



DavidChappell
& Associates

INTRODUCING THE WINDOWS AZURE PLATFORM

AN EARLY LOOK AT WINDOWS AZURE, SQL AZURE, AND
.NET SERVICES

DAVID CHAPPELL

AUGUST 2009

SPONSORED BY MICROSOFT CORPORATION

CONTENTS

An Overview of the Windows Azure Platform	3
Windows Azure.....	4
SQL Azure.....	6
.NET Services.....	8
A Closer Look at the Technologies	9
Windows Azure.....	9
<i>Running Applications</i>	9
<i>Accessing Data</i>	11
SQL Azure.....	13
<i>SQL Azure Database</i>	13
<i>“Huron” Data Sync</i>	16
.NET Services.....	17
<i>Access Control Service</i>	17
<i>Service Bus</i>	19
Conclusions	21
About the Author	21

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, their code and data might both 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. Applications that run on desktops and mobile devices can use services in the cloud to synchronize information across many systems or in other ways. 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 the .NET Framework, SQL Server, and more. To let applications exploit the cloud, cloud application platforms must also exist. And because there are a variety of ways for applications to use cloud services, different kinds of cloud platforms are useful in different situations.

Microsoft's Windows Azure platform (originally known as the Azure Services 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 applications running on local systems.

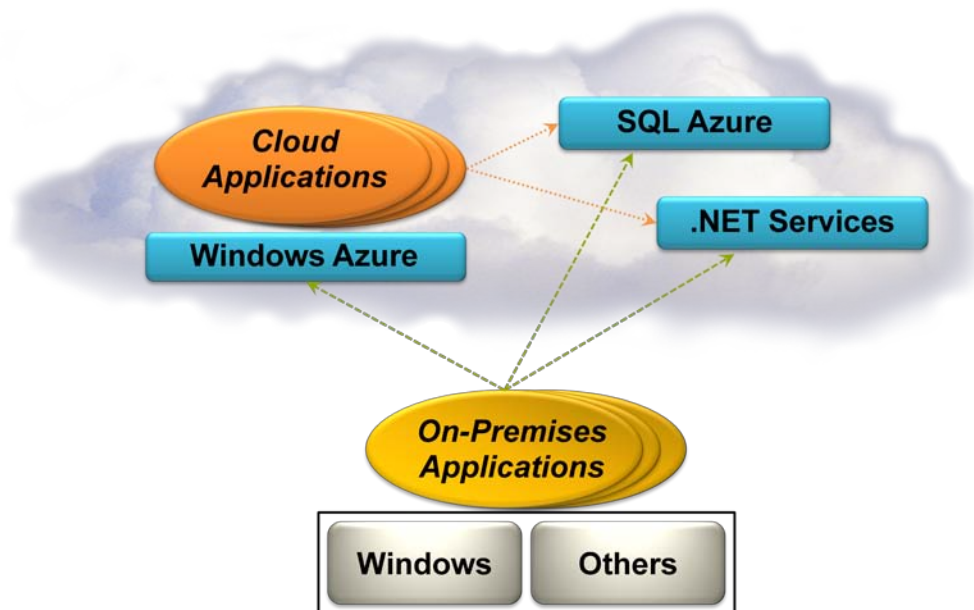


Figure 1: The Windows Azure platform supports applications, data, and infrastructure in the cloud.

The components of the Windows Azure platform include:

- 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.
- *.NET Services*: Offers distributed infrastructure services to cloud-based and local applications.

Each component of the Windows Azure platform has its own role to play. This overview describes all three of its members, first at a high level, then in a bit more detail. While none of them are yet final—details and more might change before their initial release—it’s not too early to start understanding this new set of platform technologies.

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.

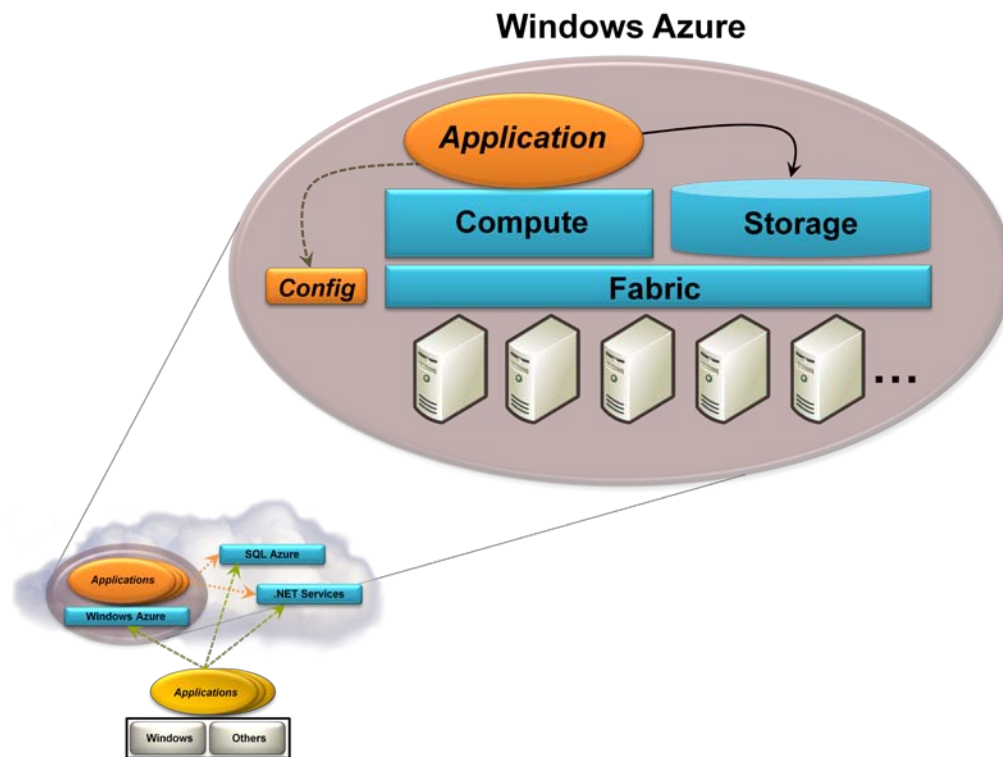


Figure 2: Windows Azure provides Windows-based compute and storage services for cloud applications.

