WINDOWS AZURE DATA MANAGEMENT AND BUSINESS ANALYTICS

Managing and analyzing data in the cloud is just as important as it is anywhere else. To let you do this, Windows Azure provides a range of technologies for working with relational and non-relational data. This article introduces each of the options.

Contents

Blob Storage

The word “blob” is short for “Binary Large OBject”, and it describes exactly what a blob is: a collection of binary information. Yet even though they’re simple, blobs are quite useful. Figure 1 illustrates the basics of Windows Azure Blob Storage.

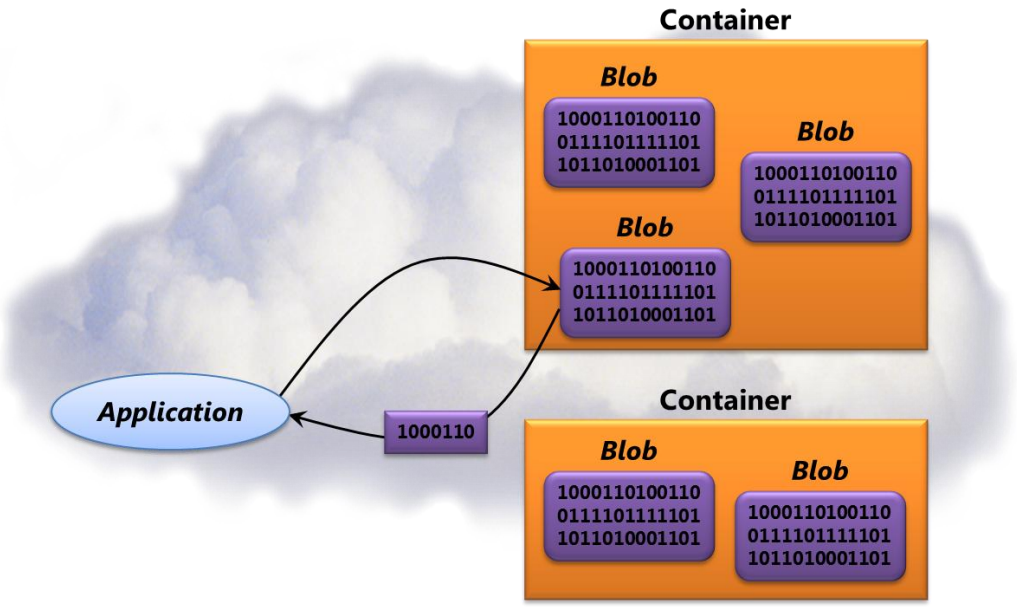

Figure 1: Windows Azure Blob Storage stores binary data—blobs—in containers.

To use blobs, you first create a Windows Azure *storage account*. As part of this, you specify the Windows Azure datacenter that will store the objects you create using this account. Wherever it lives, each blob you create belongs to some *container* in your storage account. To access a blob, an application provides a URL with the form

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

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

Windows Azure provides two different kinds of blobs. The choices are:

- *Block* blobs, each of which can contain up to 200 gigabytes of data. As its name suggests, a block blob is subdivided into some number of blocks. If a failure occurs while transferring a block blob, retransmission can resume with the most recent block rather than sending the entire blob again. Block blobs are a quite general approach to storage, and they're the most commonly used blob type today.
- *Page* blobs, which can be as large at one terabyte each. Page blobs are designed for random access, and so each one is divided into some number of pages. An application is free to read and write individual pages at random in the blob. In Windows Azure Virtual Machines, for example, VMs you create use page blobs as persistent storage for both OS disks and data disks.

Whether you choose block blobs or page blobs, applications can access blob data in several different ways. The options include the following:

- Directly through a RESTful (i.e., HTTP-based) access protocol. Both Windows Azure applications and external applications, including apps running on premises, can use this option.
- Using the Windows Azure Storage Client library, which provides a more developer-friendly interface on top of the raw RESTful blob access protocol. Once again, both Windows Azure applications and external applications can access blobs using this library.
- Using Windows Azure drives, an option that lets a Windows Azure application treat a page blob as a local drive with an NTFS file system. To the application, the page blob looks like an ordinary Windows file system accessed using standard file I/O. In fact, reads and writes are sent to the underlying page blob that implements the Windows Azure Drive.

