David Chappell
& Associates

INTRODUCING THE WINDOWS AZURE PLATFORM

DAVID CHAPPELL

DECEMBER 2009

SPONSORED BY MICROSOFT CORPORATION

CONTENTS

AN OVERVIEW OF THE WINDOWS AZURE PLATFORM

Using computers in the cloud can make lots of sense. Rather than buying and maintaining your own machines, why not exploit the acres of Internet-accessible servers on offer today? For some applications, both code and data might live in the cloud, where somebody else manages and maintains the systems they use. Alternatively, applications that run inside an organization—on-premises applications—might store data in the cloud or rely on other cloud infrastructure services. However it's done, exploiting the cloud's capabilities can improve our world.

But whether an application runs in the cloud, uses services provided by the cloud, or both, some kind of application platform is required. Viewed broadly, an application platform can be thought of as anything that provides developer-accessible services for creating applications. In the local, on-premises Windows world, for example, this includes technologies such as Windows Server, the .NET Framework, SQL Server, and more. To let applications exploit the cloud, cloud application platforms must also exist.

Microsoft's Windows Azure platform is a group of cloud technologies, each providing a specific set of services to application developers. As Figure 1 shows, the Windows Azure platform can be used both by applications running in the cloud and by on-premises applications.

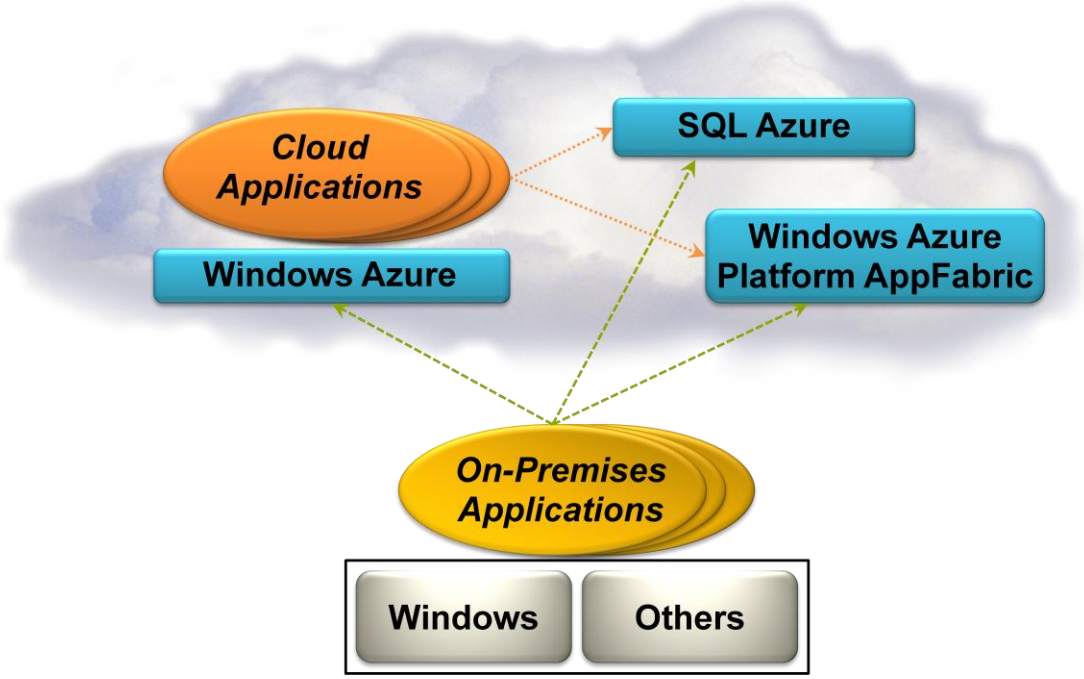

Figure 1: The Windows Azure platform supports applications, data, and infrastructure in the cloud.

The components of the Windows Azure platform are:

- *Windows Azure*: Provides a Windows-based environment for running applications and storing data on servers in Microsoft data centers.
- *SQL Azure*: Provides data services in the cloud based on SQL Server.

- *Windows Azure platform AppFabric*: Provides cloud services for connecting applications running in the cloud or on premises.

Each part of the Windows Azure platform has its own role to play. This overview describes all three, first at a high level, then in a bit more detail. The goal is to provide a big-picture introduction to this new application platform.

WINDOWS AZURE

At a high level, Windows Azure is simple to understand: It's a platform for running Windows applications and storing their data in the cloud. Figure 2 shows its main components.

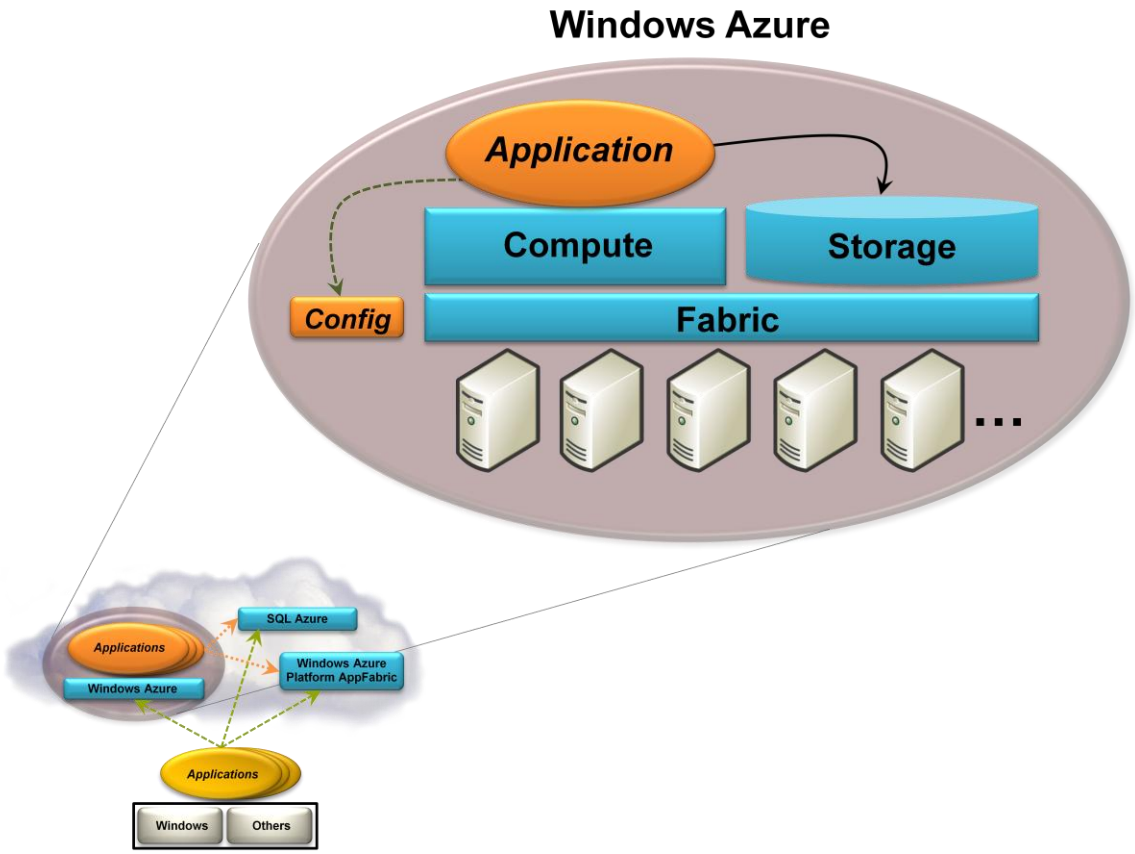

Figure 2: Windows Azure provides Windows-based compute and storage services for cloud applications.

