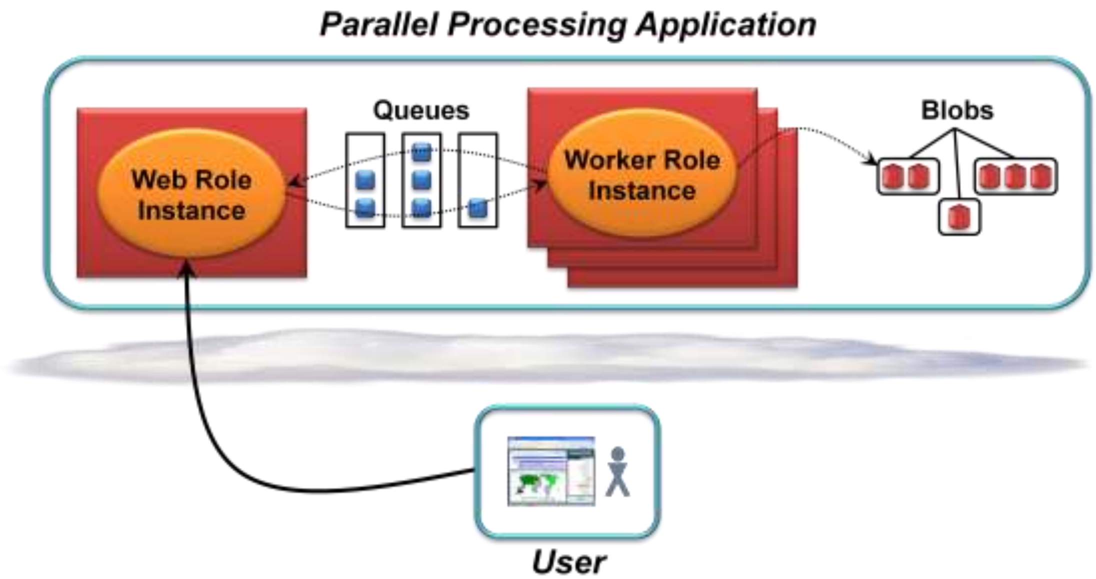

Figure 7: A parallel processing application might use a Web role instance, many Worker role instances, queues, and blobs.

In the scenario shown here, the parallel work is done by some 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, get results, and more. Communication between the Web role instance and the Worker role instances relies on Windows Azure Storage queues.

Those queues can also be accessed directly by an on-premises application. Rather than relying on a Web role instance running on Windows Azure, the user might instead interact with the Worker role instances via an on-premises application to. Figure 8 shows this situation.

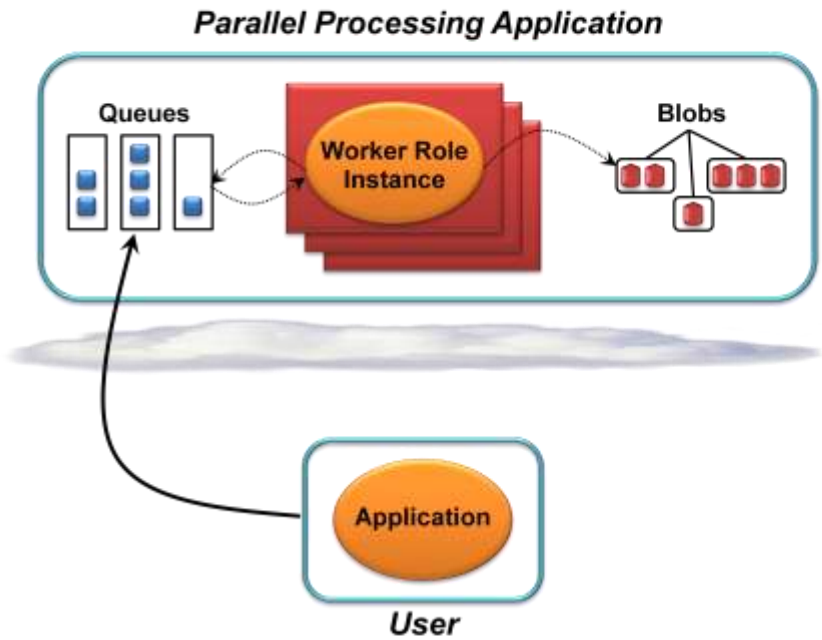

Figure 8: A parallel processing application can communicate with an on-premises application through queues.