To guard against hardware failures and improve availability, every blob is replicated across three computers in a Windows Azure datacenter. Writing to a blob updates all three copies, so later reads won't see inconsistent results. You can also specify that a blob's data should be copied to another Windows Azure datacenter in the same region but at least 500 miles away. This copying, called *geo-replication*, happens within a few minutes of an update to the blob, and it's useful for disaster recovery.

Data in blobs can also be made available via the Windows Azure *Content Delivery Network (CDN)*. By caching copies of blob data at dozens of servers around the world, the CDN can speed up access to information that's accessed repeatedly.

Simple as they are, blobs are the right choice in many situations. Storing and streaming video and audio are obvious examples, as are backups and other kinds of data archiving. Developers can also use blobs to hold any kind of unstructured data they like. Having a straightforward way to store and access binary data can be surprisingly useful.

Running a DBMS in a Virtual Machine

Many applications today rely on some kind of database management system (DBMS). Relational systems such as SQL Server are the most frequently used choice, but non-relational approaches, commonly known as *NoSQL* technologies, get more popular every day. To let cloud applications use these data management options, Windows Azure Virtual Machines allows you to run a DBMS (relational or NoSQL) in a VM. Figure 2 shows how this looks with SQL Server.

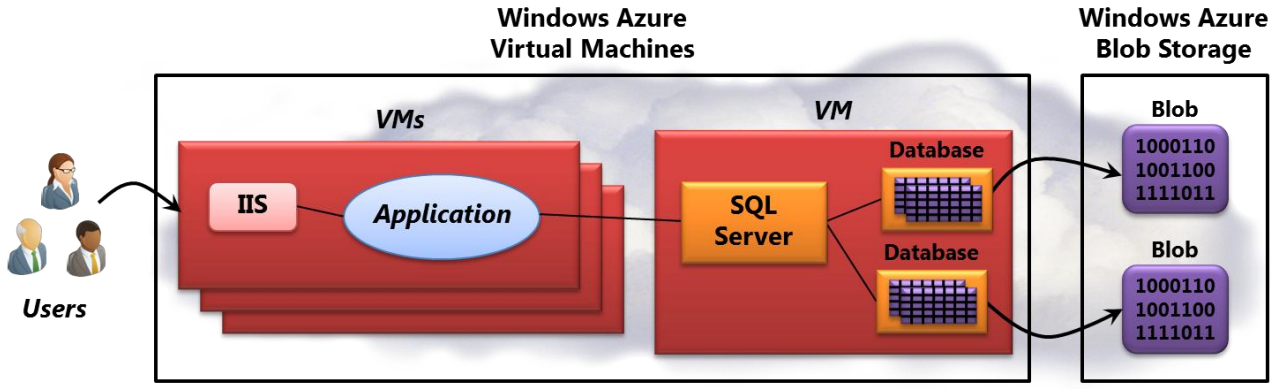

Figure 2: Windows Azure Virtual Machines allows running a DBMS in a VM, with persistence provided by blobs.

To both developers and database administrators, this scenario looks much like running the same software in their own datacenter. In the example shown here, for instance, nearly all of SQL Server's capabilities can be used, and you have full administrative access to the system. You also have the responsibility of managing the database server, of course, just as if it were running locally.

As Figure 2 shows, your databases appear to be stored on the local disk of the VM the server runs in. Under the covers, however, each of those disks is written to a Windows Azure blob. (It's similar to using a SAN in your own datacenter, with a blob acting much like a LUN.) As with any Windows Azure blob, the data it contains is replicated three times within a datacenter and, if you request it, geo-replicated to another datacenter in the same region. It's also possible to use options such as SQL Server database mirroring for improved reliability.

Another way to use SQL Server in a VM is to create a hybrid application, where the data lives on Windows Azure while the application logic runs on-premises. For example, this might make sense when applications running in multiple locations or on various mobile devices must share the same data. To make communication between the cloud database and on-premises logic simpler, an organization can use Windows Azure Virtual Network to create a virtual private network (VPN) connection between a Windows Azure datacenter and its own on-premises datacenter.

