David Chappell
& Associates

INTRODUCING THE WINDOWS AZURE PLATFORM

DAVID CHAPPELL

OCTOBER 2010

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 or storing data. In the on-premises Windows world, this includes technologies such as Windows Server and SQL Server. To let applications exploit the cloud, a cloud application platform must also exist.

This is exactly what the Microsoft Windows Azure platform provides. It's a group of cloud technologies, each providing a specific set of services to application developers. Figure 1 shows its components.

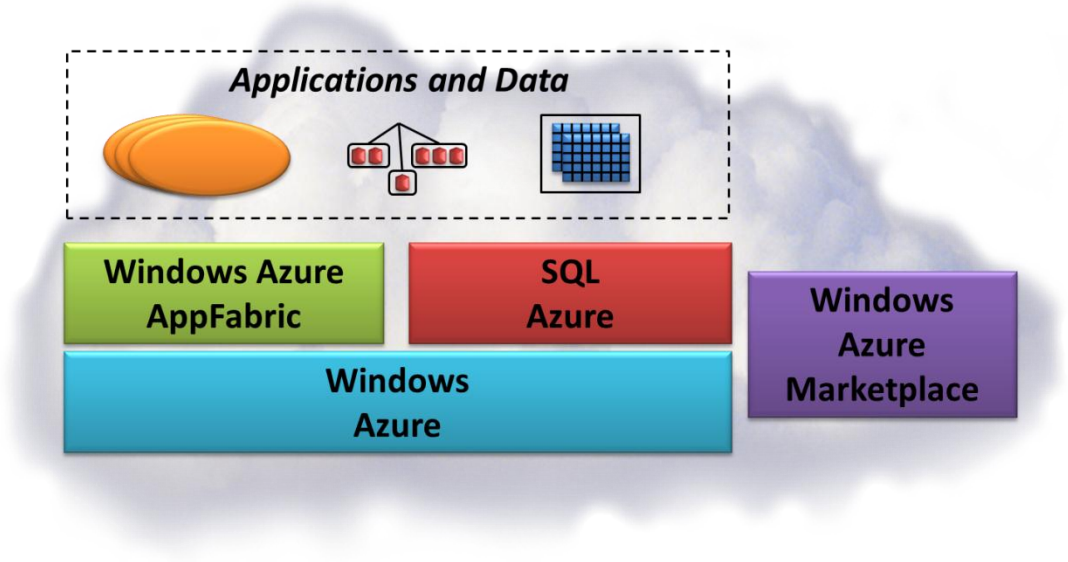

Figure 1: The Windows Azure platform supports applications, data, and infrastructure in the cloud, together with a cloud marketplace.

The Windows Azure platform today has four parts:

- Windows Azure: A Windows environment for running applications and storing data on computers in Microsoft data centers.
- SQL Azure: Relational data services in the cloud based on SQL Server.
- Windows Azure AppFabric: Cloud-based infrastructure services for applications running in the cloud or on premises.

- Windows Azure Marketplace: An online service for purchasing cloud-based data and applications.

All four of these components run in Microsoft data centers located around the world: two in North America, two in Europe, and two in Asia. Developers using the platform can control which data center runs their applications and stores their data, giving them the ability to place both closer to their users.

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

WINDOWS AZURE

At a high level, Windows Azure is simple to understand: It runs Windows applications and stores data in the cloud. Figure 2 shows its components.

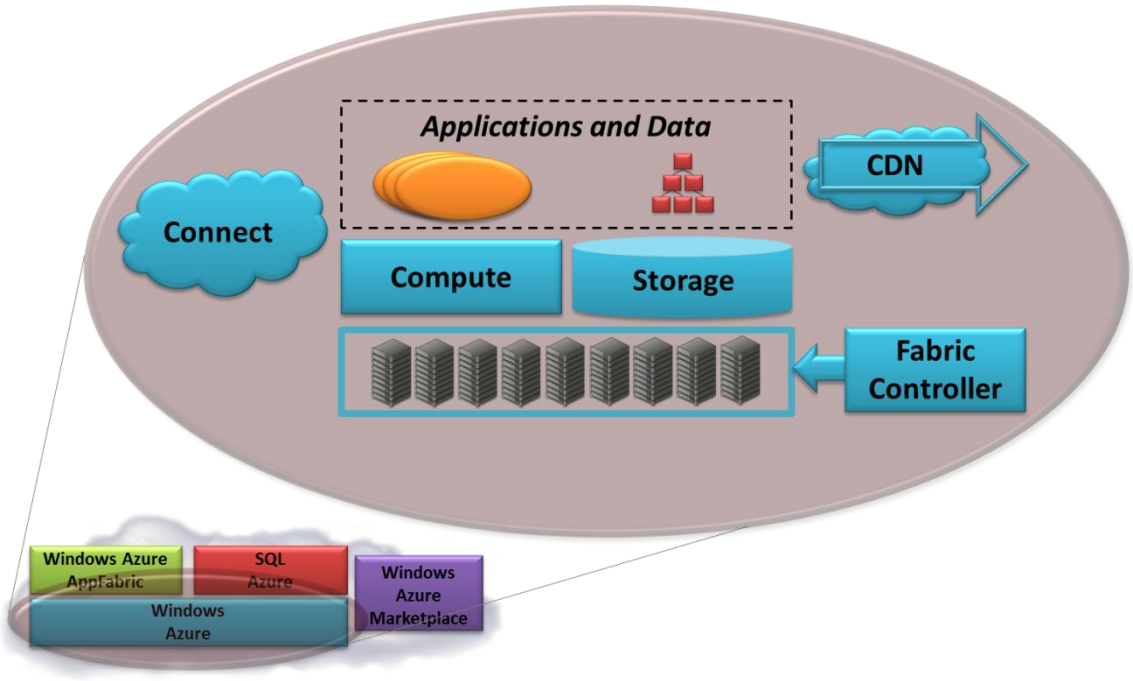

Figure 2: Windows Azure provides compute and storage services in the cloud.

The five parts of Windows Azure today are the following:

- Compute: The Windows Azure compute service runs applications on a Windows Server foundation. These applications can be created using the .NET Framework in languages such as C# and Visual Basic, or they can be built without .NET in C++, Java, and other languages. Developers can use Visual Studio or other development tools, and they're free to use technologies such as ASP.NET, Windows Communication Foundation (WCF), and PHP.

- Storage: 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. (Windows Azure applications that need traditional relational storage can also use

SQL Azure.) 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.

- Fabric controller: As the figure suggests, Windows Azure runs on a large number of machines. The fabric controller's job is to knit the machines in a single Windows Azure data center into a cohesive whole. The Windows Azure compute and storage services are then built on top of this pool of processing power.

- Content delivery network (CDN): Caching frequently accessed data closer to its users speeds up access to that data. The Windows Azure CDN can do this for blobs, maintaining cached copies at sites around the world.

- Connect: It's often useful for organizations to interact with cloud applications as if they were inside the organization's own firewall. Windows Azure Connect allows this, making it easier for, say, a Windows Azure application to access an on-premises database.

Running applications and storing 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 applications written for Windows Azure can scale better, be more reliable, and require less administration than those written using the traditional Windows Server programming model.

To create, configure, and monitor applications, Windows Azure customers can use a browser-accessible portal. A customer logs in with a Windows Live ID, then chooses whether to create a *hosting* account for running applications, a *storage* account for storing data, or both. Microsoft then charges each customer based on how much compute time, storage, and bandwidth that customer uses. How an application charges its own customers—if it charges them at all—is entirely up to the people who create that application.

Windows Azure is a general platform that can be used in a broad set of scenarios. Here are a few examples:

- 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 business logic—the thing that makes them money—rather than spending time on infrastructure.

- An enterprise creating an application for its customers or employees 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 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.

- 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.

These three examples illustrate the kinds of things organizations might do with Windows Azure, but they're not an exhaustive list. As interest in cloud computing continues to grow, expect to see a variety of applications created for this cloud platform.

SQL AZURE

Along with running applications, another attractive way to use the cloud is for storing data. SQL Azure addresses this area, offering cloud-based services for relational data. As Figure 3 shows, SQL Azure today includes three components.