In this example, the parallel work is accomplished just as before: Multiple Worker role instances run simultaneously, each interacting with the outside world via queues. Here, however, work is put into those queues directly by an on-premises application. In a scenario like this, the user might have no idea that the on-premises application he's using relies on Windows Azure for parallel processing.

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 also 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.

Windows Azure Web roles and Worker roles can be used together to address this scenario. Figure 9 shows how this kind of application might look.

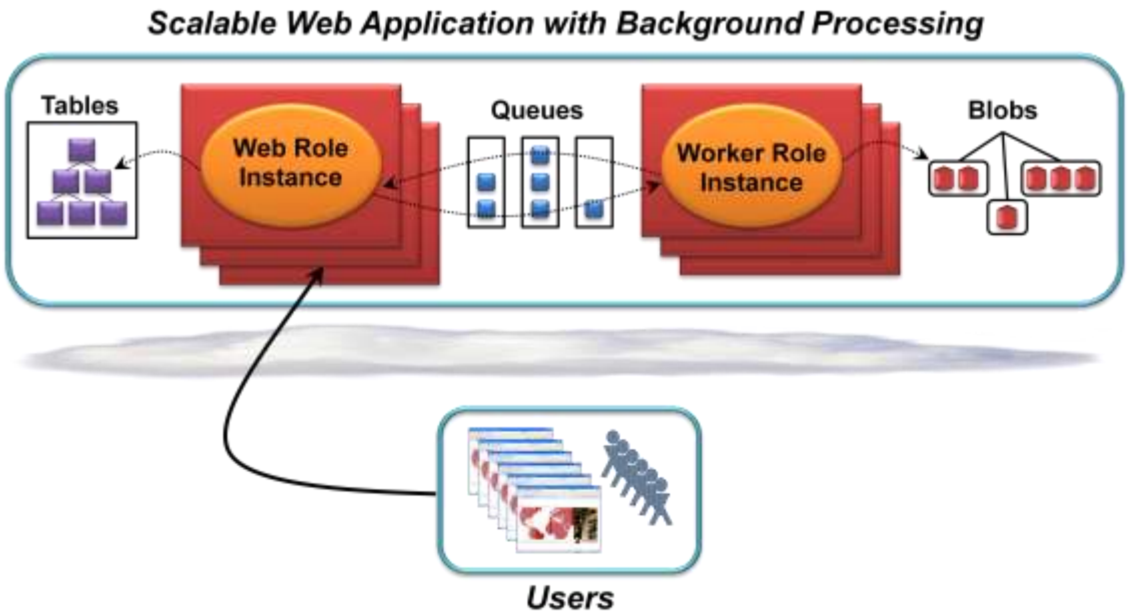

Figure 9: A scalable Web application with background processing might use all of Windows Azure's capabilities.

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 also uses tables to store information. For background processing, it relies on Worker role instances, passing them tasks via queues. In this example, those Worker instances work on blob data, but other approaches are also possible.

This example shows how an application might use all of the basic capabilities that Windows Azure exposes: Web role instances, Worker role instances, blobs, tables, and queues. While not every application needs all of these, having them all available is essential to support more complex scenarios like this one.

USING CLOUD STORAGE FROM AN ON-PREMISES OR HOSTED APPLICATION

Complex cloud platform scenarios like the one just described can be useful. Sometimes, though, an application needs only one of Windows Azure's capabilities. 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. Similarly, a news Web site running at a hoster might need a globally accessible, scalable 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. Figure 10 illustrates this idea.