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. For the initial availability of this service, a Community Technology Preview (CTP) made public in the fall of 2008, Microsoft allowed Windows Azure to run only applications built on the .NET Framework. Today, however, Windows Azure also supports unmanaged code, letting developers run applications that aren’t built on the .NET Framework. In either case, those applications are written in ordinary Windows languages—C#, Visual Basic, C++, and others—using Visual Studio 2008 or another development tool. Developers can create

Web applications using technologies such as ASP.NET and Windows Communication Foundation (WCF), 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. The underlying data store is not Microsoft SQL Server, however. In fact, Windows Azure storage isn't a relational system, and its query language isn't SQL. Because it's primarily designed to support applications built on Windows Azure, it provides simpler, more scalable kinds of storage. Accordingly, it 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 Windows Azure applications that do need traditional relational storage, however, the Windows Azure platform provides SQL Azure Database, described later.)

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 the information in this file manually or programmatically, an application's owner can control various aspects of its behavior, such as setting the number of instances that Windows Azure should run. The Windows Azure fabric 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 general platform that can be used in various scenarios. Here are a few examples, all based on what the CTP version allows:

- 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 users and investors. The company can also start small, incurring low costs while its application has only a few users. If their application catches on and usage increases, Windows Azure can scale the application as needed.
- An 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 a set of cloud-based services for storing and working with many kinds of information. While Microsoft says that SQL Azure will eventually include a range of data-oriented capabilities, including reporting, data analytics, and others, the first SQL Azure components to appear are SQL Azure Database and "Huron" Data Sync. Figure 3 illustrates this.

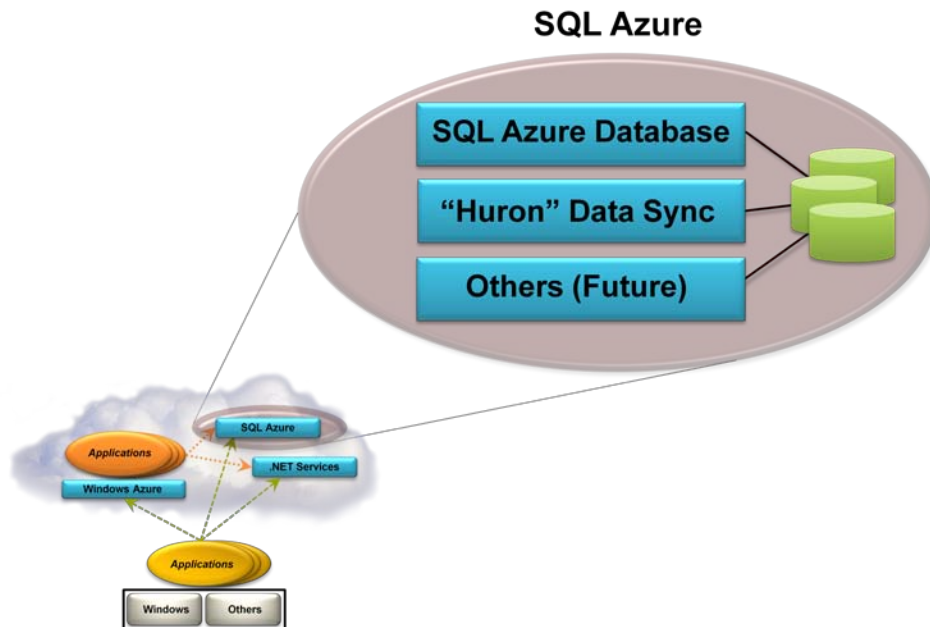


Figure 3: SQL Azure provides data-oriented facilities in the cloud.

SQL Azure Database (formerly known as SQL Data Services) provides a database management system (DBMS) in the cloud. 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.

Unlike the Windows Azure storage service, SQL Azure Database is built on Microsoft SQL Server. Still, in its original 2008 CTP release, SQL Azure Database didn't expose a traditional relational view of data. Based on customer feedback, Microsoft decided to change this. Going forward, SQL Azure Database will support relational data, offering 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 this cloud-based information.

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.

The second SQL Azure component announced so far is "Huron" Data Sync. Built on the Microsoft Sync Framework and SQL Azure Database, this technology synchronizes relational data across various on-premises DBMSs. The owners of that data can determine what should be synchronized, how conflicts should be handled, and more.

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 big 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 to both its dealer network and directly to customers. Putting this data in SQL Azure Database would allow it to be accessed by applications running at the dealers and by a customer-facing Web application run by the manufacturer itself.
- An organization with a customer database replicated across different geographies might use "Huron" Data Sync to keep these replicas in sync. Perhaps each geography needs its own copy of the data for performance or to ensure availability or for some other reason. Automatic synchronization can make this necessary distribution significantly less painful.

Whether it's for supporting a Windows Azure application, making data more accessible, keeping that data synchronized, 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.

.NET SERVICES

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 that can be used by either on-premises applications or cloud applications. Filling this gap is the goal of .NET Services.