SQL Database

For many people, running a DBMS in a VM is the first option that comes to mind for managing structured data in the cloud. It's not the only choice, though, nor is it always the best choice. In some cases, managing data using a Platform as a Service (PaaS) approach makes more sense. Windows Azure provides a PaaS technology called SQL Database that lets you do this for relational data. Figure 3 illustrates this option.

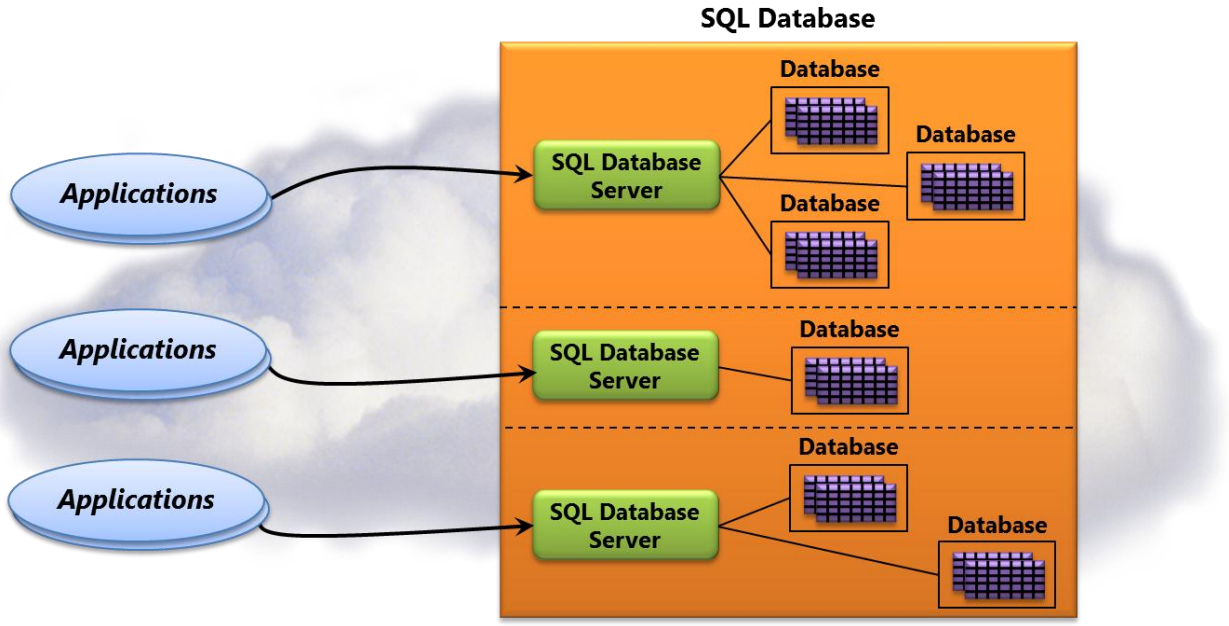

Figure 3: SQL Database provides a shared PaaS relational storage service.

SQL Database doesn't give each customer its own physical instance of SQL Server. Instead, it provides a multi-tenant service, with a logical SQL Database server for each customer. All customers share the compute and storage capacity that the service provides. And as with Blob Storage, all data in SQL Database is stored on three separate computers within a Windows Azure datacenter, giving your databases built-in high availability (HA).

To an application, SQL Database looks much like SQL Server. Applications can issue SQL queries against relational tables, use T-SQL stored procedures, and execute transactions across multiple tables. And because applications access SQL Database using the Tabular Data Stream (TDS) protocol, the same protocol used to access SQL Server, they can work with data using Entity Framework, ADO.NET, JDBC, and other familiar data access interfaces.

But because SQL Database is a cloud service running in Windows Azure data centers, you don't need to manage any of the system's physical aspects, such as disk usage. You also don't need to worry about updating software or handling other low-level administrative tasks. Each customer organization still controls its own databases, of course, including their schemas and user logins, but many of the mundane administrative tasks are done for you.