As the figure suggests, Windows Azure runs on a large number of machines, all located in Microsoft data centers and accessible via the Internet. A common Windows Azure fabric knits this plethora of processing power into a unified whole. Windows Azure compute and storage services are built on top of this fabric.

The Windows Azure compute service is based, of course, on Windows. Developers can build applications using the .NET Framework, unmanaged code, or other approaches. Those applications are written in ordinary Windows languages, such as C#, Visual Basic, C++, and Java, using Visual Studio or another development tool. Developers can create Web applications, using technologies such as ASP.NET, Windows

Communication Foundation (WCF), and PHP, applications that run as independent background processes, or applications that combine the two.

Both Windows Azure applications and on-premises applications can access the Windows Azure storage service, and both do it in the same way: using a RESTful approach. This service allows storing binary large objects (blobs), provides queues for communication between components of Windows Azure applications, and even offers a form of tables with a simple query language. For applications that need traditional relational storage, the Windows Azure platform provides SQL Azure Database, described later. An application using the Windows Azure platform is free to use any combination of these storage options.

Running applications and storing their data in the cloud can have clear benefits. Rather than buying, installing, and operating its own systems, for example, an organization can rely on a cloud provider to do this for them. Also, customers pay just for the computing and storage they use, rather than maintaining a large set of servers only for peak loads. And if they're written correctly, applications can scale easily, taking advantage of the enormous data centers that cloud providers offer.

Yet achieving these benefits requires effective management. In Windows Azure, each application has a configuration file, as shown in Figure 2. By changing this configuration manually or programmatically, an application's owner can control various aspects of its behavior, such as setting the number of application instances that Windows Azure should run. The Windows Azure fabric then monitors the application to maintain this desired state.

To let its customers create, configure, and monitor applications, Windows Azure provides a browser-accessible portal. A customer provides a Windows Live ID, then chooses whether to create a *hosting* account for running applications, a *storage* account for storing data, or both. An application is free to charge its customers in any way it likes: subscriptions, per-use fees, or anything else.

Windows Azure is a quite general platform that can be used in various scenarios. Here are a few examples:

- A start-up creating a new Web site—the next Facebook, say—could build its application on Windows Azure. Because this platform supports both Web-facing services and background processes, the application can provide an interactive user interface as well as executing work for users asynchronously. Rather than spending time and money worrying about infrastructure, the start-up can instead focus solely on creating code that provides value to its customers and investors. The company can also start small, incurring low costs while its application has only a few users. If the application catches on and usage increases, Windows Azure can scale the application as needed.

- An independent software vendor (ISV) creating a software-as-a-service (SaaS) version of an existing on-premises Windows application might choose to build it on Windows Azure. Because Windows Azure mostly provides a standard Windows environment, moving the application's business logic to this cloud platform won't typically pose many problems. And once again, building on an existing platform lets the ISV focus on their business logic—the thing that makes them money—rather than spending time on infrastructure.

- An enterprise creating an application for its customers might choose to build it on Windows Azure. Because Windows Azure supports .NET, developers with the right skills aren't difficult to find, nor are they prohibitively expensive. Running the application in Microsoft's data centers frees the enterprise

from the responsibility and expense of managing its own servers, turning capital expenses into operating expenses. And especially if the application has spikes in usage—maybe it's an on-line flower store that must handle the Mother's Day rush—letting Microsoft maintain the large server base required for this can make economic sense.

Running applications in the cloud is one of the most important aspects of cloud computing. With Windows Azure, Microsoft provides a platform for doing this, along with a way to store application data. As interest in cloud computing continues to grow, expect to see more Windows applications created for this new world.

SQL AZURE

One of the most attractive ways of using Internet-accessible servers is to handle data. The goal of SQL Azure is to address this area, offering cloud-based services for storing and working with information. While Microsoft says that SQL Azure will eventually include a range of data-oriented capabilities, including data synchronization, reporting, data analytics, and others, the first SQL Azure component to appear is SQL Azure Database. Figure 3 illustrates this.

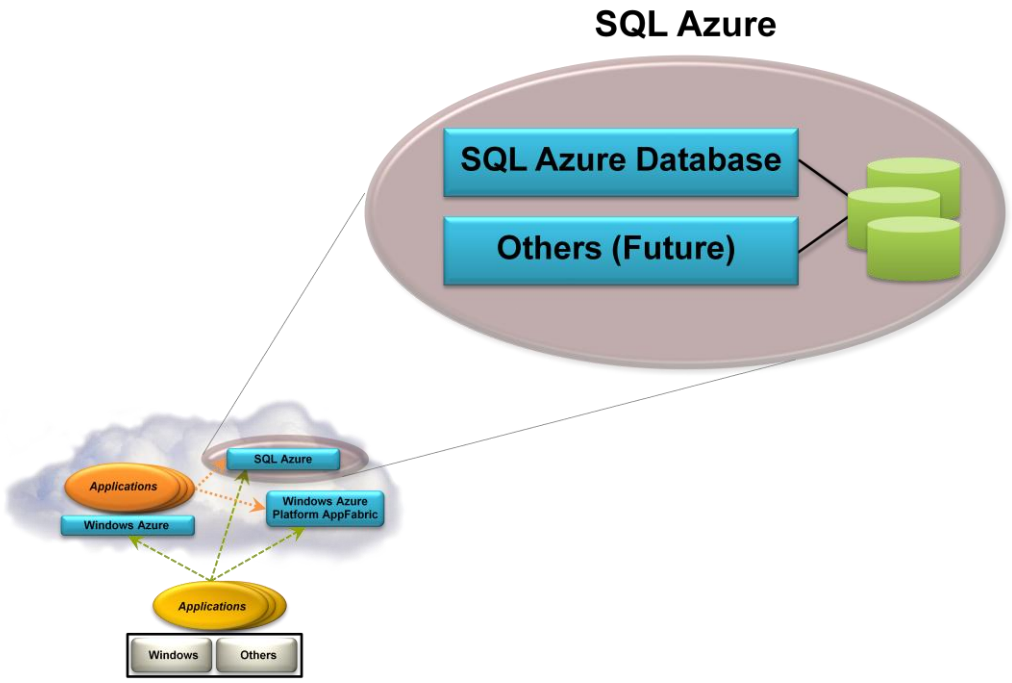

Figure 3: SQL Azure provides data-oriented services in the cloud.

SQL Azure Database provides a cloud-based database management system (DBMS). This technology lets on-premises and cloud applications store relational and other types of data on Microsoft servers in Microsoft data centers. As with other cloud technologies, an organization pays only for what it uses, increasing and decreasing usage (and cost) as the organization's needs change. Using a cloud database also allows converting what would be capital expenses, such as investments in disks and DBMS software, into operating expenses.

SQL Azure Database is built on Microsoft SQL Server. To a great extent, this technology offers a SQL Server environment in the cloud, complete with indexes, views, stored procedures, triggers, and more. This data can be accessed using ADO.NET and other Windows data access interfaces. In fact, applications that today access SQL Server locally will largely work unchanged with data in SQL Azure Database. Customers can also use on-premises software such as SQL Server Reporting Services to work with their cloud-based data.