Originally known as BizTalk Services, the functions provided by .NET Services address common infrastructure challenges in creating distributed applications. Figure 4 shows its components.

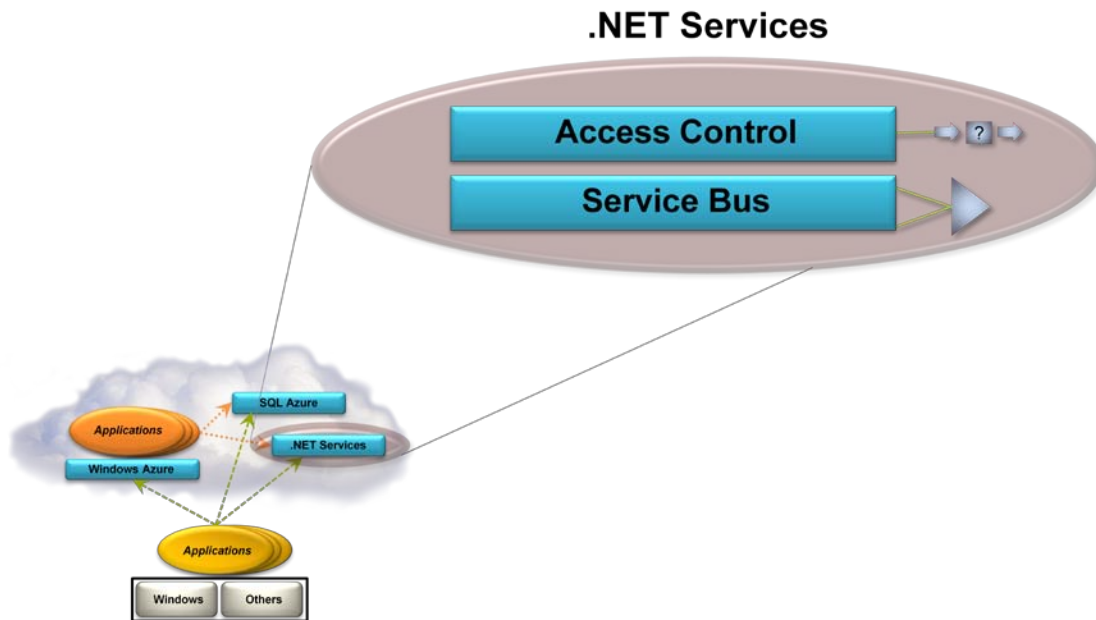


Figure 4: .NET Services provides cloud-based infrastructure that can be used by both cloud and on-premises applications.

The components of .NET Services are:

- Access Control: An increasingly common approach to identity is to have each user supply an application with a *token* containing some set of *claims*. The application can then decide what this user is allowed to do based on these claims. Doing this effectively across companies requires *identity federation*, which lets claims created in one identity scope be accepted in another. It might also require *claims transformation*, modifying claims when they're passed between identity scopes. The Access Control service provides a cloud-based implementation of both.
- Service Bus: Exposing an application's services on the Internet is harder than most people think. The goal of Service Bus is to make this simpler by letting an application expose Web services 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.

Here are some examples of how .NET Services might be used:

- An ISV that provides an application used by customers in many different organizations might use the Access Control service to simplify the application's development and operation. For example, this .NET Services component could translate the diverse claims used in the various customer organizations, each of which might use a different identity technology internally, into a consistent set that the ISV's application could use. Doing this also allows offloading the mechanics of identity federation onto the cloud-based Access Control service, freeing the ISV from running its own on-premises federation software.
- 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. Since doing this doesn't require opening new ports in the organization's firewall, it reduces the risk of exposing the application. The organization might also use the Access Control service, which is designed to work with Service Bus, to rationalize identity information sent to the application by these partners.

As with Windows Azure, a browser-accessible portal is provided to let customers sign up for .NET Services using a Windows Live ID. Microsoft's goal with .NET Services is clear: providing useful cloud-based infrastructure for distributed applications.

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. How these two things are managed is also important, and so this description looks at this part of the story as well.

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 virtual machine (VM). These VMs run 64-bit Windows Server 2008, and they're provided by a hypervisor that's specifically designed for use in the cloud.

Yet a developer doesn't supply his own VM image for Windows Azure to run, nor does he need to worry about maintaining a copy of the Windows operating system. Instead, the CTP version lets a developer create applications using *Web role* instances and/or *Worker role* instances. Figure 5 shows how this looks.