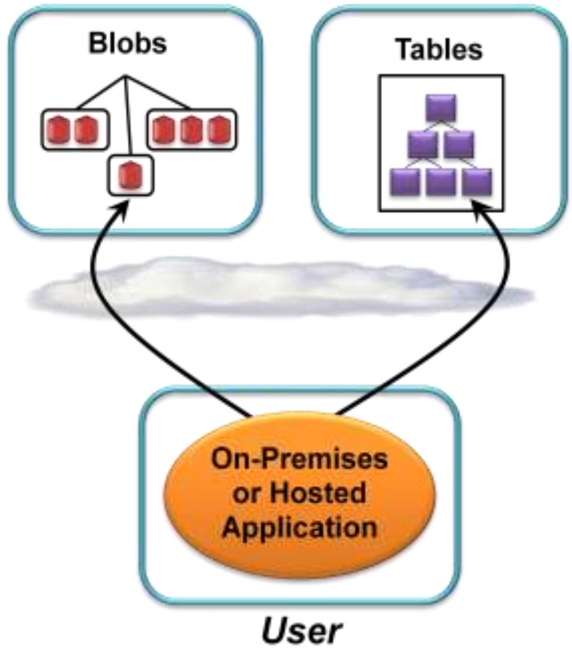

Figure 10: An on-premises or hosted application can use Windows Azure blobs and tables to store its data in the cloud.

As the figure shows, an on-premises or hosted application can directly access Windows Azure's 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.

Supporting the four scenarios described in this section—scalable Web applications, parallel processing applications, scalable Web applications with background processing, and non-cloud applications accessing cloud storage—is a fundamental goal for the Windows Azure CTP. As this cloud platform grows, however, expect the range of problems it addresses to expand as well. The scenarios described here are important, but they're not the end of the story.

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 the platform's more interesting aspects.

DEVELOPING WINDOWS AZURE APPLICATIONS

For developers, building a Windows Azure application looks much like building a traditional Windows application. As described earlier, the platform supports both .NET applications and applications built using unmanaged code, so a developer can use whatever best fits her problem. To make life easier, Windows Azure provides Visual Studio 2008 project templates for creating Web roles, Worker roles, and applications that combine the two.

One obvious difference, however, is that Windows Azure applications don't run locally. This difference has the potential to make development more challenging (and more expensive, since using Windows Azure resources isn't free). To mitigate this, Microsoft provides the *development fabric*, a version of the Windows Azure environment that runs on a developer's machine. Figure 11 shows how this looks.