While applications can use SQL Azure Database much as they do a local DBMS, the management requirements are significantly reduced. Rather than worry about mechanics, such as monitoring disk usage and servicing log files, a SQL Azure Database customer can focus on what's important: the data. Microsoft handles the operational details. And like other components of the Windows Azure platform, using SQL Azure Database is straightforward: Just go to a Web portal and provide the necessary information.

Applications might rely on SQL Azure in a variety of ways. Here are some examples:

- A Windows Azure application can store its data in SQL Azure Database. While Windows Azure provides its own storage, relational tables aren't among the options it offers. Since many existing applications use relational storage and many developers know how to work with it, a significant number of Windows Azure applications are likely to rely on SQL Azure Database to work with data in this familiar way. To improve performance, customers can specify that a particular Windows Azure application must run in the same data center in which SQL Azure Database stores that application's information.

- An application in a small business or a department of a larger organization might rely on SQL Azure Database. Rather than storing its data in a SQL Server or Access database running on a computer under somebody's desk, the application can instead take advantage of the reliability and availability of cloud storage.

- Suppose a manufacturer wishes to make product information available both to its dealer network and directly to customers. Putting this data in SQL Azure Database would let it be accessed by applications running at the dealers and by a customer-facing Web application run by the manufacturer itself.

Whether it's for supporting a Windows Azure application, making data more accessible, or other reasons, data services in the cloud can be attractive. As new technologies become available under the SQL Azure umbrella, organizations will have the option to use the cloud for more and more data-oriented tasks.

WINDOWS AZURE PLATFORM APPFABRIC

Running applications and storing data in the cloud are both important aspects of cloud computing. They're far from the whole story, however. Another option is to provide cloud-based infrastructure services. Filling this gap is the goal of Windows Azure platform AppFabric.

The functions provided by AppFabric today address common infrastructure challenges in connecting distributed applications. Figure 4 shows its components.

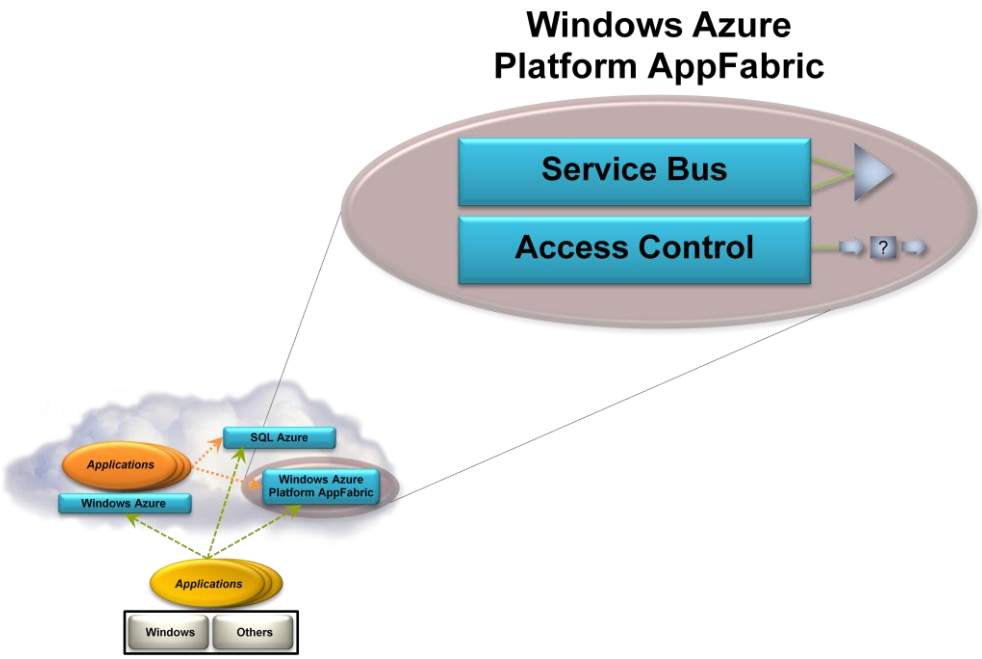

Figure 4: Windows Azure platform AppFabric provides cloud-based infrastructure that can be used by both cloud and on-premises applications.

The components of Windows Azure platform AppFabric are:

- Service Bus: Exposing an application's services on the Internet is harder than it might seem. The goal of Service Bus is to make this simpler by letting an application expose endpoints that can be accessed by other applications, whether on-premises or in the cloud. Each exposed endpoint is assigned a URI, which clients can use to locate and access the service. Service Bus also handles the challenges of dealing with network address translation and getting through firewalls without opening new ports for exposed applications.

- Access Control: This service allows a RESTful client application to authenticate itself and to provide a server application with identity information. The server can then use this information to decide what this application is allowed to do.

As with Windows Azure and SQL Azure, a browser-accessible portal is provided to let customers sign up for AppFabric using a Windows Live ID. Once this has been done, these services can be used in variety of ways, including the following:

- Suppose an enterprise wished to let software at its trading partners access one of its applications. It could expose this application's functions through SOAP or RESTful Web services, then register their endpoints with Service Bus. Its trading partners could then use Service Bus to find these endpoints and access the services.

- An application running on Windows Azure might need to access data stored in an on-premises database. Doing this requires making that database available via the Internet, a problem that can be solved by creating a service that accesses that data, then exposing this service via Service Bus.

- Imagine an enterprise that exposes application services to its trading partners. If those services are exposed using REST, the application could rely on Access Control to authenticate and provide identity information for each client application. Rather than maintaining information internally about each trading partner application, this information could instead be stored in the Access Control service.

As just described, Windows Azure platform AppFabric provides cloud-based infrastructure services. Microsoft is also creating an analogous technology known as Windows Server AppFabric. As its name suggests, the services it provides run on Windows Server—they support on-premises applications—rather than in the cloud. The on-premises services are also different from those in Windows Azure platform AppFabric, focused today on hosting WCF services and on distributed caching. Don't be confused; throughout this paper, the name "AppFabric" is used to refer to the cloud-based services.

Also, don't confuse the Windows Azure platform AppFabric with the fabric component of Windows Azure itself. Even though both contain the term "fabric", they're wholly separate technologies addressing quite distinct problems.

A CLOSER LOOK AT THE TECHNOLOGIES

Having a broad understanding of the Windows Azure platform is an important first step. Getting a deeper understanding of each technology is also useful, however. This section takes a slightly more in-depth look at each member of the family.

WINDOWS AZURE

Windows Azure does two main things: It runs applications and it stores their data. Accordingly, this section is divided into two parts, one for each of these areas.

Running Applications

On Windows Azure, an application typically has multiple *instances*, each running a copy of all or part of the application's code. Each of these instances runs in its own Windows virtual machine (VM). These VMs are provided by a hypervisor that's specifically designed for use in the cloud.

Yet a developer doesn't explicitly create these VMs. He also doesn't supply his own VM image for Windows Azure to run or worry about maintaining a copy of the Windows operating system. Instead, a developer creates applications using *Web roles* and/or *Worker roles*, then tells Windows Azure how many instances of each role to run. Windows Azure silently creates a VM for each instance, then runs the application in those VMs. Figure 5 shows how this looks.