While SQL Database looks much like SQL Server to applications, it doesn't behave exactly the same as a DBMS running on a physical or virtual machine. Because it runs on shared hardware, its performance will vary with the load placed on that hardware by all of its customers. This means that the performance of, say, a stored procedure in SQL Database might vary from one day to the next.

Today, SQL Database lets you create a database holding up to 150 gigabytes. If you need to work with larger databases, the service provides an option called *Federation*. To do this, a database administrator creates two or more *federation members*, each of which is a separate database with its own schema. Data is spread across these members, something that's often referred to as *sharding*, with each member assigned a unique *federation key*. An application issues SQL queries against this data by specifying the federation key that identifies the federation member the query should target. This allows using a traditional relational approach with large amounts of data. As always, there are trade-offs; neither queries nor transactions can span federation members, for instance. But when a relational PaaS service is the best choice and these trade-offs are acceptable, using SQL Federation can be a good solution.

SQL Database can be used by applications running on Windows Azure or elsewhere, such as in your on-premises datacenter. This makes it useful for cloud applications that need relational data, as well as on-premises applications that can benefit from storing data in the cloud. A mobile application might rely on SQL Database to manage shared relational data, for instance, as might an inventory application that runs at multiple dealers around the world.

Thinking about SQL Database raises an obvious (and important) issue: When should you run SQL Server in a VM, and when is SQL Database a better choice? As usual, there are trade-offs, and so which approach is better depends on your requirements.

One simple way to think about it is to view SQL Database as being for new applications, while SQL Server in a VM is a better choice when you're moving an existing on-premises application to the cloud. It can also be useful to look at this decision in a more fine-grained way, however. For example, SQL Database is easier to use, since there's minimal setup and administration. But running SQL Server in a VM can have more predictable performance—it's not a shared service—and it also supports larger non-federated databases than SQL Database. Still, SQL Database provides built-in replication of both data and processing, effectively giving you a high-availability DBMS with very little work. While SQL Server gives you more control and a somewhat broader set of options, SQL Database is simpler to set up and significantly less work to manage.

Finally, it's important to point out that SQL Database isn't the only PaaS data service available on Windows Azure. Microsoft partners provide other options as well. For example, ClearDB offers a MySQL PaaS offering, while Cloudant sells a NoSQL option. PaaS data services are the right solution in many situations, and so this approach to data management is an important part of Windows Azure.

SQL Data Sync

While SQL Database does maintain three copies of each database within a single Windows Azure datacenter, it doesn't automatically replicate data between Windows Azure datacenters. Instead, it provides SQL Data Sync, a service that you can use to do this. Figure 4 shows how this looks.

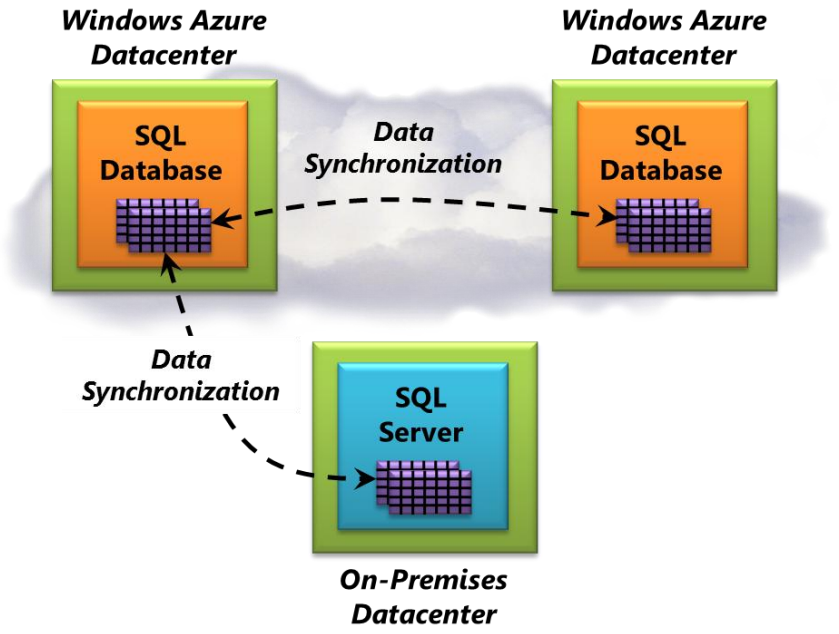

Figure 4: SQL Data Sync synchronizes data in SQL Database with data in other Windows Azure and on-premises datacenters.