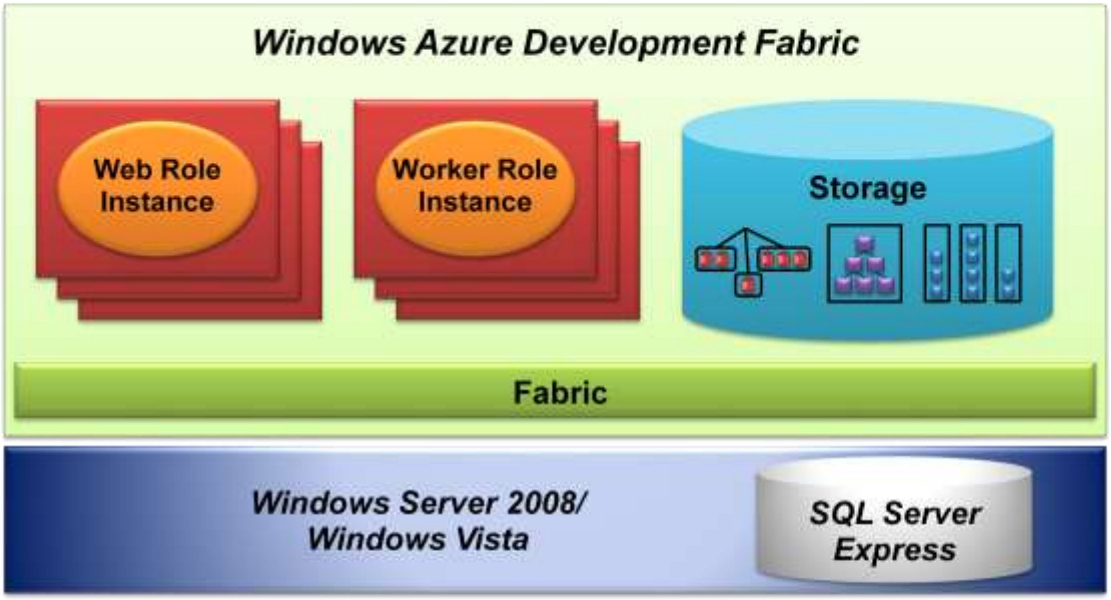

Figure 11: The development fabric provides a local facsimile of Windows Azure for developers.

The development fabric runs on a single machine running either Windows Server 2008 or Windows Vista. It emulates the functionality of Windows Azure in the cloud, complete with Web roles, Worker 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. (In fact, it's entirely possible to create a Windows Azure application without ever using Windows Azure in the cloud.) Once the application has been developed and tested locally, the developer can upload the code and its configuration file via the Windows Azure portal, then run it.

Still, some things are different in the cloud. You can't attach a debugger to an application running on Windows Azure, for example, and so developers must rely on logging. Yet even logging could be problematic. Several instances of a Windows Azure application are typically running simultaneously, and life would be simpler if they could write to a common log file. Fortunately, they can: As mentioned earlier, this is a service provided by the Windows Azure agent. By calling an agent API, all writes to a log by all instances of a Windows Azure application can be written to a single log file.

Windows Azure also provides other services for developers. For example, a Windows Azure application can send an alert string through the Windows Azure agent, and the platform will forward that alert via email, instant messaging, or some other mechanism to its recipient. If desired, the Windows Azure fabric can itself detect an application failure and send an alert. The Windows Azure platform also provides detailed information about the application's resource consumption, including processor time, incoming and outgoing bandwidth, and storage.

EXAMINING THE COMPUTE SERVICE

Sometimes, you might be happy letting Microsoft choose which data center your application and its data live in. In other situations, however, you might need more control. Suppose your data needs to remain within the European Union for legal reasons, for example, or maybe most of your customers are in North America. In situations like these, you want to be able to specify the data centers in which your application runs and stores its data.

To allow this, Windows Azure lets a developer indicate which data center an application should run in and where its data should be stored. She can also specify that a particular group of applications and/or data should all run in the same data center. Microsoft is initially providing Windows Azure data centers only in the United States, but a European data center will also be available in the not-too-distant future.

Wherever it runs, a Windows Azure application is installed and made available to its users in a two-step process. A developer first uploads the application to the platform's staging area. The staged application's HTTP/HTTPS endpoint has a DNS name of the form *<GUID>*.cloudapp.net, where *<GUID>* represents a globally unique identifier assigned by Windows Azure. This DNS name is associated with a virtual IP address (VIP) that identifies the Windows Azure load balancer through which the application can be accessed.

When the developer is ready to make the application live, she uses the Windows Azure portal to request that it be put into production. Windows Azure then atomically changes its DNS server entry to associate the application's VIP with the production DNS name the developer has chosen, such as myazureservice.cloudapp.net. (To use a custom domain rather than Microsoft's cloudapp.net domain, the owner of a Windows Azure application can create a DNS alias using a standard CNAME.)

A couple of things about this process are worth pointing out. First, because the VIP swap is atomic, a running application can be upgraded to a new version with no downtime. This is important for many kinds of cloud services. Second, notice that throughout this process, the actual IP addresses of the Windows Azure VMs—and the physical machines those VMs run on—are never exposed.

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 application might use a membership provider to store its own user ID and password, for example, just like any other ASP.NET application, or it might use some other method, such as Microsoft's LiveID service. The choice is entirely up to the application's creator.