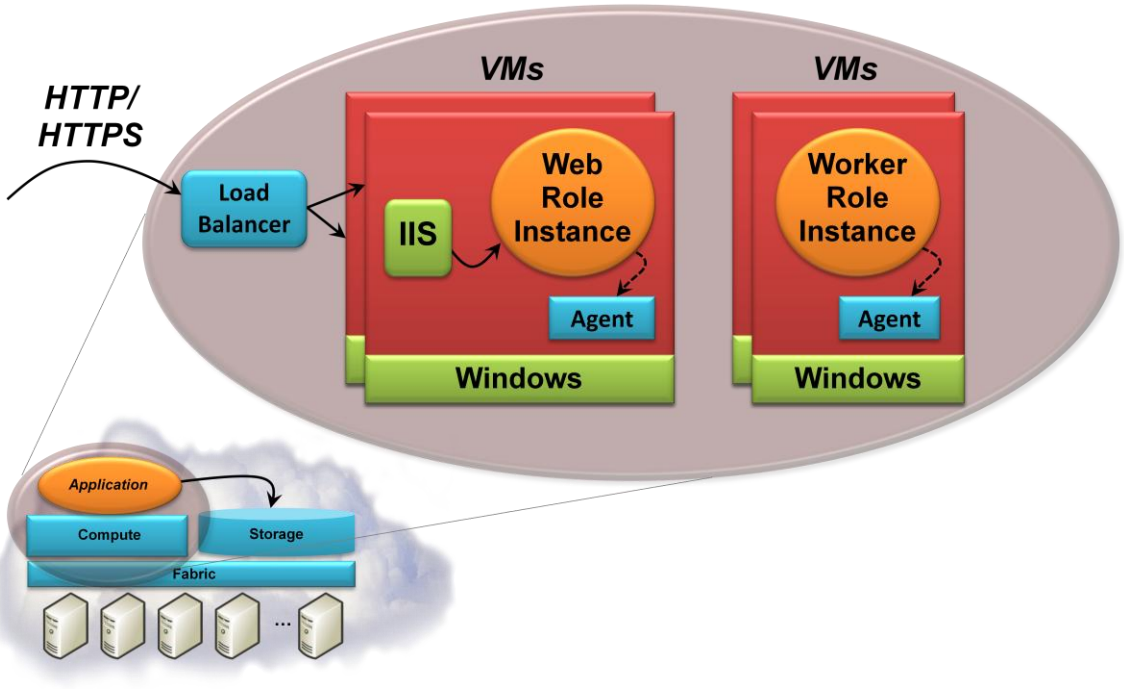

Figure 5: Windows Azure applications can consist of Web role instances and Worker role instances, with each instance running in its own virtual machine.

As its name suggests, each Web role instance accepts incoming HTTP (or HTTPS) requests via Internet Information Services (IIS) 7. A Web role can be implemented using ASP.NET, WCF, or another technology that works with IIS. As Figure 5 shows, Windows Azure provides built-in load balancing to spread requests across Web role instances that are part of the same application.

Web roles are intended to accept and process HTTP requests using IIS. Not everything is a Web application, however, and so Windows Azure also provides Worker roles. A Worker role instance is quite similar to a Web role instance. The key difference is that a Worker role doesn't have IIS preconfigured to run in each instance, and so unlike Web roles, Worker role instances aren't hosted in IIS. A Worker role can still accept requests from the outside world, however, and developers can even run another Web server, such as Apache, in a Worker role instance.

Worker role instances can communicate with Web role instances in various ways. One option is to use Windows Azure storage queues. A Web role instance can insert a work item in a queue, and a Worker role instance can remove and process this item. Another option is for Worker roles and Web roles to set up direct connections via Windows Communication Foundation (WCF) or another technology.

Whether it runs a Web role instance or a Worker role instance, each VM also contains a Windows Azure *agent* that allows the application to interact with the Windows Azure fabric, as Figure 5 shows. The agent exposes a Windows Azure-defined API that lets the instance do things such as find the root of a local storage resource in its VM instance.

For both Web roles and Worker roles, Windows Azure lets developers choose from four VM sizes: one core, two cores, four cores, and eight cores. Since each VM is assigned one or more cores, applications

can have predictable performance. And to increase performance, an application's owner can increase the number of running instances specified in the application's configuration file. The Windows Azure fabric will then spin up new VMs, assign them to cores, and start running more instances of this application. (The fabric doesn't do this automatically in response to changing load, but instead provides APIs that let an application do this itself.) The fabric also detects when a Web role or Worker role instance has failed, then starts a new one.

Notice what this implies: To be scalable, Windows Azure Web role instances must be stateless. Any client-specific state should be written to Windows Azure storage, sent to SQL Azure Database, or passed back to the client in a cookie. Web role statelessness is also all but mandated by Windows Azure's built-in load balancer. Because it 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.

Both Web roles and Worker roles are implemented using standard Windows technologies. Yet moving existing applications to Windows Azure might require a few changes. For example, Windows Azure applications today can run only in user mode—admin mode isn't allowed. To a great degree, however, the world an application sees running on Windows Azure is much like what it sees on an on-premises Windows Server 2008 system.

For developers, building a Windows Azure application also looks much like building a traditional Windows application. Microsoft provides Visual Studio project templates for creating Windows Azure Web roles, Worker roles, and combinations of the two, and developers are free to use any Windows programming language. Also, the Windows Azure software development kit includes a version of the Windows Azure environment that runs on the developer's machine. Known as the Windows Azure *Development Fabric*, it includes Windows Azure storage, a Windows Azure agent, and everything else seen by an application running in the cloud. A developer can create and debug his application using this local simulacrum, then deploy the app to Windows Azure in the cloud when it's ready. Still, some things really are different in the cloud. It's not possible to attach a debugger to a cloud-based application, for example, and so debugging cloud applications relies primarily on writing to a Windows Azure-maintained log. Windows Azure also provides other services for developers, such as information about a running application's CPU consumption, incoming and outgoing bandwidth, and storage.

Accessing Data

Applications work with data in many different ways. Sometimes, all that's required are simple blobs, while other situations call for a more structured way to store information. And in some cases, all that's really needed is a way to exchange data between different parts of an application. Windows Azure storage addresses all three of these requirements, as Figure 6 shows.