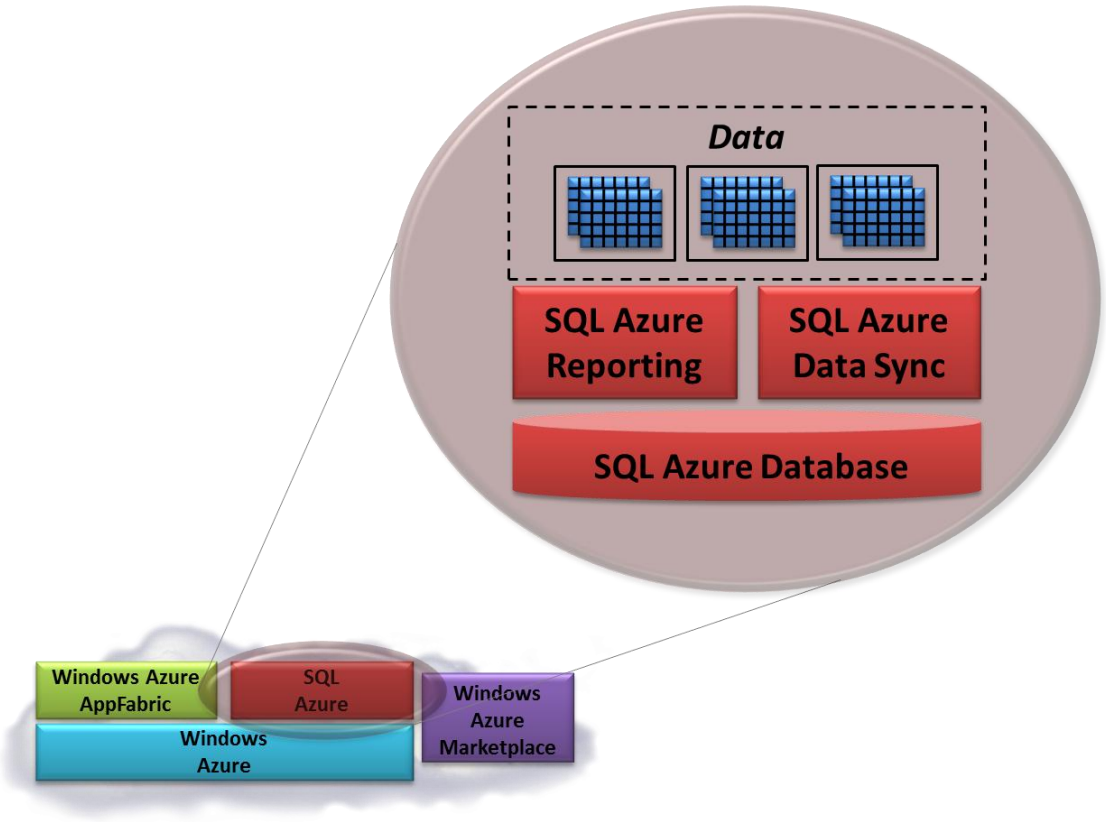

Figure 3: SQL Azure provides relational database services in the cloud.

The components of SQL Azure today are the following:

- SQL Azure Database provides a cloud-based database management system (DBMS). This technology lets on-premises and cloud applications store relational 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 Reporting is a version of SQL Server Reporting Services (SSRS) that runs in the cloud. Intended primarily for use with SQL Azure Database, it allows creating and publishing standard SSRS reports on cloud data.

- SQL Azure Data Sync allows synchronizing data between SQL Azure Database and on-premises SQL Server databases. It can also be used to synchronize data across different SQL Azure databases in different Microsoft data centers.

SQL Azure is built on Microsoft SQL Server. As with SQL Server, developers can create indexes and views, use stored procedures, define triggers, and more. Applications can access SQL Azure data using Entity Framework, 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. Customers can also use on-premises software such as SQL Server Analysis Services to work with their cloud-based data.

While applications can use SQL Azure 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 customer can focus on their data; Microsoft handles the operational details. And as with other components of this cloud platform, customers use the common Windows Azure platform portal to access its services.

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

- A Windows Azure application can store its data in SQL Azure. 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 rely on SQL Azure to work with data in this familiar way. For example, a SaaS application built on Windows Azure might create a separate SQL Azure database for each customer, providing an intrinsically multi-tenant design. And 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. 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. It can also create reports on this data using either SQL Azure Reporting or SSRS on-premises. If the organization wishes to maintain an on-premises copy of the data as well, it can use SQL Azure Data Sync to synchronize the cloud and on-premises databases.

- Suppose a manufacturer wishes to make product information available both to its dealer network and directly to customers. Putting this data in SQL Azure 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. The goal of SQL Azure is to provide these services in a familiar, usable way for cloud and on-premises applications.

WINDOWS AZURE APPFABRIC

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

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

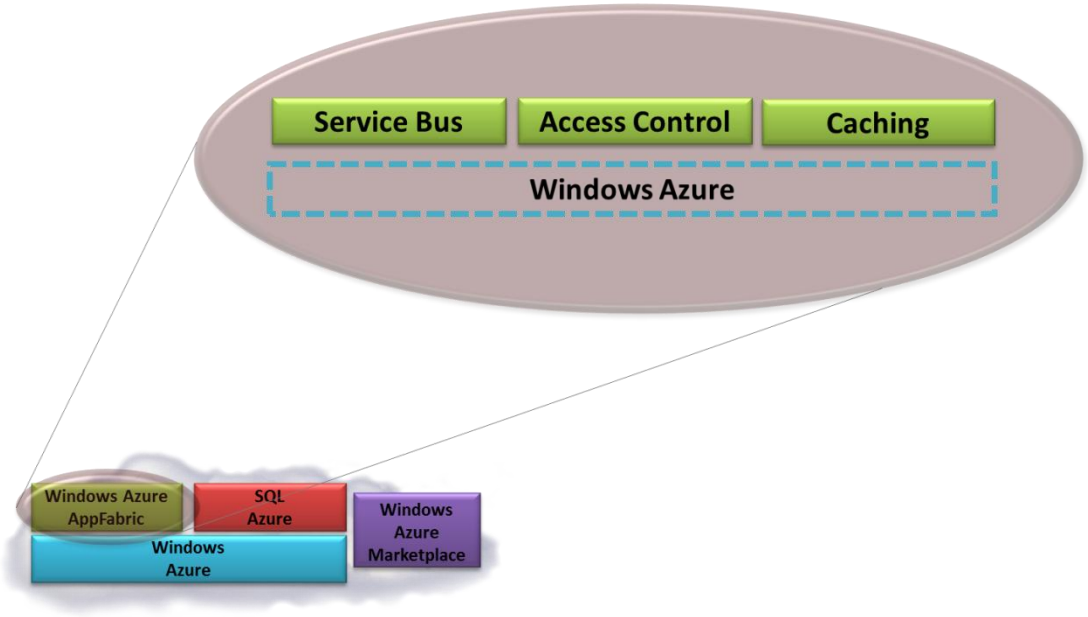

Figure 4: Windows Azure AppFabric provides Windows Azure-based infrastructure services that can be used by both cloud and on-premises applications.

As the figure suggests, all of the components of Windows Azure AppFabric are built on Windows Azure (although they don't all provide services solely to Windows Azure applications). Those components are the following:

- 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 in the cloud 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: There are many ways for a user to get a digital identity today. The options include Active Directory, Windows Live ID, Google Accounts, Facebook, and more. If an application wants to let users log in with any of these, the application's creator faces the daunting challenge of supporting

this plethora of approaches. Access Control simplifies this by providing built-in support for all of them (and more). It also provides a single place for defining rules to control what each user is allowed to access.

- Caching: It's common for applications to access the same data over and over. One way to speed up this kind of application is to cache frequently accessed information, reducing the number of times that application must query a database. The Caching service provides this—and the performance boost it brings—for Windows Azure applications.

Microsoft has announced plans to add more services under the Windows Azure AppFabric banner, so expect this list to grow in the not-too-distant future.

As with Windows Azure and SQL Azure, customers use a browser-accessible portal to sign up for AppFabric. Once this has been done, these services can be used in a variety of ways. Here are some of the possibilities:

- 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 services created using WCF, then register those service endpoints with Service Bus. Its trading partners could then use Service Bus to find these endpoints and access the services.