As the diagram shows, SQL Data Sync can synchronize data across different locations. Suppose you're running an application in multiple Windows Azure datacenters, for instance, with data stored in SQL Database. You can use SQL Data Sync to keep that data synchronized. SQL Data Sync can also synchronize data between a Windows Azure datacenter and an instance of SQL Server running in an on-premises datacenter. This might be useful for maintaining both a local copy of data used by on-premises applications and a cloud copy used by applications running on Windows Azure. And although it's not shown in the figure, SQL Data Sync can also be used to synchronize data between SQL Database and SQL Server running in a VM on Windows Azure or elsewhere.

Synchronization can be bi-directional, and you determine exactly what data is synchronized and how frequently it's done. (Synchronization between databases isn't atomic, however—there's always at least some delay.) And however it's used, setting up synchronization with SQL Data Sync is entirely configuration-driven; there's no code to write.

SQL Reporting

Once a database contains data, somebody will probably want to create reports using that data. To let you do this with data stored in SQL Database, Windows Azure provides SQL Reporting. This cloud service provides a subset of the functionality in SQL Server Reporting Services (SSRS), the reporting technology included with SQL Server.

In its initial incarnation, SQL Reporting is aimed primarily at independent software vendors (ISVs) who need to embed reports in their applications. Figure 5 shows how the process works.

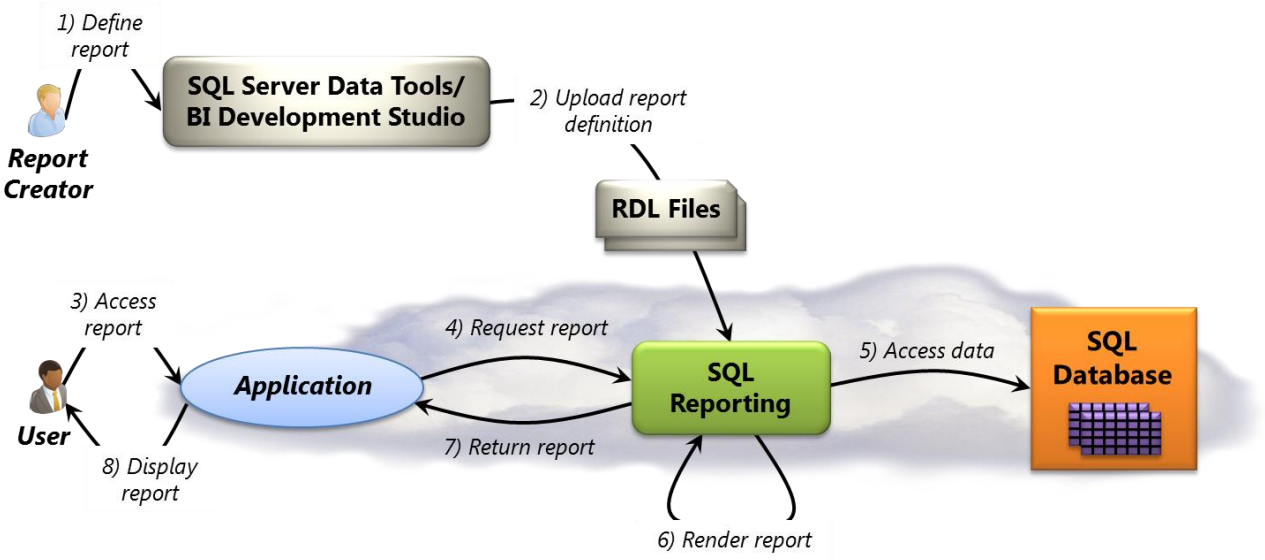

Figure 5: Windows Azure SQL Reporting provides reporting services for data in SQL Database.