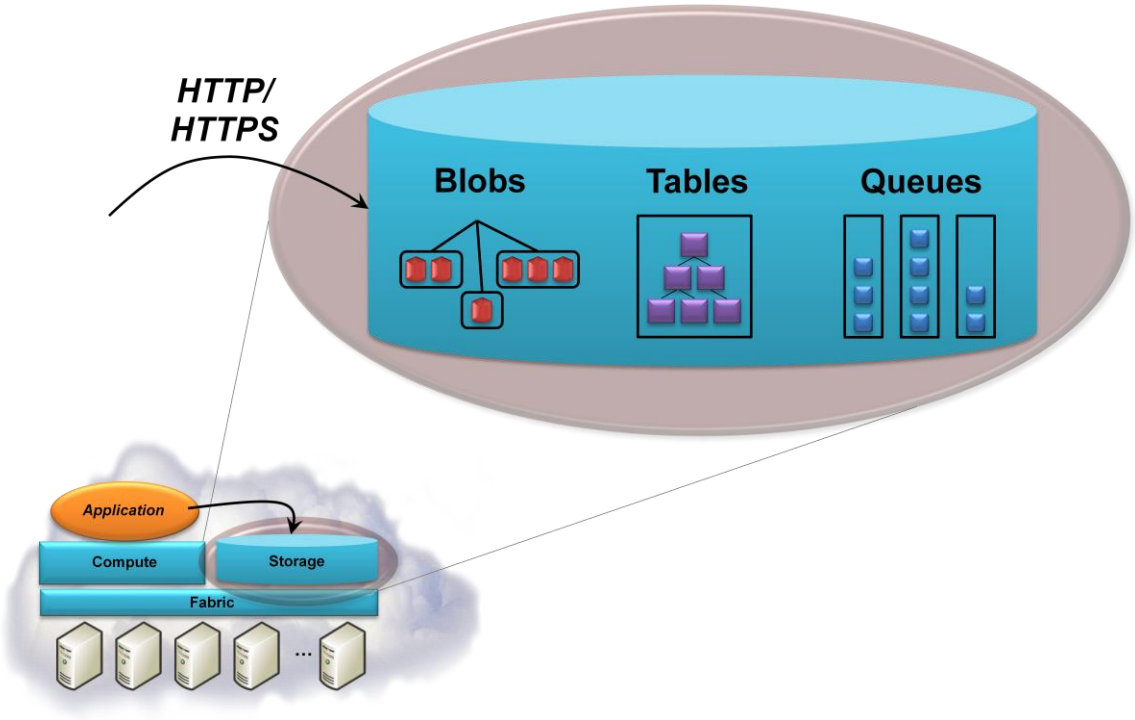

Figure 6: Windows Azure allows storing data in blobs, tables, and queues, all accessed in a RESTful style via HTTP or HTTPS.

The simplest way to store data in Windows Azure storage is to use blobs. As Figure 6 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—potentially as large as a terabyte each—and to make transferring large blobs more efficient, each one can be subdivided into blocks. If a failure occurs, retransmission can resume with the most recent block rather than sending the entire blob again. Blobs can also have associated metadata, such as information about where a JPEG photograph was taken or who the composer is for an MP3 file. And to make distributed access to blob data more efficient, Windows Azure provides a content delivery network (CDN), storing frequently accessed data at locations closer to the applications that use it.

Another way to use blobs is through Windows Azure *XDrives*, which can be mounted by a Web role instance or Worker role instance. The underlying storage for an XDrive is a blob, and so once a drive is mounted, the instance can read and write file system data that gets stored persistently in a blob.

Blobs are just right for some kinds of data, but they're too unstructured for many situations. To allow applications to 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 contain is actually stored in a set of entities with properties. A table has no defined schema; instead, properties can have various types, such as int, string, Bool, or DateTime. And rather than using SQL, an application can access a table's data using ADO.NET Data Services or LINQ. A single table can be quite large, with billions of entities holding terabytes of data, and Windows Azure storage can partition it across many servers if necessary to improve performance.

Blobs and tables are both focused on storing data. The third option in Windows Azure storage, queues, has a somewhat different purpose. A primary use 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 it's stored—in blobs, tables, or queues—all data held in Windows Azure storage is replicated three times. This replication allows fault tolerance, since losing a copy isn't fatal. The system guarantees consistency, however, so an application that reads data it has just written will get what it expects.

Windows Azure storage can be accessed either by a Windows Azure application or by an application running somewhere else. In both cases, all three Windows Azure storage styles use the conventions of REST to identify and expose data. Everything is named using URIs and accessed with standard HTTP operations. A .NET client can rely on ADO.NET Data Services and LINQ, but access to Windows Azure storage from, say, a Java application can just use standard REST.

The Windows Azure platform charges independently for compute and storage resources. This means that an on-premises application could use just Windows Azure storage, accessing its data in the RESTful way just described. And because that data can be accessed directly from non-Windows Azure applications, it remains available even if the Windows Azure application that uses it isn't running.

The goal of application platforms, whether on-premises or in the cloud, is to support applications and data. Windows Azure provides a home for both of these things. Going forward, expect to see a share of what would have been on-premises Windows applications instead running on this new cloud platform.

SQL AZURE

A DBMS in the cloud is attractive for many reasons. For some organizations, letting a specialized service provider ensure reliability, handle back-ups, and perform other management functions makes sense. Data in the cloud can also be accessed by applications running anywhere, even on mobile devices. And given the economies of scale that a service provider enjoys, using a cloud database may well be cheaper than doing it yourself. The goal of SQL Azure Database, the first member of the SQL Azure family, is to provide all of these benefits. Figure 7 shows a simple view of this technology.

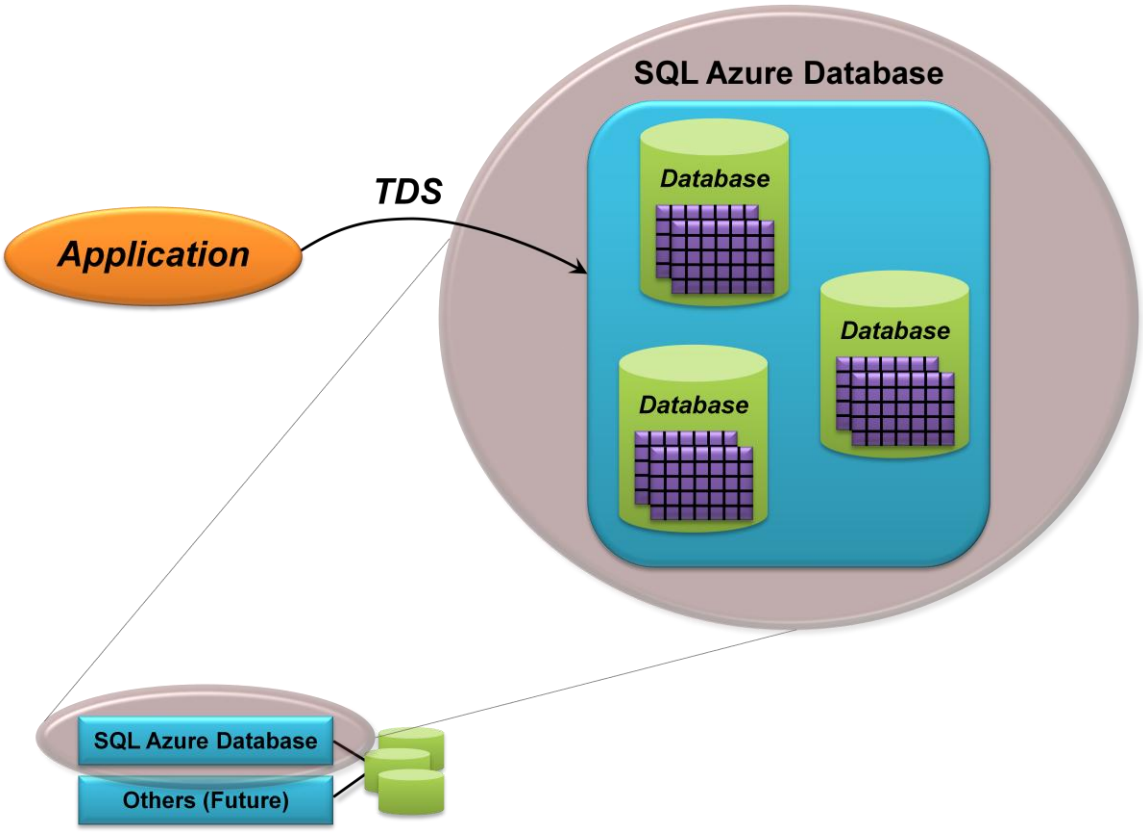

Figure 7: Applications access data in SQL Azure Database through Microsoft's TDS protocol, allowing them to use ADO.NET and other common data interfaces.