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. 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 contains.

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 supplies 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. Containers can't be nested—they can contain only blobs, not other containers—so it's not possible to create a hierarchy of blobs. Still, it's legal for a blob name to contain a "/", so a developer can create the illusion of a hierarchy if desired.

Recall that blobs can be large—up to 50 gigabytes—and so to make transferring them more efficient, each blob can be 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.

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 video, photos, or other unstructured data generally available on the Internet.

Tables

A blob is easy to understand—it's just a slab of bytes—but tables are a bit more complex. Figure 12 illustrates how the parts of a table fit together.

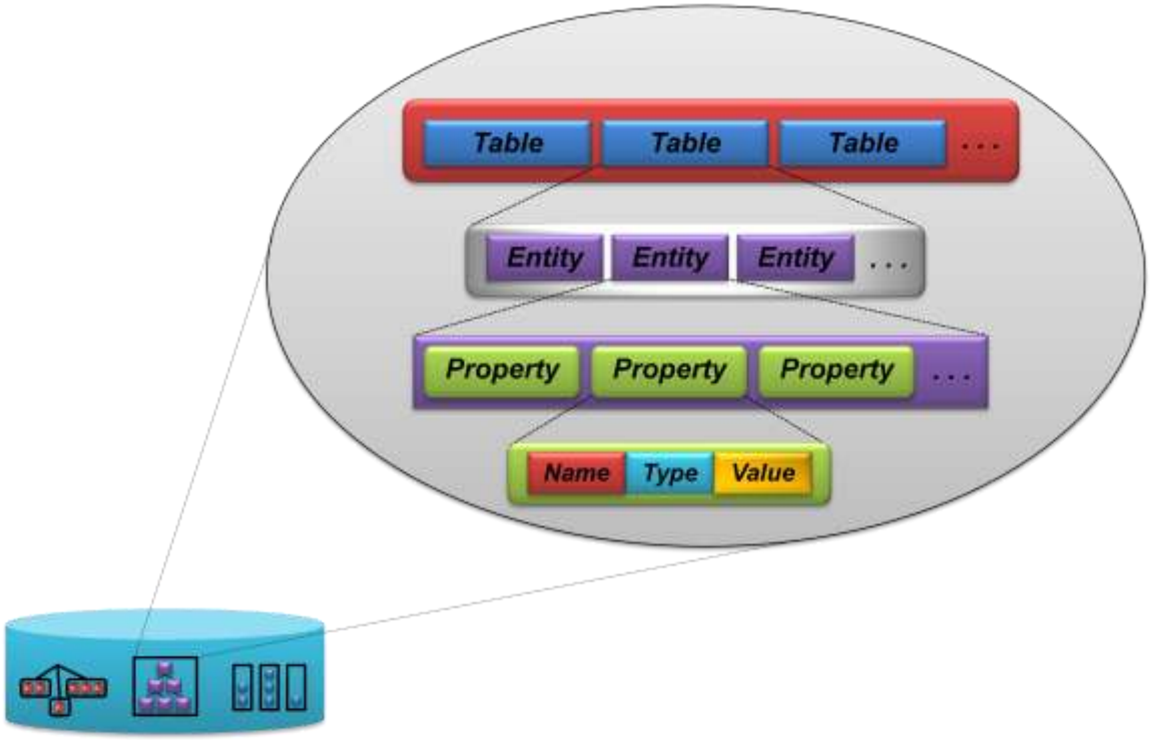

Figure 12: Tables provide hierarchical storage.

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, and a property can take on different types at different times depending on the value stored in it. Furthermore, 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 her 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 atomically replaces all of its properties. (Microsoft has also stated that tables will support atomic multi-entity writes within a single table by the time of Windows Azure's first commercial release.)