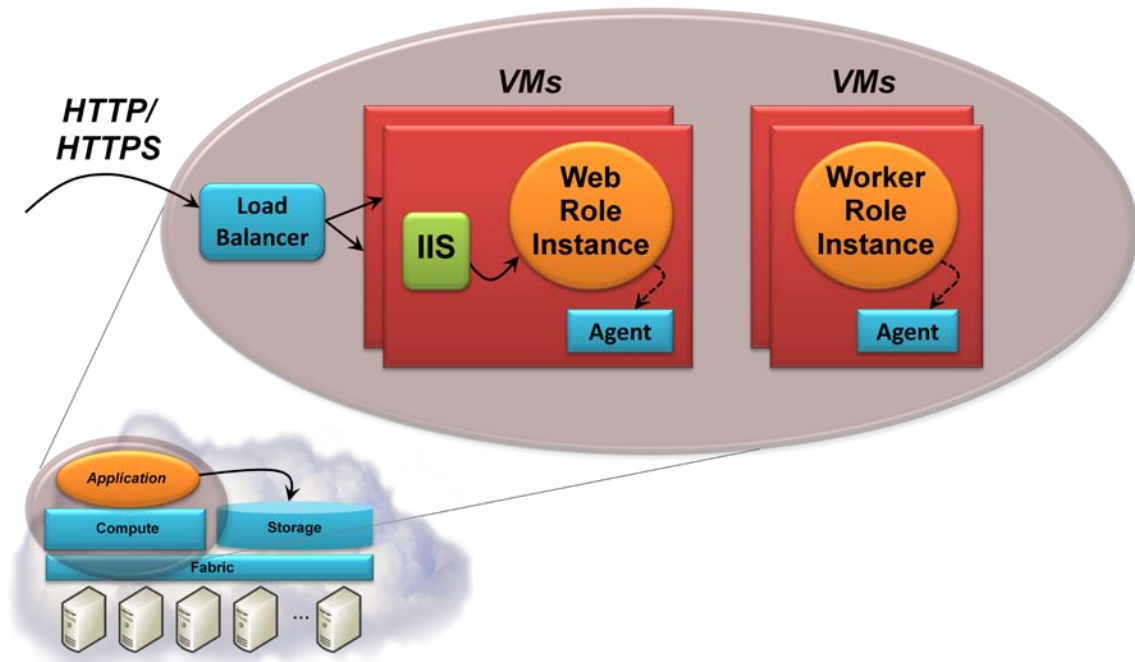


Figure 5: In the CTP version, Windows Azure applications can consist of Web role instances and Worker role instances, with each instance running in its own virtual machine.

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.

A Worker role instance, by contrast, cannot accept requests directly from the outside world—it's not allowed to have any incoming network connections, and IIS isn't running in its VM. Instead, it typically gets its input via a queue in Windows Azure storage. The messages in this queue might come from a Web role instance, an on-premises application, or something else. Wherever its input comes from, a Worker role instance can send output to another queue or to the outside world—outgoing network connections are allowed. And unlike a Web role instance, which is created to handle incoming HTTP requests, a Worker role instance is a batch job. Befitting this generality, a Worker role can be implemented using any Windows technology with a `main()` method.

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 write to a Windows Azure-maintained log, send alerts to its owner via the Windows Azure fabric, and more.

While this might change over time, Windows Azure's initial CTP release maintains a one-to-one relationship between a VM and a physical processor core. Because of this, the performance of each application can be guaranteed—each Web role instance and Worker role instance has its own dedicated processor core. To increase an application's performance, its 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 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 one thing, access to Windows Azure storage uses ADO.NET Data Services, a relatively new technology that isn't yet ubiquitous in on-premises applications. (A Windows Azure application can also use standard ADO.NET to access the relational storage provided by SQL Azure Database, however, which makes it easier to move an existing application to this cloud platform.) Also, Worker role instances typically rely on queues in Windows Azure storage for their input, an abstraction that's not available in on-premises Windows environments. In the main, however, the world an application sees running on Windows Azure is much like what it sees on any other Windows Server 2008 system.

For developers, building a Windows Azure application in the CTP version looks much like building a traditional Windows application. Microsoft provides Visual Studio 2008 project templates for creating Windows Azure Web roles, Worker roles, and combinations of the two. Developers are free to use any Windows programming language (although it's fair to say that Microsoft's initial focus for Windows Azure has been on C#). 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 via the Windows Azure agent.

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 Windows Azure will forward that alert via email, instant messaging, or some other mechanism to its specified 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.

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.

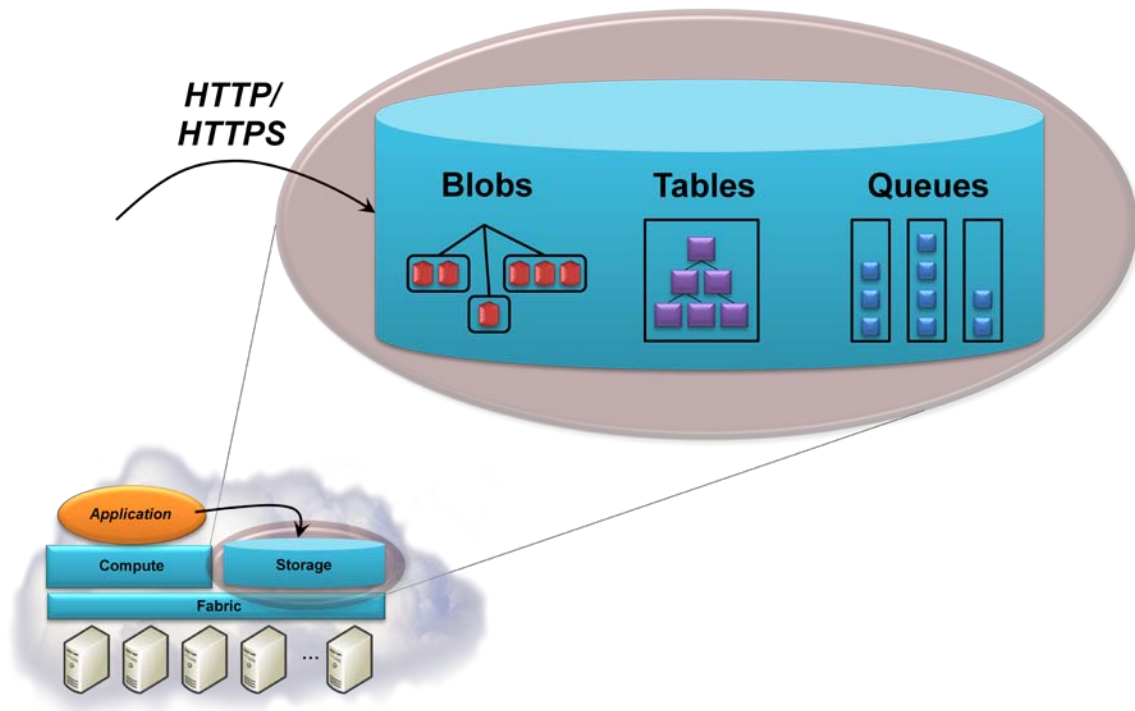


Figure 6: Windows Azure allows storing data in blobs, tables, and queues, all accessed in a RESTful style via HTTP or HTTPS.