An application using SQL Azure Database might run on Windows Azure, in an enterprise's data center, on a mobile device, or somewhere else. Wherever it runs, the application accesses data via a protocol called Tabular Data Stream (TDS). This is the same protocol used to access a local SQL Server database, and so a SQL Azure Database application can use any existing SQL Server client library. This includes ADO.NET, ODBC, and PHP. And because SQL Azure Database looks like an ordinary SQL Server system, standard tools can also be used, including SQL Server Management Studio, SQL Server Integration Services, and BCP for bulk data copy.

For the most part, an application using SQL Azure Database sees a familiar SQL Server environment. A few things are omitted in the technology's first release, however, such as the SQL Common Language Runtime (CLR) and support for spatial data. (Microsoft says that both will be available in a future version.) Also, because administration is handled by Microsoft, the service doesn't expose physical administrative functions. A customer can't shut down the system, for example, or interact directly with the hardware it runs on. And as you'd expect in a shared environment, a query can run for only a limited time—no single request can take up more than a pre-defined amount of resources.

Yet while the environment looks quite standard, the service an application gets is more robust than what a single instance of SQL Server provides. As in Windows Azure storage, all data stored in SQL Azure Database is replicated three times. Also like Windows Azure storage, the service provides strong consistency: When a write returns, the data has been made persistent. The goal is to provide reliable data storage even in the face of system and network failures.

In its first release, the maximum size of a single database in SQL Azure Database is 10 gigabytes. An application whose data is within this limit can use just one database, while an application with more data will need to create multiple databases. Figure 8 illustrates this idea.

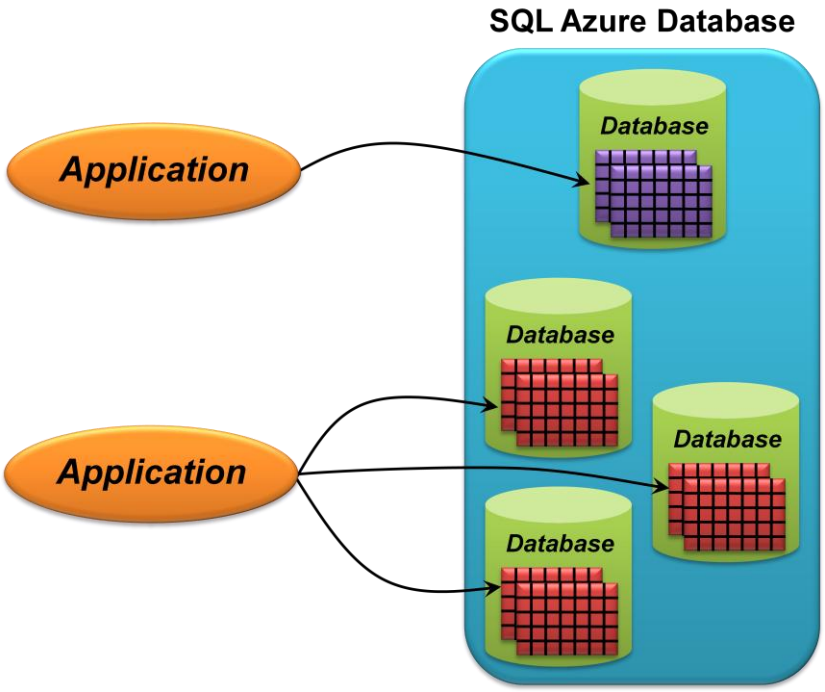

Figure 8: An application can use a single database or multiple databases.

With a single database, an application sees one set of data, and so SQL queries can be used as usual across all of this data. With multiple databases, however, the application must divide its data among them. Information about customers whose names start with "A" might be in one database, for example, customers whose names start with "B" in another, and so on. While each database exposes the usual relational interface, the application can no longer issue a single SQL query that accesses all data in all databases. Instead, applications that work with multiple databases will need to be aware of how that data is divided.

In some cases, even applications with smaller amounts of data might choose to use multiple databases. This approach allows parallel queries, for example, and so it can provide better performance. Similarly, a multi-tenant application that provides services to different customer organizations might choose to use multiple databases, perhaps assigning one to each organization.

Whether an application needs multiple databases or just one, SQL Azure Database can help developers address a range of scenarios. Whatever problem is being solved, the technology's fundamental goal remains the same: to provide a familiar, reliable, and low-cost cloud database for many kinds of applications.

WINDOWS AZURE PLATFORM APPFABRIC

Whether applications run in the cloud or on premises, they frequently need to connect to other applications. The goal of Windows Azure platform AppFabric is to help make these connections possible.

This section takes a closer look at Service Bus and Access Control, the two components of AppFabric.

Service Bus

Suppose you have an application running inside your organization that exposes a Web service built with Windows Communication Foundation (WCF). Suppose further that you'd like to connect this service through the Internet to software running outside your organization. This client software might be running either on a cloud platform, such as Windows Azure, or inside another organization.

At first glance, this can seem like a simple problem. Since your application provides its functionality through Web services (either RESTful or SOAP-based), you can just make those Web services visible to the outside world. When you actually try to do this, though, some problems appear.

First, how can clients in other organizations (or even in other parts of your own organization) find endpoints they can connect to for your service? It would be nice to have some kind of registry where others could locate your application. And once they've found it, how can requests from software in other organizations get through to your service? Network address translation (NAT) is very common, so an application frequently doesn't have a fixed IP address to expose externally. And even if NAT isn't being used, how can requests get through your firewall? It's possible to open firewall ports to allow access to your application, but most network administrators frown on this.

Service Bus addresses these challenges. Figure 10 shows how.