Before a user can see a report, someone defines what that report should look like (step 1). With SQL Reporting, this can be done using either of two tools: SQL Server Data Tools, part of SQL Server 2012, or its predecessor, Business Intelligence (BI) Development Studio. As with SSRS, these report definitions are expressed in the *Report Definition Language (RDL)*. Once the RDL files for a report have been created, they are uploaded to SQL Reporting in the cloud (step 2). The report definition is now ready to use.

Next, a user of the application accesses the report (step 3). The application passes this request to SQL Reporting (step 4), which contacts SQL Database to get the data it needs (step 5). SQL Reporting uses this data and the relevant RDL files to render the report (step 6), then returns the report to the application (step 7), which displays it to the user (step 8).

Embedding a report in an application, the scenario shown here, isn't the only option. It's also possible to view reports in a SQL Reporting portal or in other ways. Reports can also be combined, with one report containing a link to another.

Like SQL Database, SQL Reporting is a multi-tenant PaaS service. You can use it immediately—there's nothing to install—and it requires minimal management. Microsoft monitors the service, provides patches, handles scaling, and does the other work needed to keep the service available. While it's possible to run reports on SQL Database tables using the on-premises version of SSRS, SQL Reporting is typically a better alternative for adding reporting to Windows Azure applications.

Table Storage

Relational data is useful in many situations, but it's not always the right choice. If your application needs fast, simple access to very large amounts of loosely structured data, for instance, a relational database might not work well. A NoSQL technology is likely to be a better option.

Windows Azure Table Storage is an example of this kind of NoSQL approach. Despite its name, Table Storage doesn't support standard relational tables. Instead, it provides what's known as a *key/value store*, associating a set of data with a particular key, then letting an application access that data by providing the key. Figure 6 illustrates the basics.

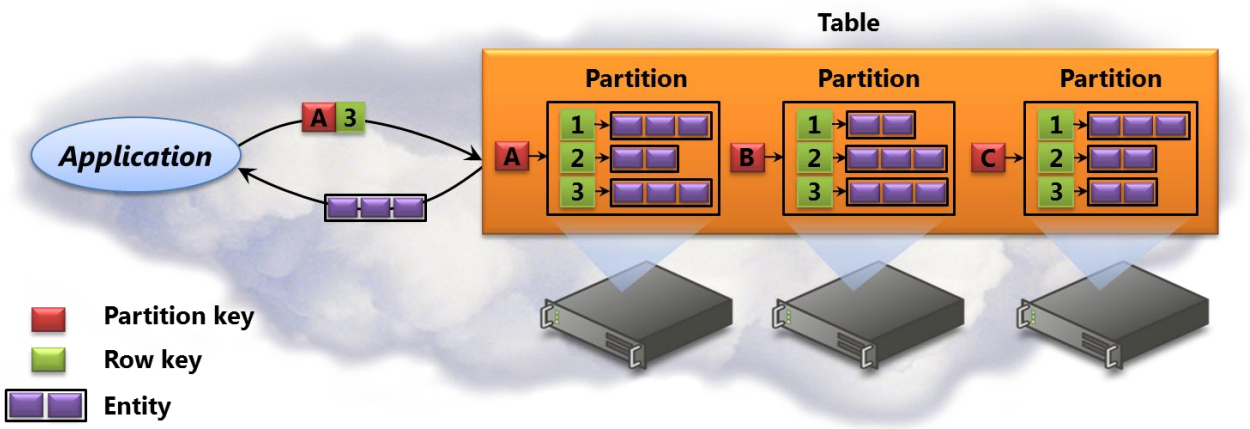

Figure 6: Windows Azure Table Storage is a key/value store that provides fast, simple access to large amounts of data.

Like blobs, each table is associated with a Windows Azure storage account. Tables are also named much like blobs, with a URL of the form

`http://<StorageAccount>.table.core.windows.net/<TableName>`

As the figure shows, each table is divided into some number of *partitions*, each of which can be stored on a separate machine. (This is a form of sharding, as with SQL Federation.) Both Windows Azure applications and applications running elsewhere can access a table using either the RESTful OData protocol or the Windows Azure Storage Client library.