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—up to 50 gigabytes 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.

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 quite different purpose. The primary role 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 also use ADO.NET Data Services and LINQ, but access to Windows Azure storage from, say, a Java application can just use standard REST. For example, a blob can be read with an HTTP GET against a URI formatted like this:

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

<StorageAccount> is an identifier assigned when a new storage account is created, and it uniquely identifies the blobs, tables, and queues created using this account. <Container> and <BlobName> are just the names of the container and blob that this request is accessing.

Similarly, 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.

Even queues can be accessed by both Windows Azure applications and external applications by issuing an HTTP GET against a URI formatted like this:

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

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

SQL Azure is an umbrella name for a group of cloud-based technologies for working with relational and other types of data. The first members of this family to appear are SQL Azure Database and "Huron" Data Sync. This section takes a closer look at both.

SQL Azure Database

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 is to provide all of these benefits. Figure 7 shows a simple view of this technology.

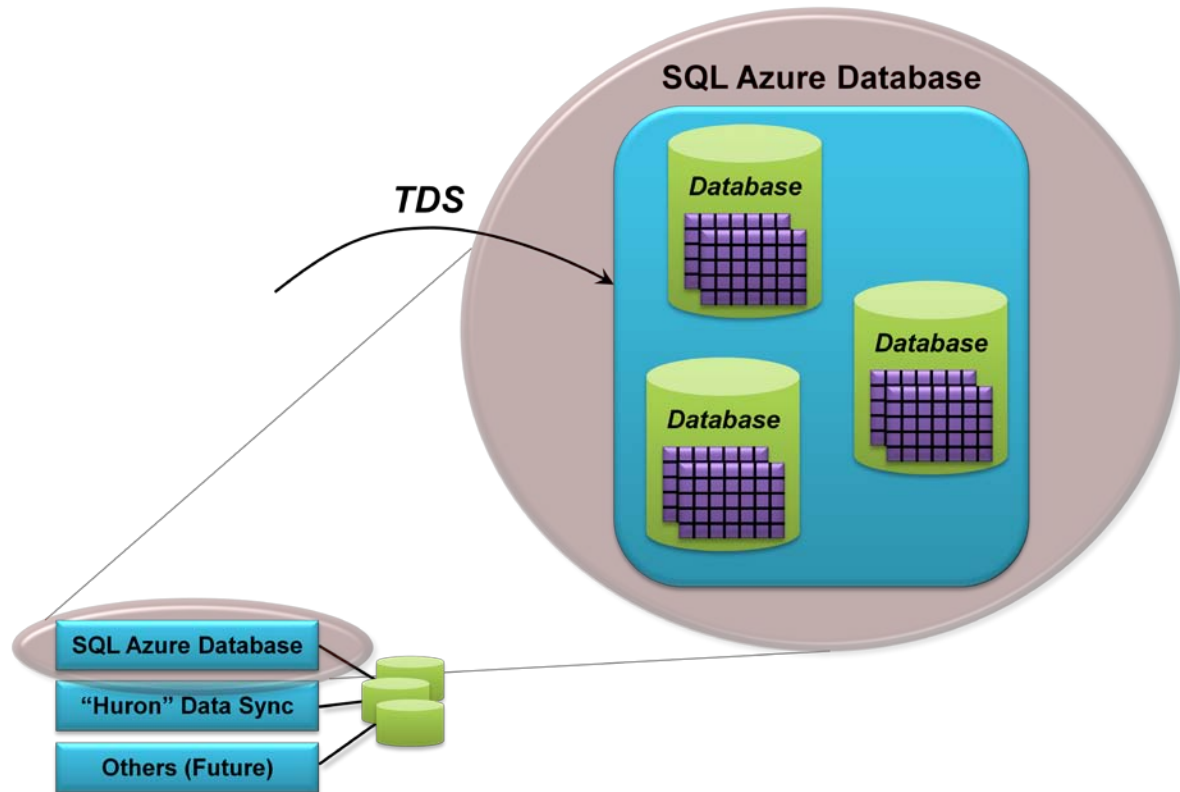


Figure 7: Applications access data in SQL Azure Database through Microsoft’s TDS protocol, allowing them to use ADO.NET and other common data interfaces.

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. The most important of these is probably ADO.NET, but ODBC and others can also be used.

For the most part, an application using SQL Azure Database sees a familiar SQL Server environment. A few things are omitted, however, such as the SQL Common Language Runtime (CLR) and support for spatial data. 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.) 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 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, all copies have been written. The goal is to provide reliable data storage even in the face of system and network failures.

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.

