



DavidChappell
& Associates

THE WINDOWS AZURE PROGRAMMING MODEL

DAVID CHAPPELL

OCTOBER 2010

SPONSORED BY MICROSOFT CORPORATION

CONTENTS

Why Create a New Programming Model?	3
The Three Rules of the Windows Azure Programming Model.....	3
A Windows Azure Application is Built from One or More Roles.....	4
A Windows Azure Application Runs Multiple Instances of Each Role	5
A Windows Azure Application Behaves Correctly When Any Role Instance Fails	6
What the Windows Azure Programming Model Provides.....	7
Some Background: The Fabric Controller	7
The Benefits: Improved Administration, Availability, and Scalability	8
Implications of The Windows Azure Programming Model: What Else Changes?	10
Interactions with the Operating System.....	11
Interactions with Persistent Storage	11
Interactions among Role Instances.....	13
Moving Windows Server Applications to Windows Azure	14
Conclusion.....	16
For Further Reading	16
About the Author	16

WHY CREATE A NEW PROGRAMMING MODEL?

Millions of developers around the world know how to create applications using the Windows Server programming model. Yet applications written for Windows Azure, Microsoft's cloud platform, don't exactly use this familiar model. While most of a Windows developer's skills still apply, Windows Azure provides its own programming model.

Why? Why not just exactly replicate the familiar world of Windows Server in the cloud? Many vendors' cloud platforms do just this, providing virtual machines (VMs) that act like on-premises VMs. This approach, commonly called *Infrastructure as a Service (IaaS)*, certainly has value, and it's the right choice for some applications. Yet cloud platforms are a new world, offering the potential for solving today's problems in new ways. Instead of IaaS, Windows Azure offers a higher-level abstraction that's typically categorized as *Platform as a Service (PaaS)*. While it's similar in many ways to the on-premises Windows world, this abstraction has its own programming model meant to help developers build better applications.

The Windows Azure programming model focuses on improving applications in three areas:

- **Administration:** In PaaS technologies, the platform itself handles the lion's share of administrative tasks. With Windows Azure, this means that the platform automatically takes care of things such as applying Windows patches and installing new versions of system software. The goal is to reduce the effort—and the cost—of administering the application environment.
- **Availability:** Whether it's planned or not, today's applications usually have down time for Windows patches, application upgrades, hardware failures, and other reasons. Yet given the redundancy that cloud platforms make possible, there's no longer any reason to accept this. The Windows Azure programming model is designed to let applications be continuously available, even in the face of software upgrades and hardware failures.
- **Scalability:** The kinds of applications that people want to write for the cloud are often meant to handle lots of users. Yet the traditional Windows Server programming model wasn't explicitly designed to support Internet-scale applications. The Windows Azure programming model, however, was intended from the start to do this. Created for the cloud era, it's designed to let developers build the scalable applications that massive cloud data centers can support. Just as important, it also allows applications to scale down when necessary, letting them use just the resources they need.

Whether a developer uses an IaaS technology or a PaaS offering such as Windows Azure, building applications on cloud platforms has some inherent benefits. Both approaches let you pay only for the computing resources you use, for example, and both let you avoid waiting for your IT department to deploy servers. Yet important as they are, these benefits aren't the topic here. Instead, the focus is entirely on making clear what the Windows Azure programming model is and what it offers.

THE THREE RULES OF THE WINDOWS AZURE PROGRAMMING MODEL

To get the benefits it promises, the Windows Azure programming model imposes three rules on applications:

- A Windows Azure application is built from one or more roles.

- A Windows Azure application runs multiple instances of each role.
- A Windows Azure application behaves correctly when any role instance fails.

It's worth pointing out that Windows Azure can run applications that don't follow all of these rules—it doesn't actually enforce them. Instead, the platform simply assumes that every application obeys all three. Still, while you might choose to run an application on Windows Azure that violates one or more of the rules, be aware that this application isn't actually using the Windows Azure programming model. Unless you understand and follow the model's rules, the application might not run as you expect it to.

A WINDOWS AZURE APPLICATION IS BUILT FROM ONE OR MORE ROLES

Whether an application runs in the cloud or in your data center, it can almost certainly be divided into logical parts. Windows Azure formalizes these divisions into *roles*. A role includes a specific set of code, such as a .NET assembly, and it defines the environment in which that code runs. Windows Azure today lets developers create three different kinds of roles:

- **Web role:** As the name suggests, Web roles are largely intended for logic that interacts with the outside world via HTTP. Code written as a Web role typically gets its input through Internet Information Services (IIS), and it can be created using various technologies, including ASP.NET, Windows Communication Foundation (WCF), PHP, and Java.
- **Worker role:** Logic written as a Worker role can interact with the outside world in various ways—it's not limited to HTTP. For example, a Worker role might contain code that converts videos into a standard format or calculates the risk of an investment portfolio or performs some kind of data analysis.
- **Virtual Machine (VM) role:** A VM role runs an image—a virtual hard disk (VHD)—of a Windows Server 2008 R2 virtual machine. This VHD is created using an on-premises Windows Server machine, then uploaded to Windows Azure. Once it's stored in the cloud, the VHD can be loaded on demand into a VM role and executed.