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 the primary Windows Azure goal of 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 anymore. 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 structures can't be applied unchanged. Still, for creating very scalable applications, this approach makes sense. For one thing, 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 ADO.NET Data Services or Language Integrated Query (LINQ) to do this, both of which hide 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 repetitively allows retrieving the complete result set in chunks.

Updates pose another problem: What happens if multiple applications attempt to update the same entity simultaneously? Updating an entity requires reading that entity, changing its contents by modifying, adding, and/or deleting properties, then writing the updated entity back to the same table. Suppose that two applications both read the same entity, modify it, then write it back—what happens? The default answer is that the application whose write gets there first will succeed. The other application's write will fail. This approach, an example of optimistic concurrency, relies on version numbers maintained by Windows Azure tables. Alternatively, an application can unconditionally update an entity, guaranteeing that its changes will be written. (And although it wasn't mentioned earlier, blobs offer the same two approaches to handling concurrent updates.)

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 13 shows how this looks.

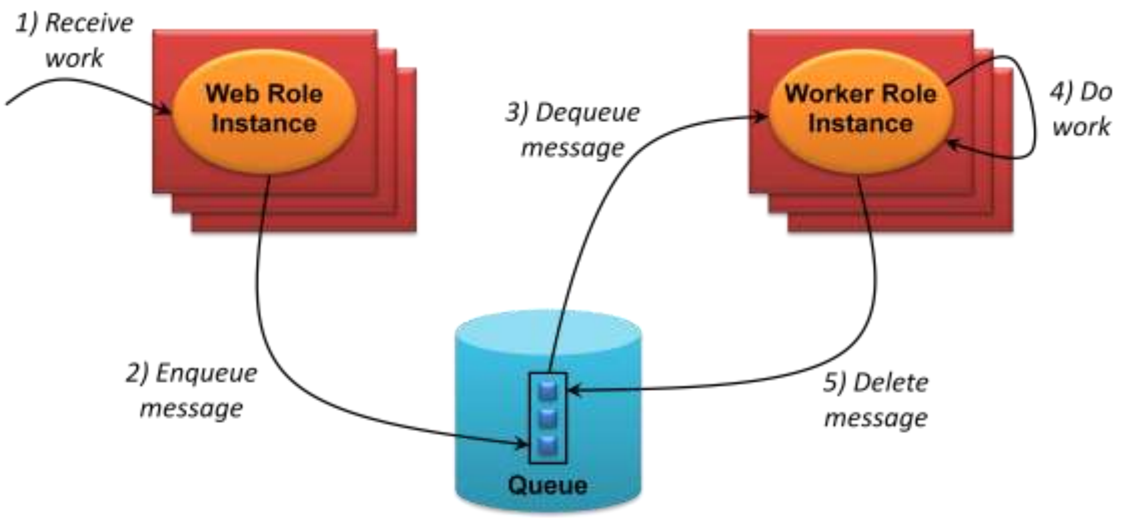

Figure 13: Messages are enqueued, dequeued, processed, then explicitly deleted from the queue.

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 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 entity in a table, or something else—it's up to the application. Worker 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 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

To an application developer, Windows Azure consists of the Compute service and the Storage service. Yet neither one could function without the Windows Azure Fabric. By knitting together a data center full of machines into a coherent whole, the Fabric provides a foundation for everything else.

As described earlier, the fabric controller owns all resources in a particular Windows Azure data center. It's also responsible for assigning instances of both applications and storage 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 14 illustrates this idea.