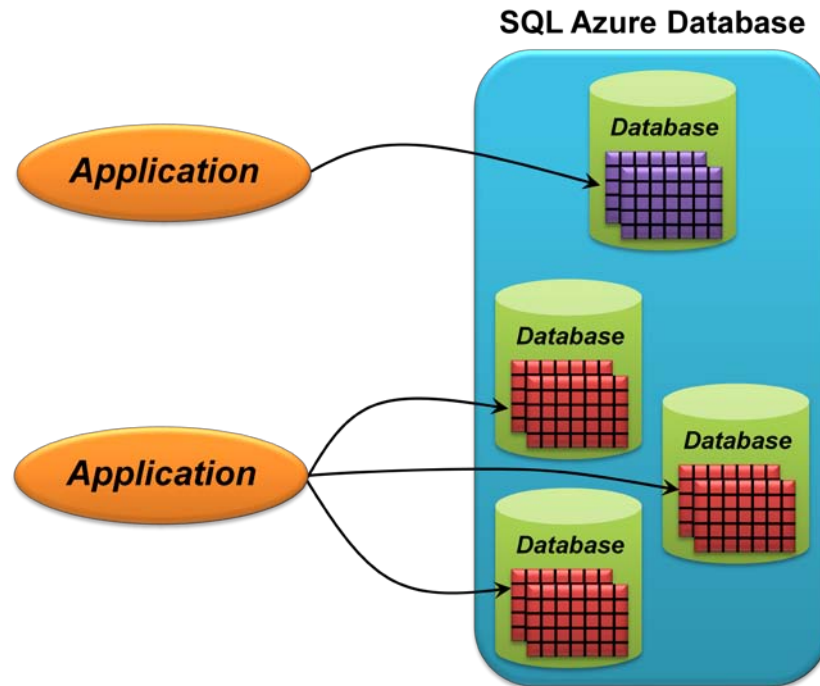


Figure 8: An application can use a single database or multiple databases.

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 in some situations. Similarly, a multi-tenant application that provides services to different 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 application 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 all kinds of applications.

“Huron” Data Sync

Ideally, data is kept in just one place. Realistically, though, this often isn’t possible. Many organizations have copies of the same data spread across different databases, often in different geographic locations. Keeping that data in sync is challenging but necessary.

“Huron” Data Sync addresses this problem. Built on the Microsoft Sync Framework and SQL Azure Database, it can synchronize relational data in multiple databases. Figure 9 shows the basics of this technology.

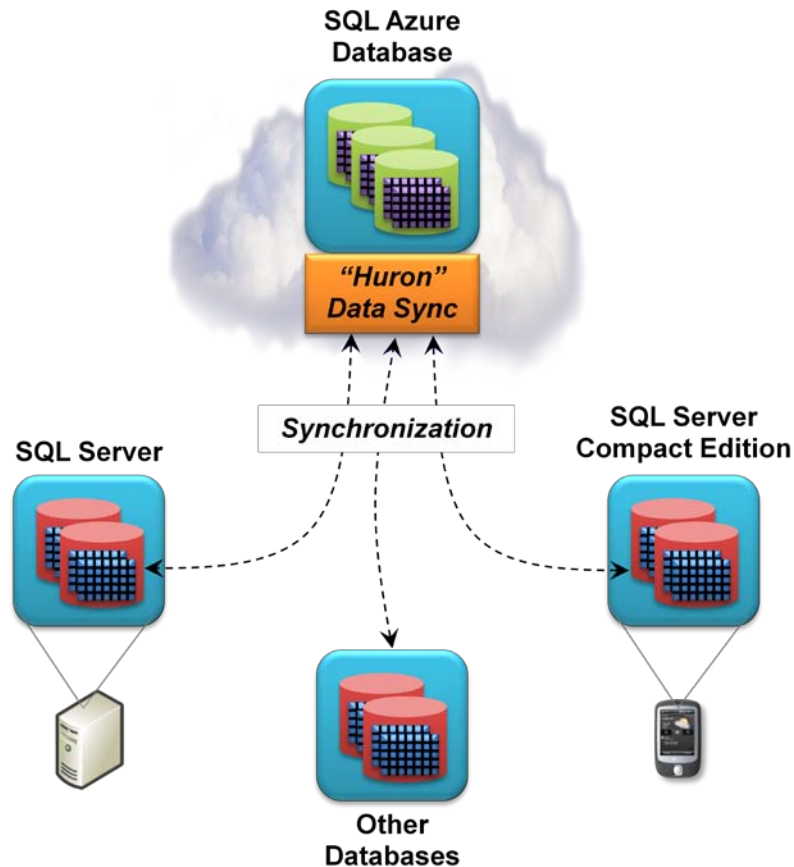


Figure 9: "Huron" Data Sync uses the Microsoft Sync Framework to synchronize data across SQL Azure Database and on-premises databases.

“Huron” Data Sync will initially support SQL Server and SQL Server Compact Edition. The technology also includes an SDK, however, allowing others to add support for more DBMSs. Whatever database technologies are used, “Huron” Data Sync works in the same way: Data changes are synchronized first to SQL Azure Database, then to the DBMSs being synchronized. The technology provides a graphical interface that lets users define which data should be synchronized across which databases.

Synchronization is multi-master, which means that changes can be made to any of the copies. The user who sets up the synchronization can also define how conflicts should be handled. Options include making the last write win, requiring that changes made to a specific database win, and more.

“Huron” Data Sync, working with SQL Azure Database, addresses an important problem in many organizations. As the SQL Azure family expands, expect to see more cloud-based solutions to common data-oriented problems.

.NET SERVICES

Storing data in the cloud is useful, but so is providing cloud-based infrastructure services. These services can be used by either on-premises or cloud-based applications, and they can address problems that can't be solved as well in any other way. This section takes a closer look at Microsoft's offerings in this area: the .NET Access Control Service and .NET Service Bus, known collectively as .NET Services.