All three roles are useful. The VM role was made available quite recently, however, and so it's fair to say that the most frequently used options today are Web and Worker roles. Figure 1 shows a simple Windows Azure application built with one Web role and one Worker role.

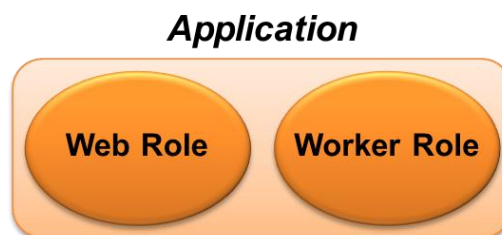


Figure 1: A Windows Azure application is built from one or more roles, such as the combination of Web and Worker role shown here.

This application might use a Web role to accept HTTP requests from users, then hand off the work these users request, such as reformatting a video file and making it available for viewing, to a Worker role. A

primary reason for this two-part breakdown is that dividing tasks in this way can make an application easier to scale.

It's also fine for a Windows Azure application to consist of just a single Web role or a single Worker role—you don't have to use both. A single application can even contain different kinds of Web and Worker roles. For example, an application might have one Web role that implements a browser interface, perhaps built using ASP.NET, and another Web role that exposes a Web services interface implemented using WCF. Similarly, a Windows Azure application that performed two different kinds of data analysis might define a distinct Worker role for each one. To keep things simple, though, we'll assume that the example application described here has just one Web role and one Worker role.

As part of building a Windows Azure application, a developer creates a service definition file that names and describes the application's roles. This file can also specify other information, such as the ports each role can listen on. Windows Azure uses this information to build the correct environment for running the application.

A WINDOWS AZURE APPLICATION RUNS MULTIPLE INSTANCES OF EACH ROLE

Every Windows Azure application consists of one or more roles. When it executes, an application that conforms to the Windows Azure programming model must run at least two copies—two distinct instances—of each role it contains. Each instance runs as its own VM, as Figure 2 shows.

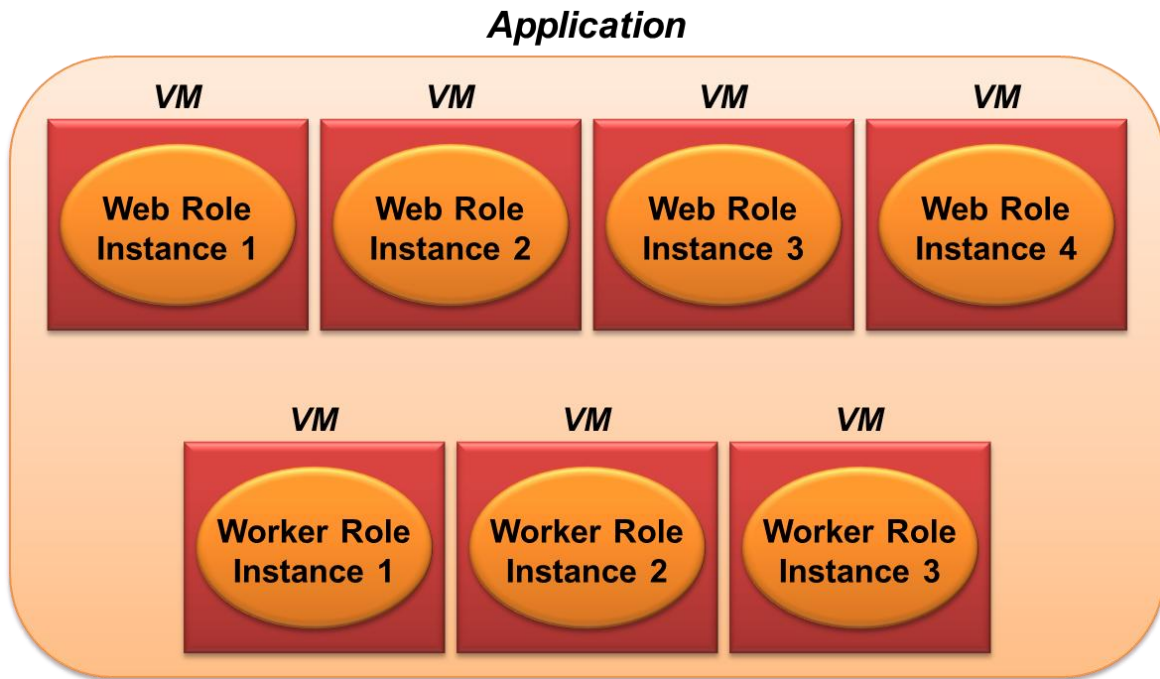


Figure 2: A Windows Azure application runs multiple instances of each role.

As described earlier, the example application shown here has just one Web role and one Worker role. A developer can tell Windows Azure how many instances of each role to run through a service configuration file (which is distinct from the service definition file mentioned in the previous section). Here, the

developer has requested four instances of the application's Web role and three instances of its Worker role.