- Imagine that the creator of this same application needs to let trading partners log in with a variety of different identities. Rather than implementing support for these identities himself, he could use the Access Control service to hide this complexity.

- A Windows Azure application created using ASP.NET might use the Session object to store per-client state. By changing only a configuration setting, the application can cause this data to be kept in the Caching service rather than, say, Windows Azure Storage tables. Doing this is likely to make the application faster and more scalable.

Along with the cloud-based services of Windows Azure AppFabric, Microsoft also provides 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 aren't exactly the same today as those in Windows Azure AppFabric (although Microsoft's announced plan is make the two congruent). Don't be confused; throughout this paper, the name "AppFabric" is used to refer to the cloud-based services. Also, don't confuse Windows Azure AppFabric with the Windows Azure fabric controller. Even though both contain the term "fabric", they're wholly separate technologies addressing quite distinct problems.

WINDOWS AZURE MARKETPLACE

In the on-premises world, many applications are purchased rather than custom-built. Many organizations also rely on datasets from commercial providers. With the rise of the cloud, why not let customers find and buy cloud applications and cloud-accessible data? Making this possible is the goal of Windows Azure Marketplace. Figure 5 shows its two components.

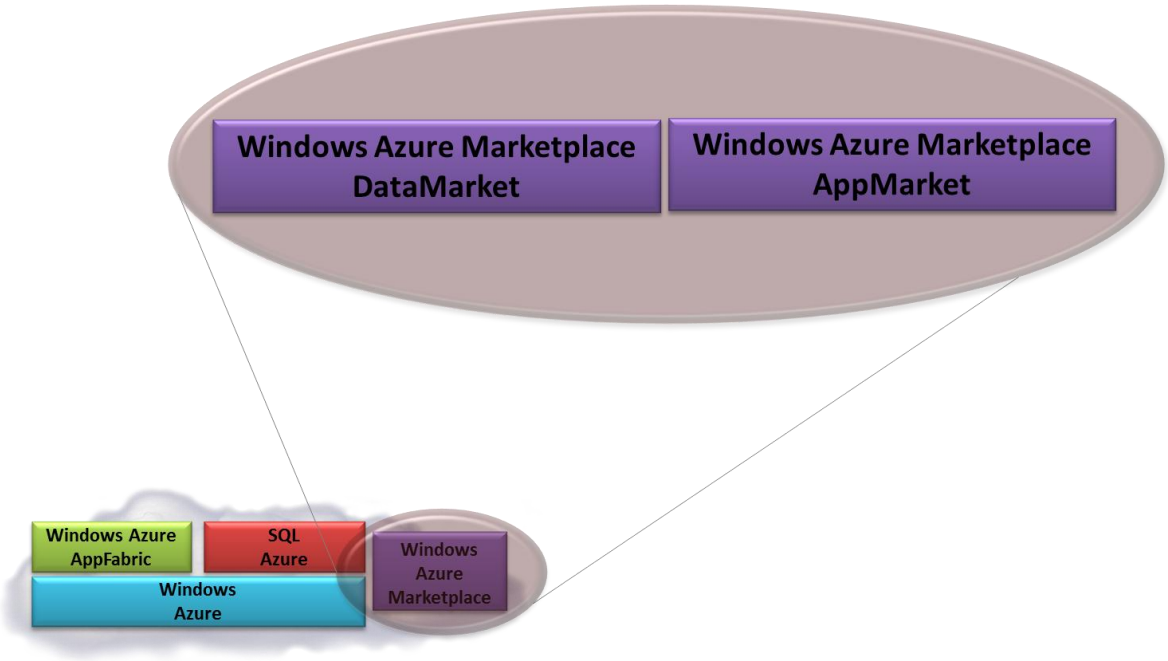

Figure 5: Windows Azure Marketplace provides an online marketplace for cloud applications and data.

The two parts of Windows Azure Marketplace are:

- DataMarket (originally codenamed "Dallas"), providing a way for content providers to make datasets available. Customers can browse the offerings, then purchase whatever they find useful. Applications, both custom and off-the-shelf (such as Microsoft Excel), can then access this data through RESTful requests or the OData protocol.
- AppMarket, providing a way for creators of cloud applications to expose those applications to potential customers. AppMarket will be available sometime after DataMarket.

Using the cloud to find, evaluate, and purchase data and applications makes sense. The goal of Windows Azure Marketplace is to make these things easier to do.

A CLOSER LOOK AT THE TECHNOLOGIES

Having a broad understanding of the Windows Azure platform is an important first step. Making good decisions require a deeper understanding of these technologies, however. This section takes a slightly more in-depth look at each member of the family.

WINDOWS AZURE

Windows Azure provides what's commonly called *Platform as a Service (PaaS).* Rather than offering a cloud replica of the on-premises world, it offers a higher level of service that's meant to make life easier for both developers and administrators. What follows walks through the five components that collectively provide this service.

Compute

An application built on the Windows Azure compute service is structured as one or more *roles*. When it executes, the application typically runs two or more *instances* of each role, with each instance running as its own virtual machine (VM). Figure 6 shows how this looks.

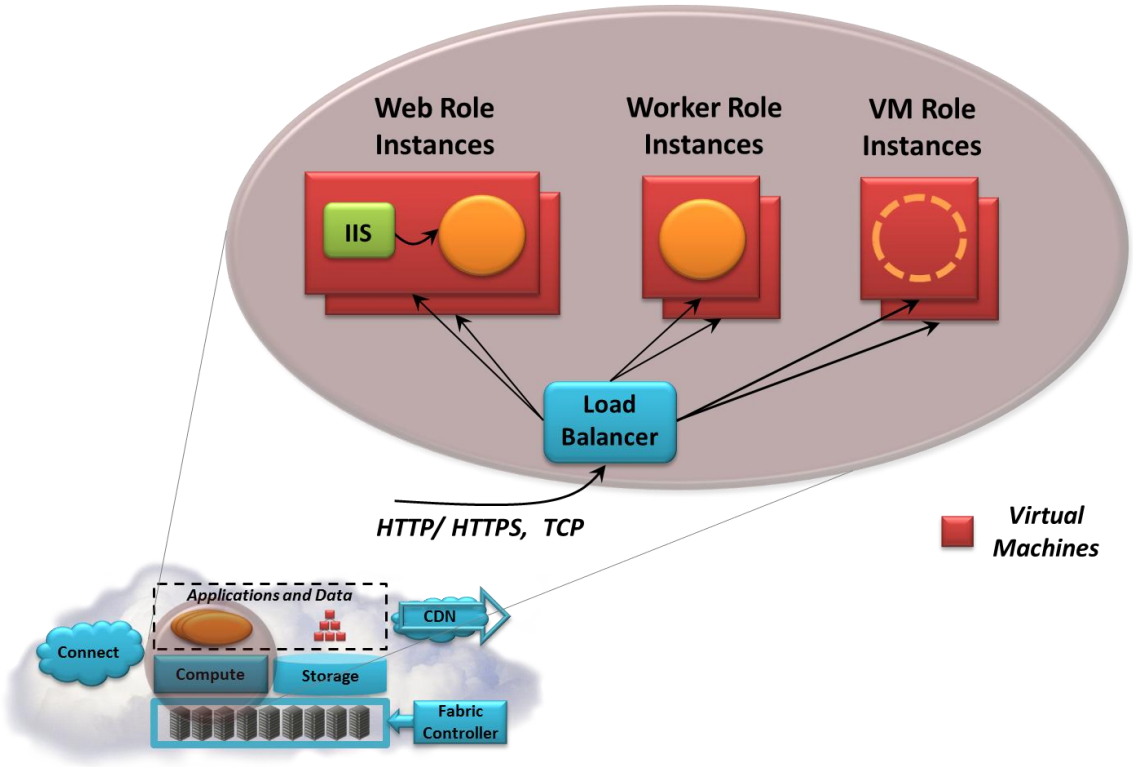

Figure 6: A running Windows Azure application consists of any combination of Web role instances, Worker role instances, and VM role instances.

A Windows Azure application today can be created using three kinds of roles:

- Web roles, intended primarily for running Web-based applications. Each Web role instance has Internet Information Services (IIS) 7 pre-configured to run inside it, so creating applications using ASP.NET, WCF, or other Web technologies is straightforward. It's also possible to create applications using PHP, Java, and other non-Microsoft technologies.