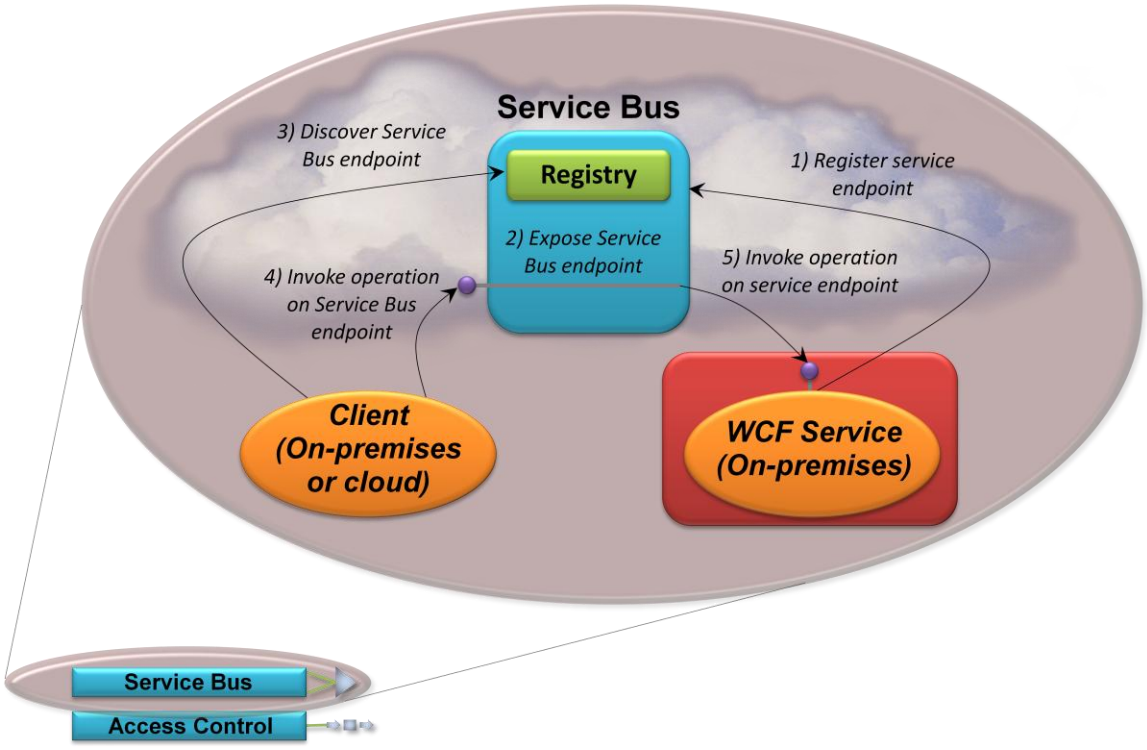

Figure 10: A WCF service can register endpoints with Service Bus, then have clients discover and use those endpoints to access the service.

To begin, your WCF service registers one or more endpoints with Service Bus (step 1). For each registered endpoint, Service Bus exposes its own corresponding endpoint (step 2). Service Bus also assigns your organization a URI root, below which you're free to create any naming hierarchy you like. This allows your endpoints to be assigned specific, discoverable URIs. Your application must also open a connection with Service Bus for each endpoint it exposes. Service Bus holds this connection open, which solves two problems. First, NAT is no longer an issue, since traffic on the open connection with Service Bus will always be routed to your application. Second, because the connection was initiated from inside the firewall, there's no problem passing information back to the application via this connection—the firewall won't block this traffic.

When a client running in the cloud or on-premises at some other organization wishes to access your service, it contacts the Service Bus registry (step 3) to find the endpoint. This request uses the Atom Publishing Protocol, and it returns an AtomPub service document with references to the endpoints Service Bus exposes on behalf of your application. Once it has these, the client can invoke operations on the services exposed through these endpoints (step 4). For each request Service Bus receives, it invokes the corresponding operation in the endpoint exposed by your WCF service (step 5). And although it's not shown in the figure, Service Bus establishes a direct connection between an application and its client whenever possible, making their communication more efficient.

Along with making communication easier, Service Bus can also improve security. Because clients now see only an IP address provided by Service Bus, there's no need to expose any IP addresses from within your organization. This effectively makes your application anonymous, since the outside world can't see its IP address. Service Bus acts as an external DMZ, providing a layer of indirection to deter attackers.

While an application that exposes its services via Service Bus is typically implemented using WCF, clients can be built with WCF or other technologies, such as Java. However they're created, clients can make requests via TCP, HTTP, or HTTPS. Applications and their clients are also free to use their own security mechanisms, such as encryption, to shield their communication from attackers and from Service Bus itself.

Exposing applications to the outside world isn't as simple as it might seem. The intent of Service Bus is to make implementing these connections as straightforward as possible.

Access Control

Working with identity is a fundamental part of most distributed applications. Based on a user's identity information, an application might make decisions about what that user is allowed to do. Applications can also have identities, and in some situations, an application receiving a request might decide whether that request is allowed based on the identity of the client application rather than on one of that application's users.

Access Control addresses exactly this problem: conveying application identity information via REST-based services. (Microsoft says that future releases will broaden this service's role, adding support for conveying user identity information, SOAP-based services, and more.) Figure 11 illustrates how it does this.

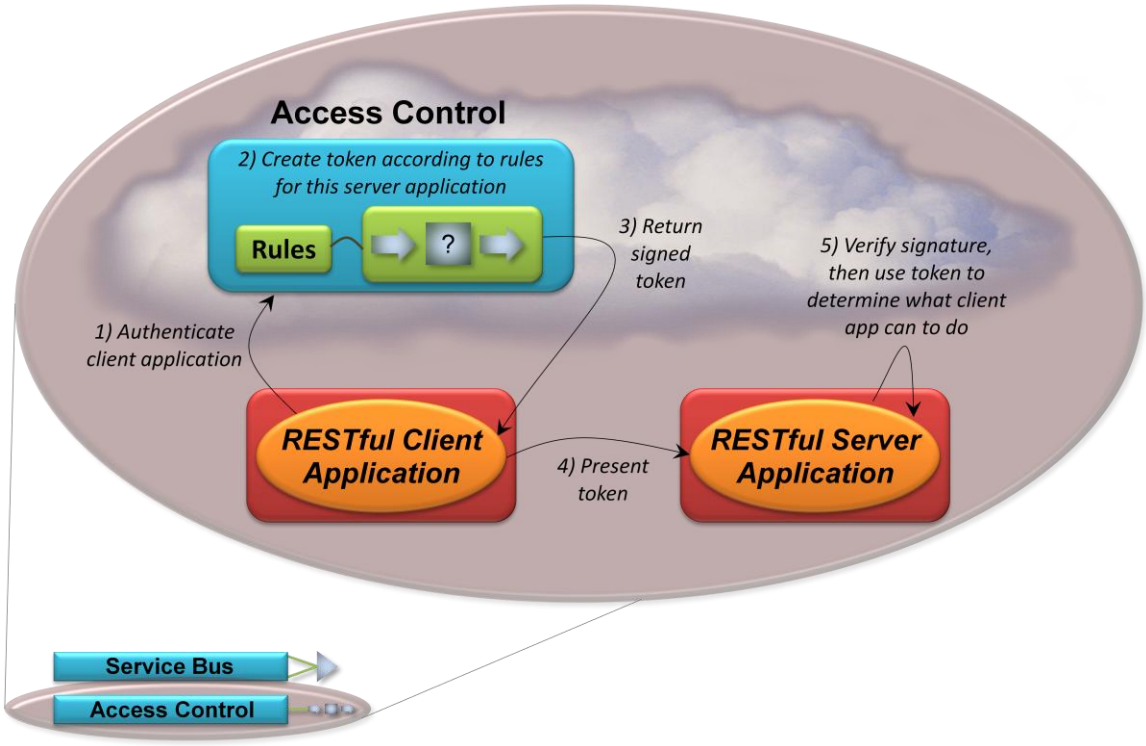

Figure 11: The Access Control service helps RESTful applications authenticate and receive identity information about their client applications.

To communicate with a particular server application, the client must first get a *token* that contains identity information about this client. This information is expressed as one or more *claims*, each of which describes the client application in some way. For example, one claim might indicate the organization that owns this client application, while another identifies a specific application type within this organization.

This token is issued by the Access Control server, and to get it, the client application must first authenticate itself, as Figure 11 shows (step 1). Using an encrypted HTTPS channel, the application sends one of three things to do this:

- A 32-byte key issued to this client application by the server application. The server can create this key itself or it can ask the Access Control service to create it. Wherever it comes from, the key must be distributed to the client application through some out-of-band mechanism. The key must also be registered with Access Control, which makes it possible for this service to authenticate the client application, much like a password. If this option is used, an administrator defines claims about the client application in Access Control, then associates them with the application's key.

- A token containing claims about this client application. This token is signed using the 32-byte key issued by the server, letting Access Control authenticate the client by checking this signature. The token's format is simple: It's just a set of human readable name/value pairs.