Every instance of a particular role runs the exact same code. In fact, with most Windows Azure applications, each instance is just like all of the other instances of that role—they're interchangeable. For example, Windows Azure automatically load balances HTTP requests across an application's Web role instances. This load balancing doesn't support sticky sessions, so there's no way to direct all of a client's requests to the same Web role instance. Storing client-specific state, such as a shopping cart, in a particular Web role instance won't work, because Windows Azure provides no way to guarantee that all of a client's requests will be handled by that instance. Instead, this kind of state must be stored externally, as described later.

A WINDOWS AZURE APPLICATION BEHAVES CORRECTLY WHEN ANY ROLE INSTANCE FAILS

An application that follows the Windows Azure programming model must be built using roles, and it must run two or more instances of each of those roles. It must also behave correctly when any of those role instances fails. Figure 3 illustrates this idea.

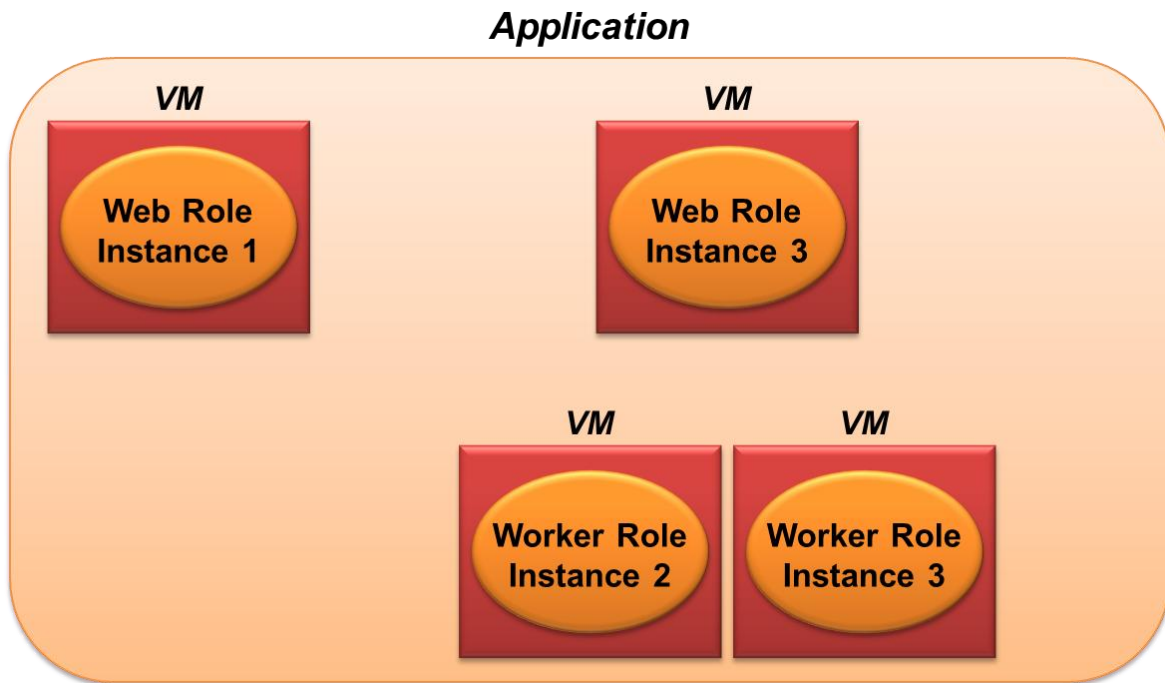


Figure 3: A Windows Azure application behaves correctly even when a role instance fails.

Here, the application shown in Figure 2 has lost two of its Web role instances and one of its Worker role instances. Perhaps the computers they were running on failed, or maybe the physical network connection to these machines has gone down. Whatever the reason, the application's performance is likely to suffer, since there are fewer instances to carry out its work. Still, the application remains up and functioning correctly.

If all instances of a particular role fail, an application will stop behaving as it should—this can't be helped. Yet the requirement to work correctly during partial failures is fundamental to the Windows Azure programming model. In fact, the service level agreement (SLA) for Windows Azure requires running at least two instances of each role. Applications that run only one instance of any role can't get the guarantees this SLA provides.

The most common way to achieve this is by making every role instance equivalent, as with load-balanced Web roles accepting user requests. This isn't strictly required, however, as long as the failure of a single role instance doesn't break the application. For example, an application might use a group of Worker role instances to cache data for Web role instances, with each Worker role instance holding different data. If any Worker role instance fails, a Web role instance trying to access the cached data it contained behaves just as it would if the data wasn't found in the cache (e.g., it accesses persistent storage to locate that data). The failure might cause the application to run more slowly, but as seen by a user, it still behaves correctly.

One more important point to keep in mind is that even though the sample application described so far contains only Web and Worker roles, all of these rules also apply to applications that use VM roles. Just like the others, every VM role must run at least two instances to qualify for the Windows Azure SLA, and the application must continue to work correctly if one of these instances fails. Even with VM roles, Windows Azure still provides a form of PaaS—it's not traditional IaaS.