- Worker roles, designed to run a variety of code. A Worker role might run a simulation, for example, or video processing or nearly anything else. It's common for an application to interact with users through a Web role, then hand tasks off to a Worker role for processing.

- VM roles, which can run a user-provided Windows Server 2008 R2 image. A VM role can be the right choice for moving some on-premises Windows Server applications to Windows Azure.

When a developer gives Windows Azure an application to run, she submits configuration information along with it. Among other things, this information tells the platform how many instances of each role to run. The Windows Azure fabric controller then creates a VM for each instance, running the code for the

appropriate role in each VM. And as Figure 6 indicates, requests from the outside world are load balanced across all instances of a role.

This has an important implication: To be scalable, Windows Azure role instances shouldn't maintain their state themselves between requests. Because the load balancer doesn't allow creating an affinity with a particular role instance—there's no support for sticky sessions—there's no way to guarantee that multiple requests from the same user will be sent to the same instance. Instead, any client-specific state should be written to Windows Azure storage, stored in SQL Azure Database, or maintained externally in some other way.

For role instances, Windows Azure lets developers choose from several VM sizes, each with a specific number of processor cores and memory. Since each instance can be assigned one or more cores, applications have predictable performance. And to increase the load an application can handle, its owner or the application itself can request an increase in the number of running instances for one or more roles. The Windows Azure fabric controller will then spin up new VMs for these instances and start them running. It's also possible to explicitly decrease the number of instances for a role, letting an application grow and shrink as needed to handle changing loads.

For developers, building a Windows Azure application looks much like building a traditional Windows application. Microsoft provides Visual Studio project templates, for instance, to help developers create Windows Azure applications. 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 implements a local replica of the Windows Azure compute and storage services. A developer can create and debug his application using this local simulacrum, then deploy it to Windows Azure in the cloud when it's ready. Windows Azure also provides other services for developers, such as information about a running application's CPU consumption, incoming and outgoing bandwidth, and storage usage.

Storage

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 7 shows.

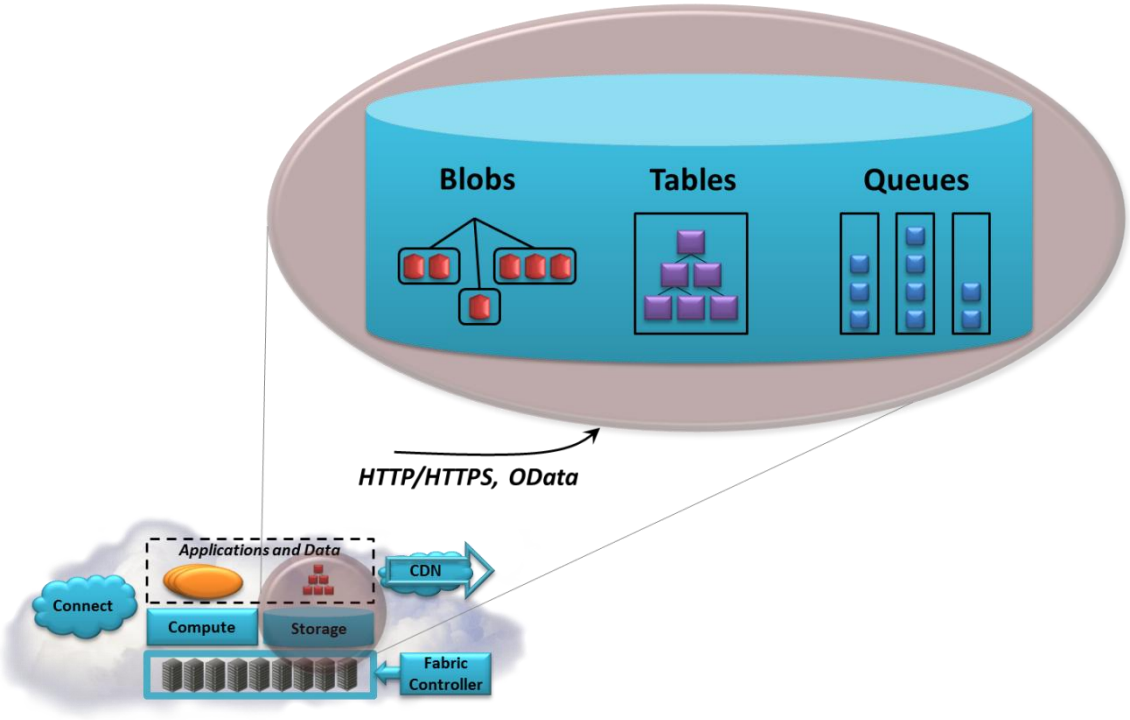

Figure 7: Windows Azure provides blobs, tables, and queues, all accessed in a RESTful style via HTTP, HTTPS, or OData.

The simplest way to store data in Windows Azure storage is to use blobs. As Figure 7 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—as large as a terabyte each—and to make transferring large blobs more efficient, they can potentially 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.

Another way to use blobs is through Windows Azure drives, which can be mounted by a role instance. The underlying storage for a drive 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 the simple query language defined by OData. 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 let Web role instances 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 (and the OData protocol for tables) to identify and expose data. Everything is named using URIs and accessed with standard HTTP operations, so clients can be created using .NET, Java, or other familiar technologies.

The Windows Azure platform charges independently for compute and storage resources. This means that an on-premises application can use just Windows Azure storage, accessing its data in the RESTful way just described. For example, a Windows Server application running in an enterprise data center might choose to store backups in Windows storage blobs.

Fabric Controller

Even though a Windows Azure application runs in VMs, a developer doesn't explicitly create and manage those VMs. Instead, as described earlier, she just tells the platform how many instances the application needs, and Windows Azure silently creates the necessary VMs and runs her application. This magic is performed by the fabric controller, a fundamental aspect of Windows Azure.

Along with creating VMs and starting the applications that run in them, the fabric controller also monitors all running instances. A role instance might fail for many reasons; maybe the code threw an exception or the VM crashed or the physical server it's running on went down. Whatever the reason, the fabric controller will start a new instance to maintain the total number of instances required for this application.

The fabric controller also handles housekeeping tasks like patching the operating system and other system software, which minimizes the need for administrators. These are all examples of why Windows Azure is viewed as PaaS: It provides a platform that lets customers run applications without worrying about administering the environment they run in.

Content Delivery Network

One common use of blobs is to store information that will be accessed from many different places. Think of an application that serves up videos, for example, to Flash, Silverlight, or HTML 5 clients around the world. To improve performance in situations like this, Windows Azure provides a content delivery network. The CDN stores copies of a blob at sites closer to the clients that use it. This speeds up delivery of frequently accessed content, improving performance for users of this information.

Connect

Windows Azure applications can interact with the outside world via HTTP, HTTPS, or TCP. But suppose you'd like to connect a role in an application to a machine outside the cloud at the IP level. For example, suppose an organization decides to move an existing ASP.NET application to Windows Azure, but wishes to keep the application's data in an on-premises SQL Server database. Windows Azure Connect allows this.

Using this option requires running Windows Azure Connect software on the on-premises machine that contains SQL Server, and it also requires some straightforward configuration. (This configuration can be done by the developer—there's no need to get a network administrator involved.) Once this is done, all of the instances in the Web role can behave as if they're on the same IP network as the SQL Server machine. In fact, they can use the same SQL Server connection string as when the application was running on premises—there's no difference.

Note that this isn't a full-fledged virtual private network (VPN). While Microsoft has announced plans to let customers connect to Windows Azure via a VPN, this isn't currently supported. It is possible to use Windows Azure Connect to domain-join a Windows Azure application to an on-premises Active Directory, however. Doing this allows single sign-on by on-premises users to the cloud application, and it also lets the application use existing Active Directory accounts and groups for access control.

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 substantial 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. Letting a specialized service provider ensure reliability and perform essential management functions can make sense, especially for organizations that find these things hard to do on their own. 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.

SQL Azure provides these things, together with cloud-based reporting and data synchronization services. And while all of them are important, the place to start is with the foundation: SQL Azure Database.