- A token described using the Security Assertion Markup Language (SAML). In this case, the token is not signed using the server-provided 32-byte key. Instead, the organization running the client application must establish a federation relationship with the Access Control service. This option is especially

useful for organizations with an established claims-based identity infrastructure based on a technology such as Active Directory Federation Services (AD FS) 2.0. In this case, the AD FS 2.0 server inside the application's Windows domain can issue it a SAML token, and the application can use this token to identify itself to Access Control.

Once the client application has authenticated itself, the Access Control service creates another token containing identity information for this client (step 2). This token uses the same format as the second option above: It's a set of human readable name/value pairs, each of which expresses some claim about this application.

The server application for which this token is intended can define rules about how the token will be created. If the client provides just the 32-bit key, the rules are quite simple: the Access Control service can only send a pre-defined set of claims in the new token. If the client provides either a name/value token or a SAML token, however, those rules can be more complex. For example, suppose different purchasing applications in different client organizations use different strings to express this aspect of their identity. One might send the string "Purchasing Application", another the string "PurchApp", and a third the numeric code "08041963". Using rules defined by the administrator of the server applications, Access Control can convert all of these into the string "Purchasing". This kind of *claims transformation* makes life easier for the server application. It can now look for this single value rather than knowing about all of the variations in its clients.

Once the new token is created, it's sent back to the client application (step 3). As Figure 11 shows, this new token is signed using a key provided by the server application. The client sends this signed token to the server (step 4), which then validates the token's signature and uses the claims it contains (step 5).

Working with identity is an important part of nearly every distributed application. The goal of Access Control's first release is to make this easier for application-to-application interactions via REST. Going forward, expect the functions of this service to include a wider range of cloud-based identity services.

PUTTING THE PIECES TOGETHER: AN EXAMPLE

The components of the Windows Azure platform can be used in many different ways. To get a feeling for the possibilities, it's useful to look at a simple scenario that shows one way to put the pieces together.

Suppose a software company builds a cloud-based application that supports creative professionals. This application lets its users manage projects, store and access digital assets such as videos and images, and more. The application's users might be both independent designers and employees in larger organizations. Figure 12 shows how this application could use the Windows Azure platform.

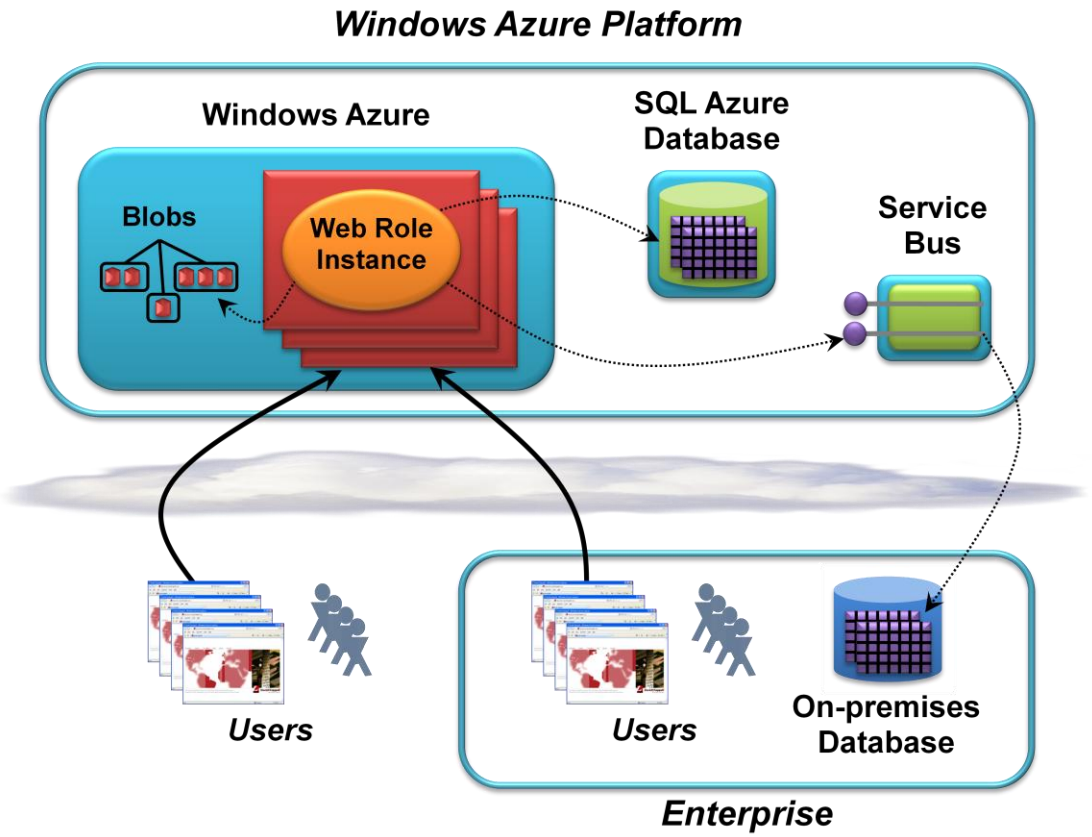

Figure 12: A Web application running on Windows Azure might use both blobs and SQL Azure for storage, then rely on Service Bus to access an on-premises database.

The application's logic is implemented using Web roles, and so users access it via Web browsers. Those users might be employees of some enterprise or independent creatives working on their own. For both, the application stores their digital assets in Windows Azure Storage blobs.

The application also keeps track of project information, such as the client name and hours billed. Customers have a choice: They can let the application store this data in the cloud using SQL Azure Database, or they can maintain the data in a database within their own organization. (Some firms might wish to keep this data inside their firewall for regulatory reasons, for example, or to conform to their country's data privacy laws.) To support the second option, the application uses Service Bus to connect to an on-premises database.

As this simple example shows, the Windows Azure platform is designed to provide a cohesive set of services for cloud applications. Using it effectively means understanding both the components themselves and how they can be used together.

LOOKING AHEAD

Microsoft has announced a number of updates that it plans to add to the Windows Azure platform in the near future. They include the following:

- Windows Azure will get broader support for running existing applications. Also, a technology code-named “Sydney” will let Windows Azure instances connect to an on-premises environment using IPsec.
- SQL Azure will provide data synchronization services based on the Microsoft Sync Framework that will allow synchronizing data between SQL Azure Database and on-premises databases.
- Functionality from Windows Server AppFabric, the on-premises version, will begin to appear in Windows Azure platform AppFabric.
- Microsoft Codename “Dallas”, built on Windows Azure and SQL Azure, will provide a cloud-based marketplace for information. Through RESTful services, developers will be able to subscribe to privately owned data and access public domain data, such as US census information and United Nations statistical data.

All of these changes target the same goal: making the Windows Azure platform useful in a broader range of scenarios.

CONCLUSIONS

The truth is evident: Cloud computing is here. For developers, taking advantage of the cloud means using cloud platforms. In the Windows Azure platform, Microsoft presents a range of options addressing a variety of needs:

- Windows Azure provides a Windows-based computing and storage environment in the cloud.
- SQL Azure provides a cloud DBMS with SQL Azure Database, with more cloud-based data services planned.
- Windows Azure platform AppFabric offers cloud-based infrastructure for connecting cloud and on-premises applications.

These approaches address a variety of requirements, and not every developer will use all of them. Yet whether you work for an ISV or an enterprise, some cloud platform services are likely to be useful for applications your organization creates. A new world is unfolding; prepare to be part of it.

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.