WHAT THE WINDOWS AZURE PROGRAMMING MODEL PROVIDES

The Windows Azure programming model is based on Windows, and the bulk of a Windows developer's skills are applicable to this new environment. Still, it's not the same as the conventional Windows Server programming model. So why bother to understand it? How does it help create better applications? To answer these questions, it's first worth explaining a little more about how Windows Azure works. Once this is clear, understanding how the Windows Azure programming model can help create better software is simple.

SOME BACKGROUND: THE FABRIC CONTROLLER

Windows Azure is designed to run in data centers containing lots of computers. Accordingly, every Windows Azure application runs on multiple machines simultaneously. Figure 4 shows a simple example of how this looks.

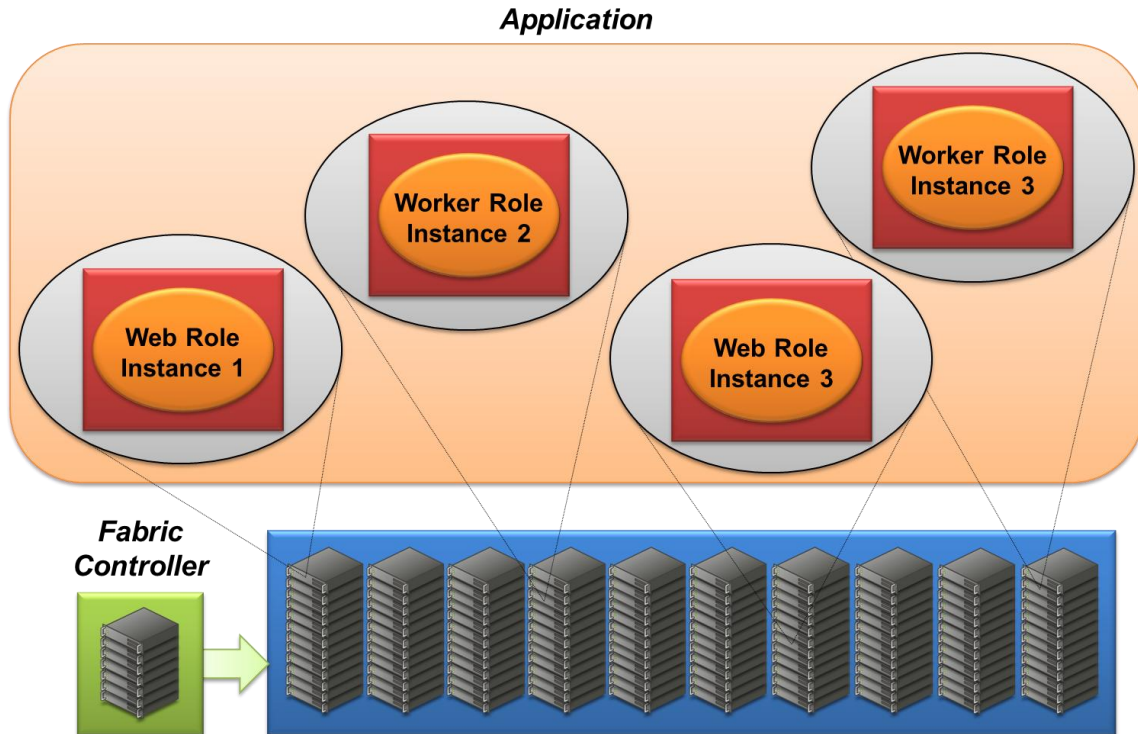


Figure 4: The Windows Azure fabric controller creates instances of an application’s roles on different machines, then monitors their execution.

As Figure 4 shows, all of the computers in a particular Windows Azure data center are managed by an application called the *fabric controller*. The fabric controller is itself a distributed application that runs across multiple computers.

When a developer gives Windows Azure an application to run, he provides the code for the application’s roles together with the service definition and service configuration files for this application. Among other things, this information tells the fabric controller how many instances of each role it should create. The fabric controller chooses a physical machine for each instance, then creates a VM on that machine and starts the instance running. As the figure suggests, the role instances for a single application are spread across different machines within this data center.

Once it’s created these instances, the fabric controller continues to monitor them. If an instance fails for any reason—hardware or software—the fabric controller will start a new instance for that role. While failures might cause an application’s instance count to temporarily drop below what the developer requested, the fabric controller will always start new instances as needed to maintain the target number for each of the application’s roles. And even though Figure 4 shows only Web and Worker roles, VM roles are handled in the same way, with each of the role’s instances running on a different physical machine.

THE BENEFITS: IMPROVED ADMINISTRATION, AVAILABILITY, AND SCALABILITY

Applications built using the Windows Azure programming model can be easier to administer, more available, and more scalable than those built on traditional Windows servers. These three attributes are worth looking at separately.