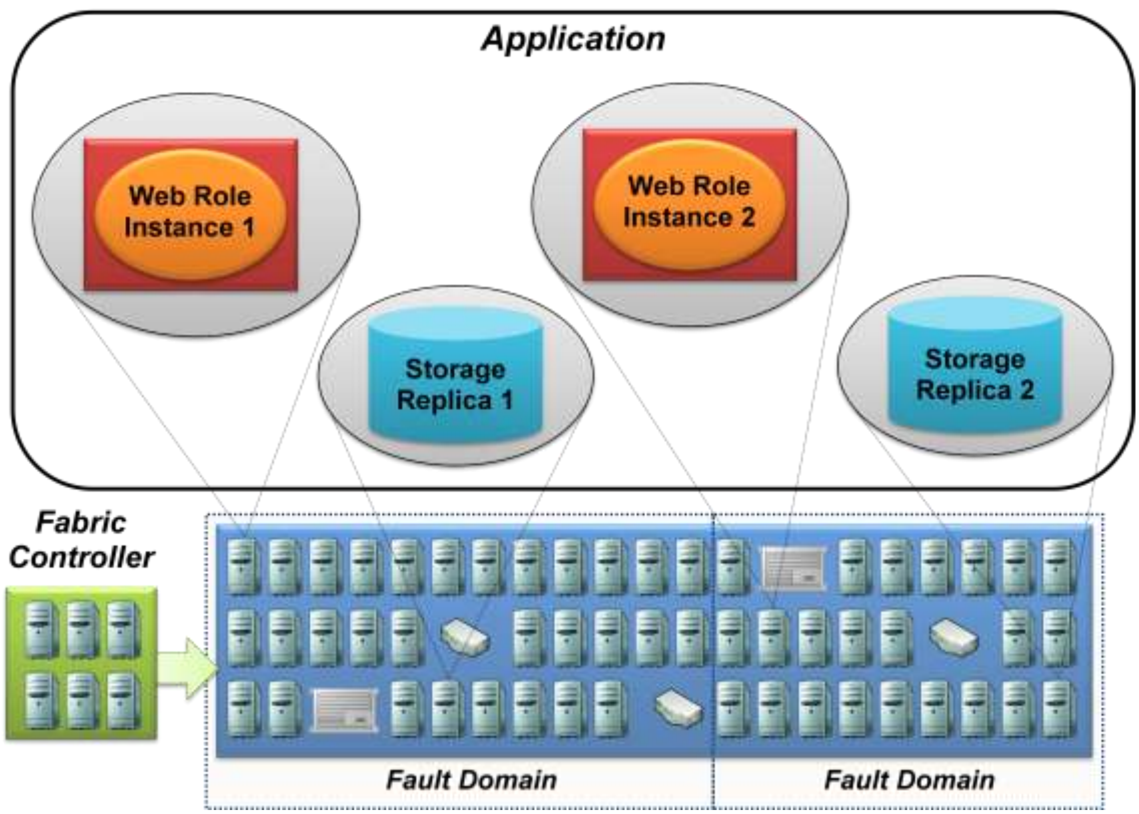

Figure 14: The fabric controller places different instances of an application in different fault domains.

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.

CONCLUSIONS

Running applications and storing data in the cloud is the right choice for many situations. Windows Azure's three parts—the Compute service, the Storage service, and the Fabric—work together to make this possible. Together with the Windows Azure development environment, they provide a bridge for Windows developers moving into this new world.

Today, cloud platforms are an 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

- Azure Home Page

  http://www.microsoft.com/azure

- Introducing the Azure Services Platform, David Chappell

  http://download.microsoft.com/download/e/4/3/e43bb484-3b52-4fa8-a9f9-ec60a32954bc/Azure_Services_Platform.pdf

- Windows Azure Blobs: Programming Blob Storage

  http://download.microsoft.com/download/D/6/E/D6E0290E-8919-4672-B3F7-56001BDC6BFA/Windows%20Azure%20Blob%20-%20Dec%202008.docx

- Windows Azure Tables: Programming Table Storage

  http://download.microsoft.com/download/3/B/1/3B170FF4-2354-4B2D-B4DC-8FED5F838F6A/Windows%20Azure%20Table%20-%20Dec%202008.docx

- Windows Azure Queues: Programming Queue Storage

  http://download.microsoft.com/download/5/2/D/52D36345-BB08-4518-A024-0AA24D47BD12/Windows%20Azure%20Queue%20-%20Dec%202008.docx

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.