Access Control Service

Working with identity is a fundamental part of most distributed applications. Based on a user's identity information, an application makes decisions about what that user is allowed to do. To convey this information, applications can rely on tokens defined using the Security Assertion Markup Language (SAML). A SAML token contains claims, each of which carries some piece of information about a user. One claim might contain her name, another might indicate her role, such as manager, while a third contains her email address. Tokens are created by software known as a *security token service (STS)*, which digitally signs each one to verify its source.

Once a client (such as a Web browser) has a token for its user, it can present the token to an application. The application then uses the token's claims to decide what this user is allowed to do. There are a couple of possible problems, however:

- What if the token doesn't contain the claims this application needs? With claims-based identity, every application is free to define the set of claims that its users must present. Yet the STS that created this token might not have put into it exactly what this application requires.
- What if the application doesn't trust the STS that issued this token? An application can't accept tokens issued by just any STS. Instead, the application typically has access to a list of certificates for trusted STSs, allowing it to validate the signatures on tokens they create. Only tokens from these trusted STSs will be accepted.

Inserting another STS into the process can solve both problems. To make sure that tokens contain the right claims, this extra STS performs *claims transformation*. The STS can contain rules that define how input and output claims should be related, then use those rules to generate a new token containing the exact claims an application requires. Addressing the second problem, commonly called identity federation, requires that the application trust the new STS. It also requires establishing a trust relationship between this new STS and the one that generated the token the STS received.

Adding another STS allows claims transformation and identity federation, both useful things. But where should this STS run? It's possible to use an STS that runs inside an organization, an option that's provided by several vendors today. Yet why not run an STS in the cloud? This would make it accessible to users and applications in any organization. It also places the burden of running and managing the STS on a service provider.

This is exactly what the Access Control Service offers: It's an STS in the cloud. To see how this STS might be used, suppose an ISV provides an Internet-accessible application that can be used by people in many different organizations. While all of those organizations might be able to provide SAML tokens for their users, these tokens are unlikely to contain the exact set of claims this application needs. Figure 10 illustrates how the Access Control Service can address these challenges.

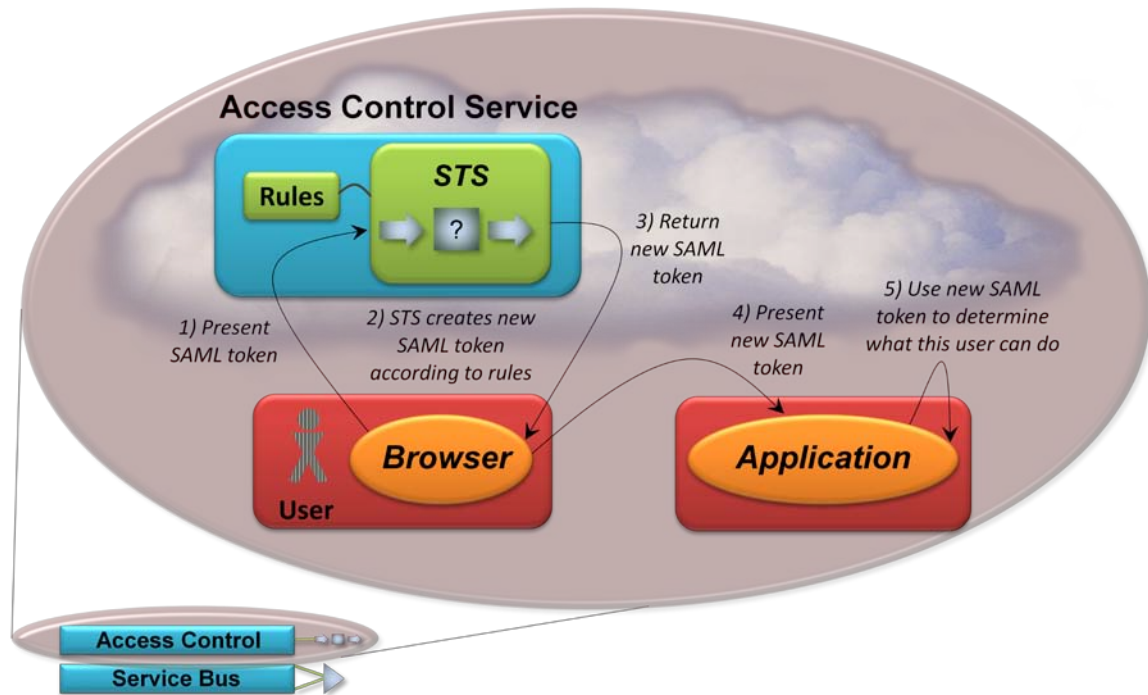


Figure 10: The Access Control Service provides rules-based claims transformation and identity federation.

First, the user's application (which in this example is a Web browser, but could also be a WCF client or something else) sends the user's SAML token to the Access Control Service (step 1). This service validates the signature on the token, verifying that it was created by an STS the service trusts. The service then creates and signs a new SAML token containing exactly the claims this application requires (step 2).

To do this, the Access Control Service's STS relies on rules defined by the owner of the application that the user is trying to access. For example, suppose the application grants specific access rights to any user who is a manager in her company. While each company might include a claim in its token indicating that a user is a manager, they'll likely all be different. One company might use the string "Manager", another the string "Supervisor", and a third an integer code. To help the application deal with this diversity, its owner could define rules in Access Control that convert all three of these claims to the common string "Decision Maker". The application's life is now simpler, since the work of claims transformation is done for it.

Once it's been created, the STS in the Access Control Service returns this new token to the client (step 3) who then passes it on to the application (step 4). The application validates the signature on the token, making sure that it really was issued by the Access Control Service STS. Note that while the Access Control Service STS must maintain a trust relationship with the STS of each customer organization, the application

itself need trust only the Access Control Service STS. Once it's certain of this token's provenance, the application can use the claims it contains to decide what this user is allowed to do (step 5).