The administrative benefits of Windows Azure flow largely from the fabric controller. Like every operating system, Windows must be patched, as must other system software. In on-premises environments, doing this typically requires some human effort. In Windows Azure, however, the process is entirely automated: The fabric controller handles updates for Web and Worker role instances (although not for VM role instances). When necessary, it also updates the underlying Windows servers those VMs run on. The result is lower costs, since administrators aren't needed to handle this function.

Lowering costs by requiring less administration is good. Helping applications be more available is also good, and so the Windows Azure programming model helps improve application availability in several ways. They are the following:

- Protection against hardware failures. Because every application is made up of multiple instances of each role, hardware failures—a disk crash, a network fault, or the death of a server machine—won't take down the application. To help with this, the fabric controller doesn't choose machines for an application's instances at random. Instead, different instances of the same role are placed in different *fault domains*. A fault domain is a set of hardware—computers, switches, and more—that share a single point of failure. (For example, all of the computers in a single fault domain might rely on the same switch to connect to the network.) Because of this, a single hardware failure can't take down an entire application. The application might temporarily lose some instances, but it will continue to behave correctly.
- Protection against software failures. Along with hardware failures, the fabric controller can also detect failures caused by software. If the code in an instance crashes or the VM in which it's running goes down, the fabric controller will start either just the code or, if necessary, a new VM for that role. While any work the instance was doing when it failed will be lost, the new instance will become part of the application as soon as it starts running.
- The ability to update applications with no application downtime. Whether for routine maintenance or to install a whole new version, every application needs to be updated. An application built using the Windows Azure programming model can be updated while it's running—there's no need to take it down. To allow this, different instances for each of an application's roles are placed in different *update domains* (which aren't the same as the fault domains described earlier). When a new version of the application needs to be deployed, the fabric controller can shut down the instances in just one update domain, update the code for these, then create new instances from that new code. Once those instances are running, it can do the same thing to instances in the next update domain, and so on. While users might see different versions of the application during this process, depending on which instance they happen to interact with, the application as a whole remains continuously available.
- The ability to update Windows and other supporting software with no application downtime. The fabric controller assumes that every Windows Azure application follows the three rules listed earlier, and so it knows that it can shut down some of an application's instances whenever it likes, update the underlying system software, then start new instances. By doing this in chunks, never shutting down all of a role's instances at the same time, Windows and other software can be updated beneath a continuously running application.

Availability is important for most applications—software isn't useful if it's not running when you need it—but scalability can also matter. The Windows Azure programming model helps developers build more scalable applications in two main ways:

- Automatically creating and maintaining a specified number of role instances. As already described, a developer tells Windows Azure how many instances of each role to run, and the fabric controller creates and monitors the requested instances. This makes application scalability quite straightforward: Just tell Windows Azure what you need. Because this cloud platform runs in very large data centers, getting whatever level of scalability an application needs isn't generally a problem.
- Providing a way to modify the number of executing role instances for a running application: For applications whose load varies, scalability is more complicated. Setting the number of instances just once isn't a good solution, since different loads can make the ideal instance count go up or down significantly. To handle this situation, Windows Azure provides both a Web portal for people and an API for applications to allow changing the desired number of instances for each role while an application is running.

Making applications simpler to administer, more available, and more scalable is useful, and so using the Windows Azure programming model generally makes sense. But as mentioned earlier, it's possible to run applications on Windows Azure that don't follow this model. Suppose, for example, that you build an application using a single role (which is permitted) but then run only one instance of that role (violating the second and third rules). You might do this to save money, since Windows Azure charges separately for each running instance. Anybody who chooses this option should understand, however, that the fabric controller won't know that his application doesn't follow all three rules. It will shut down this single instance at unpredictable times to patch the underlying software, then restart a new one. To users, this means that the application will go down from time to time, since there's no other instance to take over. This isn't a bug in Windows Azure; it's a fundamental aspect of how the technology works.

Getting all of the benefits that Windows Azure offers requires conforming to the rules of its programming model. Moving existing applications from Windows Server to Windows Azure can require some work, a topic addressed in more detail later in this paper. For new applications, however, the argument for using the Windows Azure model is clear. Why not build an application that costs less to administer? Why not build an application that need never go down? Why not build an application that can easily scale up and down? Over time, it's reasonable to expect more and more applications to be created using the Windows Azure programming model.

IMPLICATIONS OF THE WINDOWS AZURE PROGRAMMING MODEL: WHAT ELSE CHANGES?

Building applications for Windows Azure means following the three rules of its programming model. Following these rules isn't enough, though—other parts of a developer's world must also adjust. The changes the Windows Azure programming model brings to the broader development environment can be grouped into three areas:

- How role instances interact with the operating system.
- How role instances interact with persistent storage.

- How role instances interact with other role instances.

This section looks at all three.

INTERACTIONS WITH THE OPERATING SYSTEM

For an application running on a typical Windows Server machine, the administrator of that machine is in control. She can reboot VMs or the machine they run on, install Windows patches, and do whatever else is required to keep that computer available. In Windows Azure, however, all of the servers are owned by the fabric controller. It decides when VMs or machines should be rebooted, and for Web and Worker roles (although not for VM roles), the fabric controller also installs patches and other updates to the system software in every instance.