Each partition in a table holds some number of *entities*, each containing as many as 255 *properties*. Every property has a name, a type (such as Binary, Bool, DateTime, Int, or String), and a value. Unlike relational storage, these tables have no fixed schema, and so different entities in the same table can contain properties with different types. One entity might have just a String property containing a name, for example, while another entity in the same table has two Int properties containing a customer ID number and a credit rating.

To identify a particular entity within a table, an application provides that entity's key. The key has two parts: a *partition key* that identifies a specific partition and a *row key* that identifies an entity within that partition. In Figure 6, for example, the client requests the entity with partition key A and row key 3, and Table Storage returns that entity, including all of the properties it contains.

This structure lets tables be big—a single table can contain up to 100 terabytes of data—and it allows fast access to the data they contain. It also brings limitations, however. For example, there's no support for transactional updates that span tables or even partitions in a single table. A set of updates to a table can only be grouped into an atomic transaction if all of the entities involved are in the same partition. There's also no way to query a table based on the value of its properties, nor is there support for joins across multiple tables. And unlike relational databases, tables have no support for stored procedures.

Windows Azure Table Storage is a good choice for applications that need fast, cheap access to large amounts of loosely structured data. For example, an Internet application that stores profile information for lots of users might use tables. Fast access is important in this situation, and the application probably doesn't need the full power of SQL. Giving up this functionality to gain speed and size can sometimes make sense, and so Table Storage is just the right solution for some problems.

Hadoop

Organizations have been building data warehouses for decades. These collections of information, most often stored in relational tables, let people work with and learn from data in many different ways. With SQL Server, for instance, it's common to use tools such as SQL Server Analysis Services to do this.

But suppose you want to do analysis on non-relational data. Your data might take many forms: information from sensors or RFID tags, log files in server farms, clickstream data produced by web applications, images from medical diagnostic devices, and more. This data might also be really big, too big to be used effectively with a traditional data warehouse. Big data problems like this, rare just a few years ago, have now become quite common.

To analyze this kind of big data, our industry has largely converged on a single solution: the open-source technology Hadoop. Hadoop runs on a cluster of physical or virtual machines, spreading the data it works on across those machines and processing it in parallel. The more machines Hadoop has to use, the faster it can complete whatever work it's doing.

This kind of problem is a natural fit for the public cloud. Rather than maintaining an army of on-premises servers that might sit idle much of the time, running Hadoop in the cloud lets you create (and pay for) VMs only when you need them. Even better, more and more of the big data that you want to analyze with Hadoop is created in the cloud, saving you the trouble of moving it around. To

help you exploit these synergies, Microsoft provides a Hadoop service on Windows Azure. Figure 7 shows the most important components of this service.