Another way to use the Access Control Service is implied by its name: An application can effectively offload to the service decisions about what kind of access each user is allowed. For example, suppose access to a certain function of an application requires the user to present a particular claim. The rules in the Access Control Service for the application could be defined to give this claim only to users that present other required claims, such as one of the manager claims described earlier. When the application receives a user's token, it can grant or deny access based on the presence of this claim—the decision was effectively made for it by the Access Control Service. Doing this lets an administrator define access control rules in one common place, and it can also help in sharing access control rules across multiple applications.

All communication with the Access Control Service relies on standard protocols such as WS-Trust and WS-Federation. This makes the service accessible from any kind of application on any platform. And to define rules, the service provides both a browser-based GUI and a client API for programmatic access.

Claims-based identity is on its way to becoming the standard approach for distributed environments. By providing an STS in the cloud, complete with rules-based claims transformation, the Access Control Service makes this modern approach to identity more attractive.

Service Bus

Suppose you have an application running inside your organization that you'd like to expose to software in other organizations through the Internet. At first glance, this can seem like a simple problem. Assuming your application provides its functionality as 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 applications in other organizations (or even in other parts of your own organization) find endpoints they can connect to for your services? 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 application? 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 11 shows how.

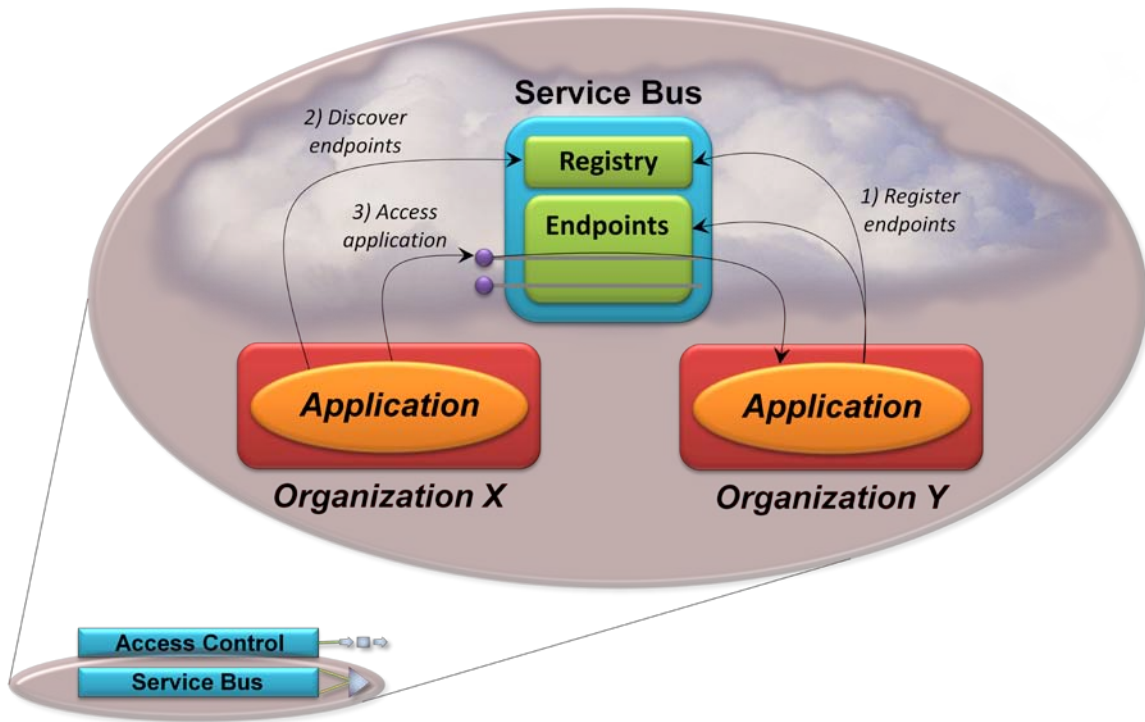


Figure 11: Service Bus allows an application to register endpoints, then have other applications discover and use those endpoints to access its services.

To begin, your application registers one or more endpoints with Service Bus (step 1), which exposes them on your behalf. Service Bus 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 an application in some other organization (or even a different part of your own) wishes to access your application, it contacts the Service Bus registry (step 2). This request uses the Atom Publishing Protocol, and it returns an AtomPub service document with references to your application's endpoints. Once it has these, it can invoke services offered through these endpoints (step 3). Each request is received by Service Bus, then passed on to your application, with responses traveling the reverse path. 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.

Service Bus also allows communication via queues. This lets a client application send messages even when the listening application isn't available; Service Bus can queue the messages for up to a week waiting for the listener to receive them. It's also possible for a client to send messages to a Service Bus queue, then have those messages received by more than one listener.

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. And finally, Service Bus is designed to be used with the Access Control Service, allowing rules-based claims transformation. In fact, Service Bus accepts only tokens issued by the Access Control Service STS.

An application that wishes to expose its services via Service Bus is typically implemented using WCF. Clients can be built with WCF or other technologies, such as Java, and they can make requests via SOAP or HTTP. 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 this useful behavior as straightforward as possible.

CONCLUSIONS

The truth is evident: Cloud computing is here. For developers, taking advantage of the cloud means using cloud platforms in some way. With the Windows Azure platform, Microsoft presents a range of platform styles 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 and data synchronization via "Huron" Data Sync, with more cloud-based data services planned.
- .NET Services offers cloud-based infrastructure for 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.