Database

SQL Azure Database is simple to understand. As seen by an application, it provides the core database functions of SQL Server as a cloud service. Figure 8 shows the basics of this technology.

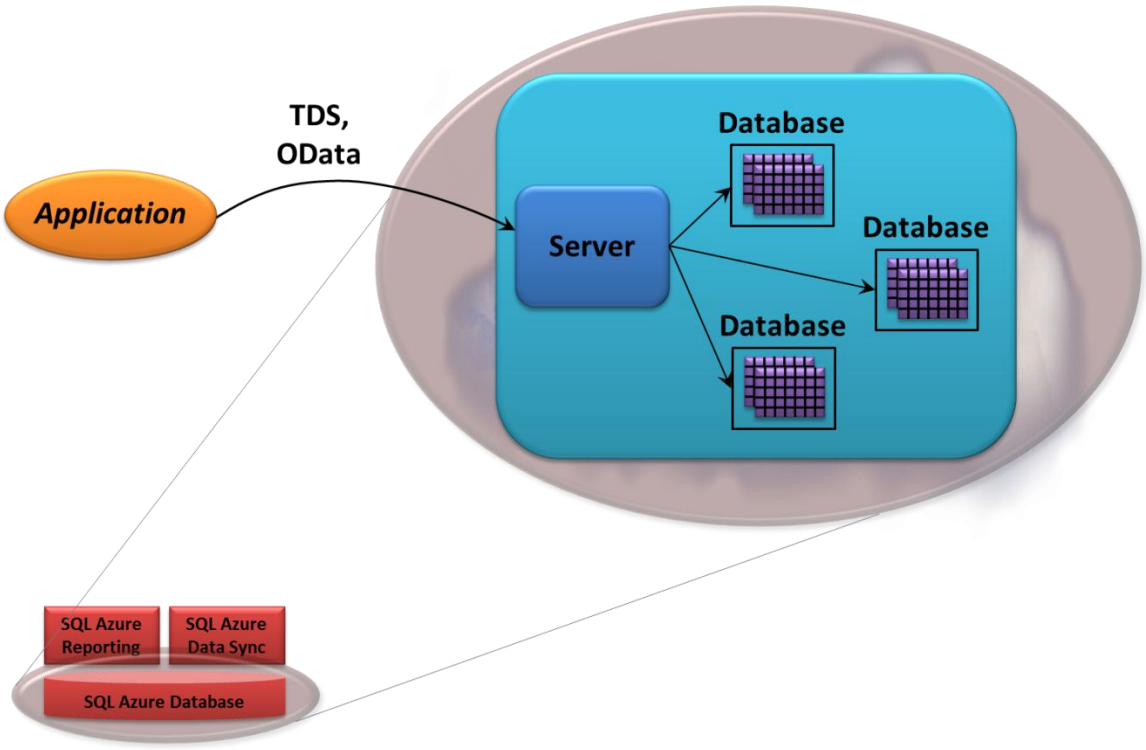

Figure 8: Applications access data in SQL Azure Database through Microsoft's TDS protocol or via OData.

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 typically 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. This includes Entity Framework, ADO.NET, ODBC, PHP and others. 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. Also, as Figure 8 shows, applications can optionally access SQL Azure Database using OData.

Each SQL Azure account can have one or more logical servers. (Note that these provide a way to organize the data and billing for an account—they aren't actual instances of SQL Server.) Each server can then have multiple databases, each of which can be up to 50 gigabytes in size. A user is free to use multiple databases if required, spreading data across them for better performance or for other reasons. It's also possible to store a snapshot of one SQL Azure database into another, providing a simple backup mechanism.

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 current release, however, such as the SQL Common Language Runtime (CLR) and support for full-text search. (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 for high availability. 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.

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.

Reporting

Storing data in SQL Azure Database can be useful. But as soon as that data exists, there's likely to be a demand for reports based on that data. Meeting this demand is the purpose of SQL Azure Reporting. Based on SQL Server Reporting Services (SSRS), this technology provides a cloud-based way to create reports.

Given that SQL Azure Database looks like SQL Server to an application, it's always possible to use SSRS on-premises with the data it contains. So why bother to provide a cloud reporting service as well? SQL Azure Reporting today targets two main scenarios:

- Reports created using SQL Azure Reporting can be published to a SQL Azure Reporting portal, letting users access it there, or made accessible directly via a URL.
- An ISV can embed reports published to the SQL Azure Reporting portal in any application, including Windows Azure applications. This lets users access those reports without leaving the application. To do this, the ISV can use the standard ReportViewer controls in Visual Studio—it's no different from embedding on-premises reports into applications.

SQL Azure Reporting is designed to work with data stored in SQL Azure Database. Reports used with SQL Azure Reporting are created on-premises, however, with Business Intelligence Developer Studio, the Visual Studio-hosted tool that's also used to create SSRS reports. In fact, SQL Azure Reporting relies on the same reporting format, expressed in the Report Definition Language (RDL), used with SSRS.

Like other aspects of SQL Azure, SQL Azure Reporting doesn't provide all of the functions offered by SSRS on premises. There's currently no support for scheduling and subscriptions, for example, that let a report be run and delivered at regular intervals. Still, cloud-based reporting services can be useful, and they're an important part of the SQL Azure technology family.

Data Sync

Storing data in SQL Azure Database can make it accessible to any application with an Internet connection. Still, there are often situations where it makes sense to maintain a copy of this cloud-based data in some other place. Suppose an organization needs to have an on-premises copy of the same information, for example, for performance reasons or to ensure access if the network fails. For scenarios like this, the ability to synchronize data in SQL Azure Database with other data stores is useful.

It's always possible to write synchronization code yourself using the Microsoft Sync Framework. To make this easier, Microsoft provides SQL Azure Data Sync. Rather than requiring its user to write code, this technology is entirely configuration-driven (although unsurprisingly, it's built on the Microsoft Sync Framework). As Figure 9 shows, SQL Azure Data Sync supports two options today.

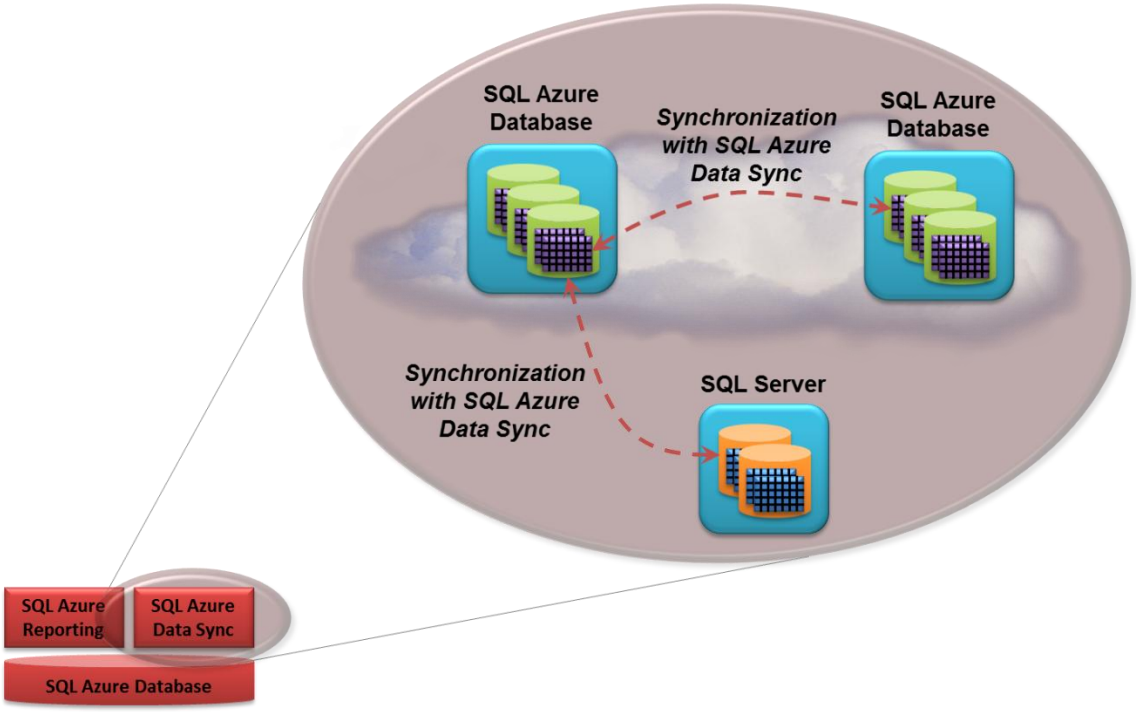

Figure 9: SQL Azure Data Sync can synchronize data between SQL Azure databases or between a SQL Azure database and on-premises SQL Server databases.