This approach has real benefits, as already described. It also creates restrictions, however. Because the fabric controller owns the physical and virtual machines that Windows Azure applications use, it's free to do whatever it likes with them. This implies that letting a Windows Azure application modify the system it runs on—letting it run in administrator mode rather than user mode—presents some challenges. Since the fabric controller can modify the operating system at will, there's no guarantee that changes a role instance makes to the system it's running on won't be overwritten. Besides, the specific virtual (and physical) machines an application runs in change over time. This implies that any changes made to the default local environment must be made each time a role instance starts running.

In its first release, Windows Azure simply didn't allow applications to modify the systems they ran on—applications only ran in user mode. This restriction has been relaxed—both Web and Worker roles now give developers the option to run applications in admin mode—but the overall programming model hasn't changed. Anybody creating a Windows Azure application needs to understand what the fabric controller is doing, then design applications accordingly.

INTERACTIONS WITH PERSISTENT STORAGE

Applications aren't just code—they also use data. And just as the programming model must change to make applications more available and more scalable, the way data is stored and accessed must also change. The big changes are these:

- Storage must be external to role instances. Even though each instance is its own VM with its own file system, data stored in those file systems isn't automatically made persistent. If an instance fails, any data it contains may be lost. This implies that for applications to work correctly in the face of failures, data must be stored persistently outside role instances. Another role instance can now access data that otherwise would have been lost if that data had been stored locally on a failed instance.
- Storage must be replicated. Just as a Windows Azure application runs multiple role instances to allow for failures, Windows Azure storage must provide multiple copies of data. Without this, a single failure would make data unavailable, something that's not acceptable for highly available applications.
- Storage must be able to handle very large amounts of data. Traditional relational systems aren't necessarily the best choice for very large data sets. Since Windows Azure is designed in part for massively scalable applications, it must provide storage mechanisms for handling data at this scale.

To allow this, the platform offers *blobs* for storing binary data along with a non-SQL approach called *tables* for storing large structured data sets.

Figure 5 illustrates these three characteristics, showing how Windows Azure storage looks to an application.

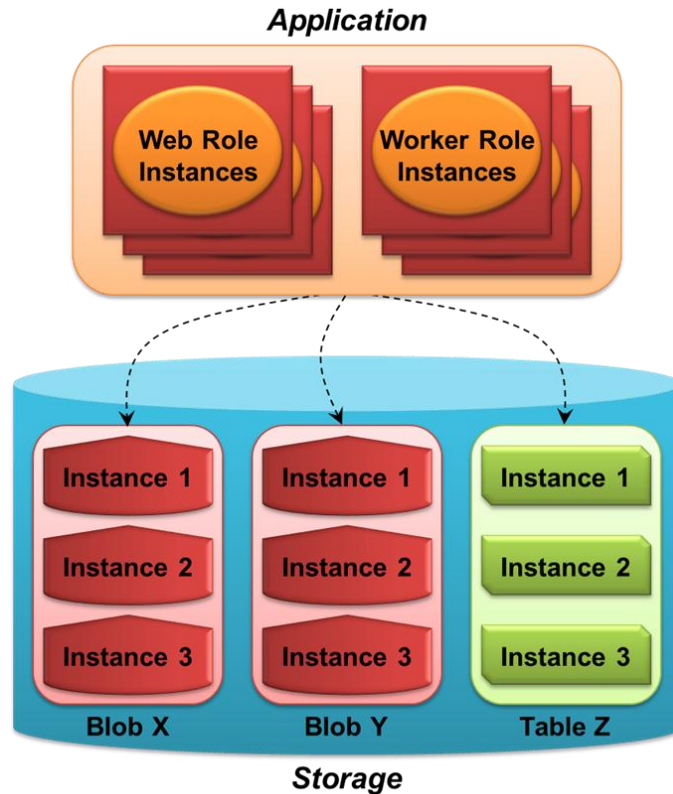


Figure 5: While applications see a single copy, Windows Azure storage replicates all blobs and tables three times.

In this example, a Windows Azure application is using two blobs and one table from Windows Azure storage. The application sees each blob and table as a single entity, but under the covers, Windows Azure storage actually maintains three instances of each one. These copies are spread across different physical machines, and as with role instances, those machines are in different fault domains. This improves the application’s availability, since data is still accessible even when some copies are unavailable. And because persistent data is stored outside any of the application’s role instances, an instance failure loses only whatever data it was using at the moment it failed.

The Windows Azure programming model requires an application to behave correctly when a role instance fails. To do this, every instance in an application must store all persistent data in Windows Azure storage or another external storage mechanism (such as SQL Azure, Microsoft’s cloud-based service for relational data). There’s one more option worth mentioning, however: Windows Azure drives. As already described, any data an application writes to the local file system of its own VM can be lost when that VM stops running. Windows Azure drives change this, using a blob to provide persistent storage for the file system of a particular instance. These drives have some limitations—only one instance at a time is allowed to

both read from and write to a particular Windows Azure drive, for example, with all other instances in this application allowed only read access—but they can be useful in some situations.