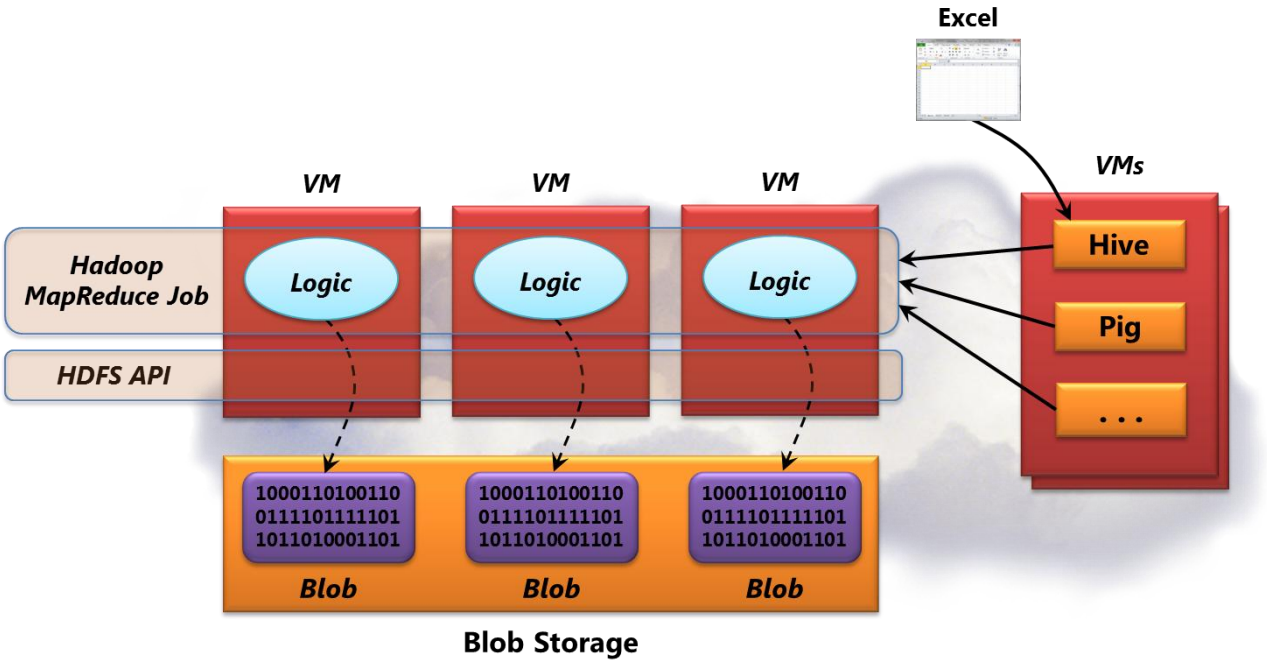

Figure 7: Hadoop on Windows Azure runs MapReduce jobs that process data in parallel using multiple virtual machines.

To use Hadoop on Windows Azure, you first ask this cloud platform to create a Hadoop cluster, specifying the number of VMs you need. Setting up a Hadoop cluster yourself is a non-trivial task, and so letting Windows Azure do it for you makes sense. When you're done using the cluster, you shut it down. There's no need to pay for compute resources that you aren't using.

A Hadoop application, commonly called a *job*, uses a programming model known as *MapReduce*. As the figure shows, the logic for a MapReduce job runs simultaneously across many VMs. By processing data in parallel, Hadoop can analyze data much more rapidly than single-machine solutions.

On Windows Azure, the data a MapReduce job works on is typically kept in blob storage. In Hadoop, however, MapReduce jobs expect data to be stored in the *Hadoop Distributed File System (HDFS)*. HDFS is similar to Blob Storage in some ways; it replicates data across multiple physical servers, for example. Rather than duplicate this functionality, Hadoop on Windows Azure instead exposes Blob Storage through the HDFS API, as the figure shows. While the logic in a MapReduce job thinks it's accessing ordinary HDFS files, the job is in fact working with data streamed to it from blobs. And to support the case where multiple jobs are run over the same data, Hadoop on Windows Azure also allow copying data from blobs into full HDFS running in the VMs.

MapReduce jobs are commonly written in Java today, an approach that Hadoop on Windows Azure supports. Microsoft has also added support for creating MapReduce jobs in other languages, including C#, F#, and JavaScript. The goal is to make this big data technology more easily accessible to a larger group of developers.

Along with HDFS and MapReduce, Hadoop includes other technologies that let people analyze data without writing a MapReduce job themselves. For example, Pig is a high-level language designed for analyzing big data, while Hive offers a SQL-like language called HiveQL. Both Pig and Hive actually generate MapReduce jobs that process HDFS data, but they hide this complexity from their users. Both are provided with Hadoop on Windows Azure.

Microsoft also provides a HiveQL driver for Excel. Using an Excel add-in, business analysts can create HiveQL queries (and thus MapReduce jobs) directly from Excel, then process and visualize the results using PowerPivot and other Excel tools. Hadoop on Windows Azure includes other technologies as well, such as the machine learning libraries Mahout, the graph mining system Pegasus, and more.

Big data analysis is important, and so Hadoop is also important. By providing Hadoop as a managed service on Windows Azure, along with links to familiar tools such as Excel, Microsoft aims at making this technology accessible to a broader set of users.

More broadly, data of all kinds is important. This is why Windows Azure includes a range of options for data management and business analytics. Whatever application you're trying to create, it's likely that you'll find something in this cloud platform that will work for you.

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.