The two synchronization options are the following:

- Synchronizing data between a SQL Azure database and an on-premises SQL Server database. Having an on-premises copy of cloud data might make sense for a variety of reasons. Maybe an organization wants to make sure that this data is always available, for instance, even in the event of network failures, or perhaps regulations require that a copy of the data always be available within a country's borders. And even though SQL Azure's built-in data replication guards against hardware failures, the data's owner might choose to maintain an on-premises backup copy in case of administrative errors, such as incorrect table deletions.

- Synchronizing data between SQL Azure databases in different Microsoft data centers. For example, suppose an ISV or a global enterprise creates an application that's used by people around the world. To provide good performance for all of them, its creator might choose to run this application in three different Windows Azure data centers: one each in North America, Europe, and Asia. If the application stores data in SQL Azure Database, it might rely on SQL Azure Data Sync to keep information synchronized across these three data centers.

SQL Azure Data Sync uses a hub-and-spoke model. All changes are copied first to the SQL Azure database hub, then out to the spokes (referred to as *members*). The members might be other SQL Azure databases

or on-premises SQL Server databases. In both cases, the technology provides synchronization of an entire database or just specific tables, with changes made to any copy propagated to all of the others. And while it's possible to initiate synchronization manually, SQL Azure Data Sync also includes a scheduling service. This lets users schedule, say, hourly synchronization between a pair of databases. However it's used, the purpose is the same: providing a straightforward way to synchronize data between a SQL Azure database in a Microsoft data center and one or more databases located somewhere else.

WINDOWS AZURE APPFABRIC

Applications exist because their business logic has value. Yet applications rely on infrastructure to provide a foundation for that logic. A good application platform provides this infrastructure; application developers shouldn't need to write it themselves.

Windows Azure AppFabric provides infrastructure for applications. The people who create applications can benefit from different kinds of infrastructure, and so AppFabric contains various parts. This section takes a closer look at Service Bus, Access Control, and Caching, the three components of Windows Azure AppFabric today.

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 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 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 network administrators frown on this.

Service Bus addresses these challenges. Figure 10 shows how.

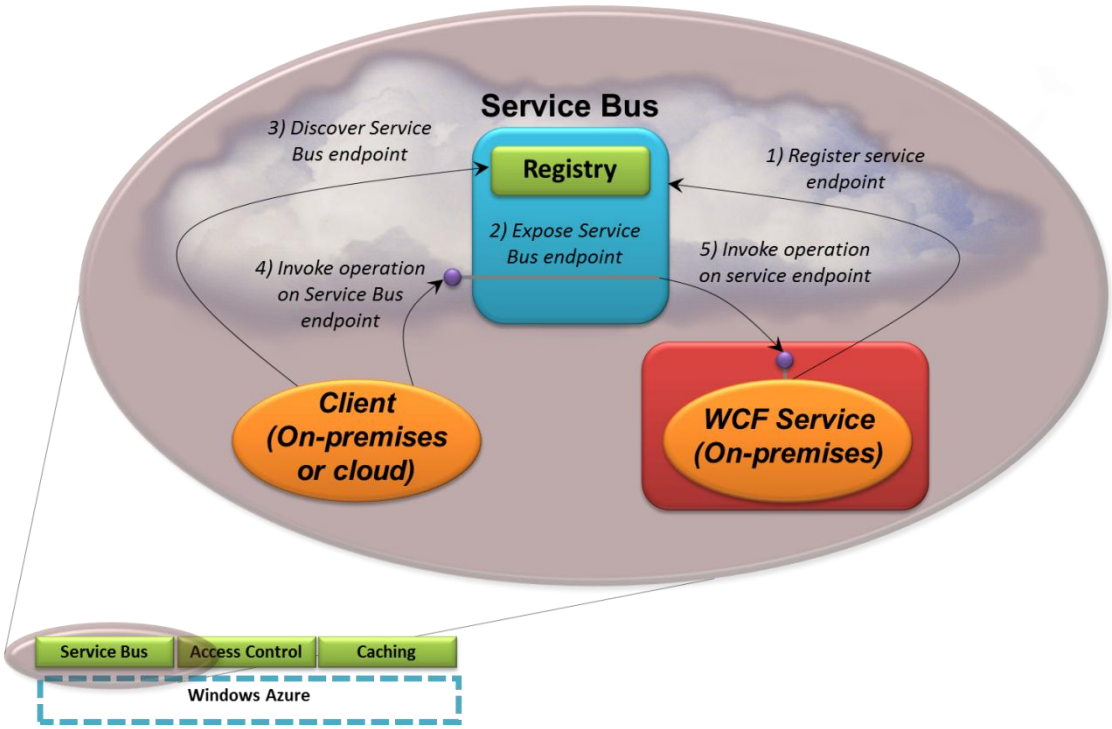

Figure 10: A WCF service can register endpoints with Service Bus, then have clients discover and use those endpoints to access the service.

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 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.

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, providing the endpoint's URI. 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). (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.)

There's an obvious question here: How exactly does step 5 work? How does the request from Service Bus back to your service deal with the challenges of NAT and firewalls? The answer is that in step 1, your service opened a TCP connection with Service Bus for this exposed endpoint. 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.

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 it. 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, these clients can make requests via TCP, HTTP, or HTTPS. Applications are also free to use their own security mechanisms, such as encryption, to shield their communication from attackers.

Service Bus provides a few more useful features, including these:

- Support for message buffers, which act like simple queues. Rather than require a client to make direct calls to a service, the client can place a message of up to 256 kilobytes into a message buffer. The message is persisted to disk—it's durable—and the service can then read this message at some later time. And as usual in the Windows Azure platform, the persisted messages are replicated to guard against failures.

- The ability for multiple WCF services to listen on the same URI. Service Bus will then randomly spread client requests across all of the listening services. The goal is to provide both load balancing and fault tolerance for the WCF services.

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

Access Control

Working with identity is a fundamental part of most distributed applications. The modern approach to this, called *claims-based identity*, lets a user send a *token* full of *claims* that contain identity information. One claim might contain the user's name, for example, while another contains her age or a group she belongs to. An application can use the claims in a token to decide what the user is allowed to do or in other ways.

In a claims-based world, tokens are issued by *identity providers (IdPs)*. Some IdPs, such as Active Directory Federation Services (AD FS) 2.0, exist inside organizations. Others, such as Windows Live ID and Google Accounts, are accessible to anyone via the Internet. Each application can decide which IdPs it trusts and thus which tokens it's willing to accept. For example, an application running inside your company might only accept tokens issued by your own AD FS server, while an application running on the Internet might accept tokens issued by, say, Google and Facebook.

Yet different IdPs use different token formats, and they represent claims in different ways. An application that directly accepts identities from Google, Facebook, and Yahoo, for instance, would need to handle these differences itself. But why do this? Why not instead create an intermediary that can generate a single token format with a common representation of claims? Doing this would make life simpler for the people who create applications, since they now need to handle only one kind of token.

The Access Control service does exactly this, providing an intermediary in the cloud for working with claims-based identity. Figure 11 shows how it works.