INTERACTIONS AMONG ROLE INSTANCES

When an application is divided into multiple parts, those parts commonly need to interact with one another. In a Windows Azure application, this is expressed as communication between role instances. For example, a Web role instance might accept requests from users, then pass those requests to a Worker role instance for further processing.

The way this interaction happens isn't identical to how it's done with ordinary Windows applications. Once again, a key fact to keep in mind is that, most often, all instances of a particular role are equivalent—they're interchangeable. This means that when, say, a Web role instance passes work to a Worker role instance, it shouldn't care which particular instance gets the work. In fact, the Web role instance shouldn't rely on instance-specific things like a Worker role instance's IP address to communicate with that instance. More generic mechanisms are required.

The most common way for role instances to communicate in Windows Azure applications is through Windows Azure queues. Figure 6 illustrates the idea.

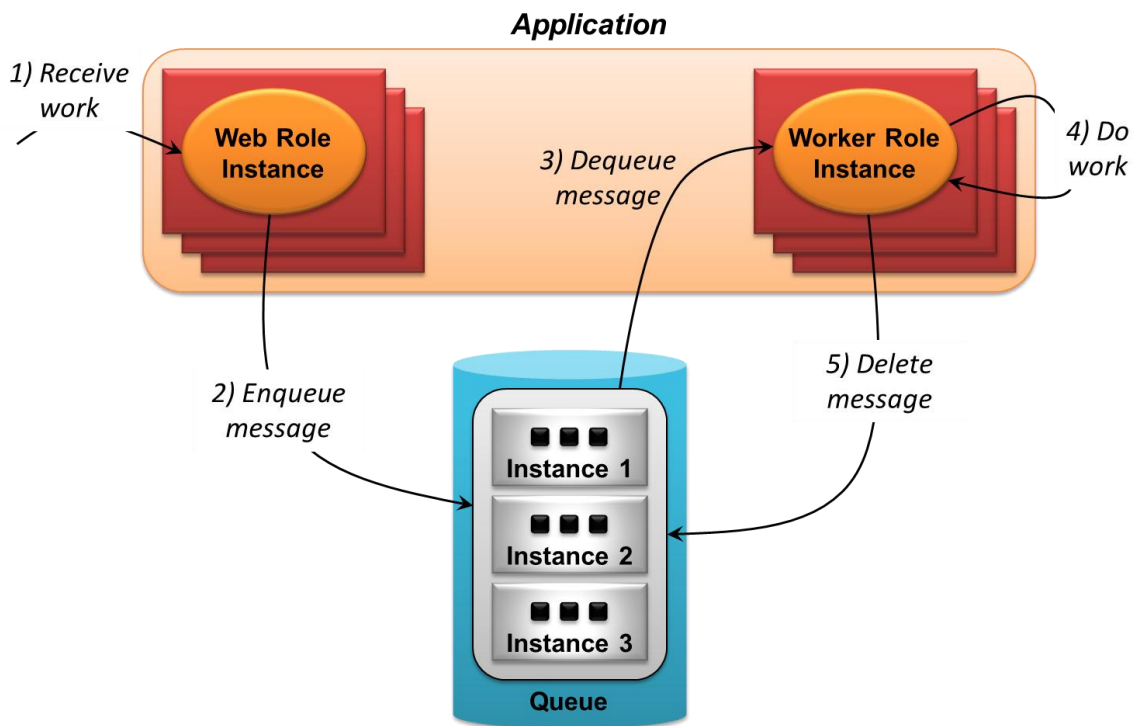


Figure 6: Role instances can communicate through queues, each of which replicates the messages it holds three times.

In the example shown here, a Web role instance gets work from a user of the application, such as a person making a request from a browser (step 1). This instance then creates a message containing this work and writes it into a Windows Azure queue (step 2). These queues are implemented as part of Windows Azure storage, and so like blobs and tables, each queue is replicated three times, as the figure

shows. As usual, this provides fault-tolerance, ensuring that the queue's messages are still available if a failure occurs.

Next, a Worker role instance reads the message from the queue (step 3). Notice that the Web role instance that created this message doesn't care which Worker role instance gets it—in this application, they're all equivalent. That Worker role instance does whatever work the message requires (step 4), then deletes the message from the queue (step 5).

This last step—explicitly removing the message from the queue—is different from what on-premises queuing technologies typically do. In Microsoft Message Queuing (MSMQ), for example, an application can do a read inside an atomic transaction. If the application fails before completing its work, the transaction aborts, and the message automatically reappears on the queue. This approach guarantees that every message sent to an MSMQ queue is delivered exactly once in the order in which it was sent.

Windows Azure queues don't support transactional reads, and so they don't guarantee exactly-once, in-order delivery. In the example shown in Figure 6, for instance, the Worker role instance might finish processing the message, then crash just before it deletes this message from the queue. If this happens, the message will automatically reappear after a configurable timeout period, and another Worker role instance will process it. Unlike MSMQ, Windows Azure queues provide at-least-once semantics: A message might be read and processed one or more times.