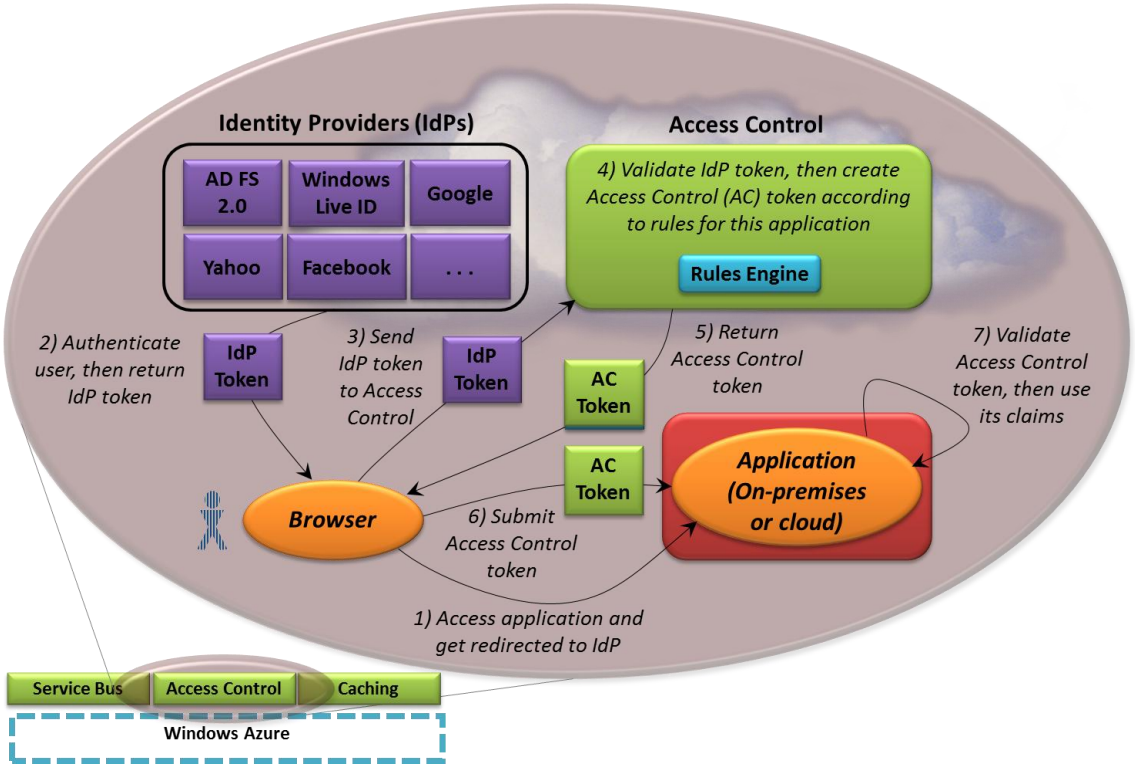

Figure 11: The Access Control service makes it easier for applications to accept identity information issued by different identity providers.

As the figure shows, an application that relies on Access Control can run either on-premises or in the cloud. In either case, the process begins when a user attempts to access this application via a browser (step 1). The application then redirects the browser to an IdP whose token this application will accept. The user authenticates herself with this IdP, such as by entering a username and password, and the IdP returns a token containing claims about her (step 2).

Next, the user's browser sends the IdP token to Access Control (step 3). Access Control validates the token, making sure that it really was issued by this IdP, then creates a new token according to whatever rules have been defined for this application (step 4). Access Control contains a rules engine, allowing each application's administrator to define how tokens from various IdPs should be transformed into an Access Control token. For example, if different IdPs use different formats for representing usernames, Access Control rules can transform all of these into a common username string. Access Control then sends this new token back to the browser (step 5), which submits it to the application (step 6). Once it has the Access Control token, the application verifies that it really was issued by Access Control, then uses the claims it contains (step 7).

While this process might seem a little complicated, it actually makes life significantly simpler for the creator of the application. Rather than handle diverse tokens containing various claims, the application can accept identities issued by multiple identity providers while still receiving only a single token with familiar claims. And rather than require each application to be configured to trust various IdPs, these trust relationships are instead maintained by Access Control—an application need only trust it.

As Figure 11 shows, Access Control has built-in support for several identity providers, including AD FS 2.0, Windows Live ID, Google, Yahoo, and Facebook. It can also work with any IdP that supports OpenID. Browsers and other clients can request Access Control tokens using either OAuth 2 or WS-Trust, and these tokens can have various formats, including SAML 1.1, SAML 2.0, and Simple Web Token (SWT). And to create applications that accept Access Control tokens, Windows developers can use Windows Identity Foundation (WIF). (It's worth pointing out, however, that nothing about Access Control is tied to Windows—it could just as well be used by a Linux application that accepted only Google and Facebook identities.)

Working with identity is important in nearly every distributed application. The goal of Access Control is to make it easier for developers to create secure applications that accept identities from diverse providers. By putting this service in the cloud, Microsoft has made it available to any application running on any platform.

Caching

For many applications, one of the most effective ways to improve performance is by caching frequently accessed data. Since applications tend to use the same information over and over, making this information more readily accessible can make the application faster. Doing this is the purpose of Windows Azure AppFabric Caching. Figure 12 illustrates the idea.

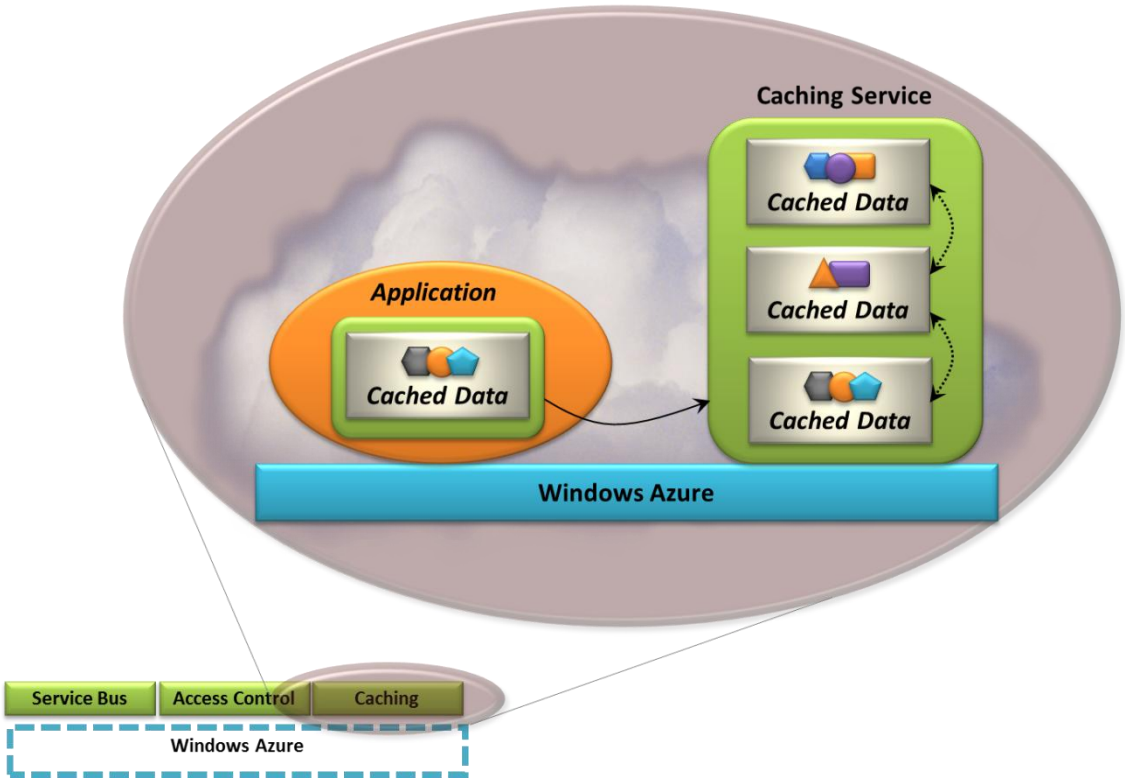

Figure 12: Windows Azure AppFabric Caching speeds up access to frequently accessed data for Windows Azure applications.

The Caching service provides a distributed cache for Windows Azure applications, along with a library to access this cache. As Figure 12 suggests, the service includes a local cache that can keep a copy of recently accessed data items in each of an application's role instances. If a data item the application needs isn't found in the local cache, the cache library automatically contacts the shared cache provided by the Caching service. As the figure shows, this cache is spread across a number of Windows Azure instances, each holding different cached data. This diversity isn't visible to applications using the cache, however. An application just requests a data item, then lets the cache find it (if it's in the cache) and return it from whatever instance contains it.

Recently accessed data isn't automatically cached, however. Instead, the application explicitly inserts data items in the cache, such as by using the Caching API. It's also possible to configure an ASP.NET application running on Windows Azure to store Session object data in the Caching service, speeding it up without changing any of its code.