This raises an obvious question: Why don't Windows Azure queues support transactional reads? The answer is that transactions require locking, and so they necessarily slow things down (especially with the message replication provided by Windows Azure queues). Given the primary goals of the platform, its designers opted for the fastest, most scalable approach.

Most of the time, queues are the best way for role instances within an application to communicate. It's also possible for instances to interact directly, however, without going through a queue. To allow this, Windows Azure provides an API that lets an instance discover all other instances in the same application that meet specific requirements, then send a request directly to one of those instances. In the most common case, where all instances of a particular role are equivalent, the caller should choose a target instance randomly from the set the API returns. This isn't always true—maybe a Worker role implements an in-memory cache with each role instance holding specific data, and so the caller must access a particular one. Most often, though, the right approach is to treat all instances of a role as interchangeable.

MOVING WINDOWS SERVER APPLICATIONS TO WINDOWS AZURE

Anybody building a new Windows Azure application should follow the rules of the Windows Azure programming model. To move an existing application from Windows Server to Windows Azure, however, that application should also be made to follow the same rules. In addition, the application might need to change how it interacts with the operating system, how it uses persistent storage, and the way its components interact with each other.

How easy it is to make these changes depends on the application. Here are a few representative examples:

- *An ASP.NET application with multiple load-balanced instances that share state stored in SQL Server.* This kind of application typically ports easily to Windows Azure, with each instance of the original application becoming an instance of a Web or Worker role. Applications like this don't use sticky sessions, which helps make them a good fit for Windows Azure. (Using ASP.NET session state is acceptable, however, since Windows Azure provides an option to store session state persistently in Windows Azure Storage tables.) And moving an on-premises SQL Server database to SQL Azure is usually straightforward.
- *An ASP.NET application with multiple instances that maintains per-instance state and relies on sticky sessions.* Because it maintains client-specific state in each instance between requests, this application will need some changes. Windows Azure doesn't support sticky sessions, and so making the application run on this cloud platform will require redesigning how it handles state.
- *A Silverlight or Windows Presentation Foundation (WPF) client that accesses WCF services running in a middle tier.* If the services don't maintain per-client state between calls, moving them to Windows Azure is straightforward. The client will continue to run on user desktops, as always, but it will now call services running on Windows Azure. If the current services do maintain per-client state, however, they'll need to be redesigned.
- *An application with a single instance running on Windows Server that maintains state on its own machine.* Whether the clients are browsers or something else, many enterprise applications are built this way today, and they won't work well on Windows Azure without some redesign. It might be possible to run this application unchanged in a single VM role instance, but its users probably won't be happy with the results. For one thing, the Windows Azure SLA doesn't apply to applications with only a single instance. Also, recall that the fabric controller can at any time reboot the machine on which this instance runs to update that machine's software. The application has no control over when this happens; it might be smack in the middle of a workday. Since there's no second instance to take over—the application wasn't built to follow the rules of the Windows Azure programming model—it will be unavailable for some period of time, and so anybody using the application will have their work interrupted while the machine reboots. Even though the VM role makes it easy to move a Windows Server binary to Windows Azure, this doesn't guarantee that the application will run successfully in its new home. The application must also conform to the rules of the Windows Azure programming model.
- *A Visual Basic 6 application that directly accesses a SQL Server database, i.e., a traditional client/server application.* Making this application run on Windows Azure will most likely require rewriting at least the client business logic. While it might be possible to move the database (including any stored procedures) to SQL Azure, then redirect the clients to this new location, the application's desktop component won't run as is on Windows Azure. Windows Azure doesn't provide a local user interface, and it also doesn't support using Remote Desktop Services (formerly Terminal Services) to provide remote user interfaces.

Windows Azure can help developers create better applications. Yet the improvements it offers require change, and so moving existing software to this new platform can take some effort. Making good decisions requires understanding both the potential business value and any technical challenges that moving an application to Windows Azure might bring.

CONCLUSION

Cloud platforms are a new world, and they open new possibilities. Reflecting this, the Windows Azure programming model helps developers create applications that are easier to administer, more available, and more scalable than those built in the traditional Windows Server environment. Doing this requires following three rules:

- A Windows Azure application is built from one or more roles.
- A Windows Azure application runs multiple instances of each role.
- A Windows Azure application behaves correctly when any role instance fails.

Using this programming model successfully also requires understanding the changes it brings to how applications interact with the operating system, use persistent storage, and communicate between role instances. For developers willing to do this, however, the value is clear. While it's not right for every scenario, the Windows Azure programming model can be useful for anybody who wants to create easier to administer, more available, and more scalable applications.

FOR FURTHER READING

Introducing Windows Azure: <http://go.microsoft.com/?linkid=9682907>

Introducing the Windows Azure Platform: <http://go.microsoft.com/?linkid=9752185>

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 technology.