Windows Server AppFabric, the on-premises analog to Windows Azure AppFabric, also provides a Caching service. In fact, the two are very similar. The biggest different is that unlike its on-premises counterpart, Windows Azure AppFabric Caching is a service—there's no need to configure servers and administer the cache. Instead, all of this is handled automatically by the service itself. And the cloud Caching service is multi-tenant, so each application using it gets its own instance. Because the application must authenticate itself to this instance, data held in the Caching service isn't accessible to other applications.

Caching can make applications faster and more scalable with little effort from developers, and so using it makes sense. By providing caching as a service, Windows Azure AppFabric Caching makes this easier to do.

WINDOWS AZURE MARKETPLACE

Cloud platforms are useful, but they're just a means to an end. The real goal is offering applications and data that provide value for people. Since the Windows Azure platform provides a foundation for both, why shouldn't it also include a way for people to find the applications and data that they need?

The Windows Azure Marketplace is designed to address this need. Its two components, AppMarket and DataMarket, let users find, try, and purchase applications and data, respectively. While both are important, DataMarket will appear first, and so what follows focuses on this aspect of the Marketplace.

Buying applications is common today—nearly every organization does it. Buying data is less common, but it's no less important. Many firms sell many different kinds of data, including demographic information, financial information, legal information, and much more. Yet using purchased data typically requires figuring out whether the data you need exists, finding a firm that offers it, then determining whether their data meets your needs.

All of this is more difficult than it needs to be. Why not create one place where customers can find all kinds of data from all kinds of content providers? Why not let them examine the data to make sure it meets their needs, then purchase what they need right then and there? The DataMarket component of Windows Azure Marketplace was created to do these things. Figure 13 shows its main components.

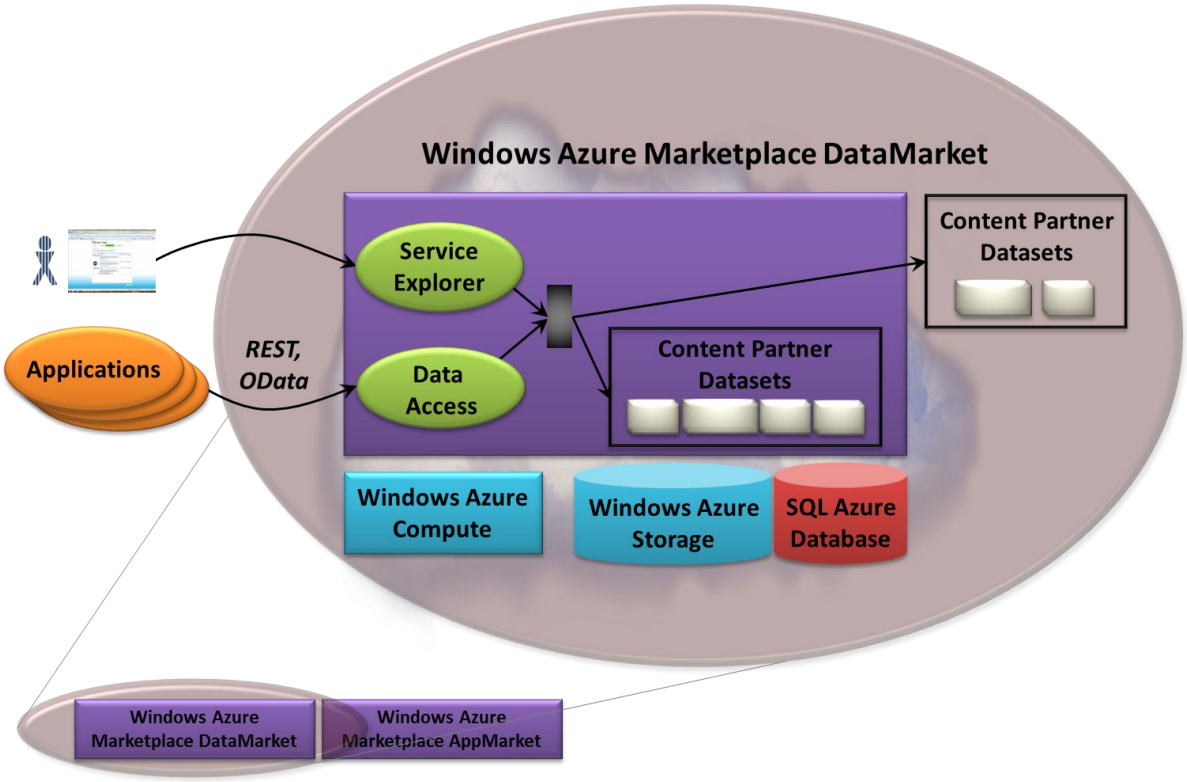

Figure 13: Built on Windows Azure, the DataMarket allows access to content partner datasets stored in Windows Azure, SQL Azure, or externally.

As the figure suggests, both people and applications can access information through DataMarket. Using a Windows Azure-based application called the Service Explorer, a user can see what datasets are available, then purchase what she needs. Once this is done, applications can access that data using RESTful or OData requests. The datasets made available by DataMarket can be stored on the Windows Azure platform itself using Windows Azure Storage or SQL Azure Database. They can also be stored externally, such as in a data center owned by the content provider. There's no requirement to put everything in the cloud.

For customers, DataMarket provides a single place to find, buy, and access a variety of commercial datasets. For content providers—the owners of those data sets—DataMarket offers a chance to expose their offerings to more customers through Microsoft's cloud platform. While content providers set prices for their data, DataMarket provides a billing service that frees them from dealing directly with customers. Microsoft will also vet the quality of the content providers, limiting DataMarket initially to just the top five providers in a particular industry.

Applications are free to use purchased data in any way that the content provider's license allows. Users of Microsoft Excel 2010, for example, can access DataMarket's information directly using an Excel add-in or use the PowerPivot for Excel support for OData to access information from DataMarket for data analysis. It's also possible to combine data purchased from DataMarket with your own data, such as for reports created using SQL Server Reporting Services.

Getting business value from information technology requires the right applications, but it also requires the right data. By making it easier to discover, evaluate, and buy commercially available datasets, DataMarket intends to make it simpler for organizations to find and use the information they need.

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:

- The Windows Azure Platform Appliance: Microsoft has announced plans to offer a Windows Azure Platform Appliance, allowing hosters and enterprises to run Windows Azure in their own data centers. Because Windows Azure requires a specific hardware configuration—it's not just software—this appliance will include servers, networking, and storage, all running Windows Azure. And while it will initially be targeted only at large organizations, Microsoft's goal is eventually to make the Windows Azure Platform Appliance useful for smaller organizations as well.

- CDN dynamic content caching: Today, the Windows Azure CDN works only with blob data. This forthcoming functionality will let the CDN also cache content created dynamically by a Windows Azure application. **CDN SSL Delivery**: Users of the Windows Azure CDN will now have the capability to deliver content via encrypted channels with SSL/TLS.

- VM role snapshotting: In its first release, the Windows Azure VM role doesn't save any changes made to the OS volume while it's running. Snapshotting will change this, providing a way to periodically save the state of this volume to persistent storage.

- Better Java support**:** While Windows Azure can run Java applications today, Microsoft plans to make this better. The coming improvements include better Java performance, stronger support for Eclipse-based tools, and a more complete set of Java libraries for Windows Azure.

- Support for composite applications: To make it easier to build Windows Azure applications by combining existing components and services, Windows Azure AppFabric will add a composition model. Accompanied by runtime services and a Visual Studio-based designer, this model will let developers work with the various parts of an application as a single logical entity throughout its lifecycle.

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. With the Windows Azure platform, Microsoft presents a range of options addressing a variety of needs:

- Windows Azure provides a computing and storage environment in the cloud.

- SQL Azure provides a relational DBMS in the cloud, together with reporting and data synchronization.

- Windows Azure AppFabric offers cloud-based infrastructure supporting both cloud and on-premises applications.
- Windows Azure Marketplace is an online store for finding and purchasing datasets from content providers, together with a forthcoming store for cloud applications.

These approaches address a variety of requirements, and not every developer will use all of them. Yet whether you work for an ISV, a systems integrator, 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.