DZone Refcardz

brought to you by... Microsoft

CONTENTS INCLUDE:

Cloud Computing with Windows Azure Platform

By Brian H. Prince

ABOUT WINDOWS AZURE

The Windows Azure Platform is the Microsoft cloud computing platform. It is comprised of several components:

- **Compute service:** servers to run your code
- **Storage service:** to store unstructured data
- **SQL Azure:** to store structured data
- **Windows Azure AppFabric:** for security and connectivity

Some examples of where to use it include:

- **Social:** Backend Facebook / social apps
- **Mobile:** One storage and services solution for iPhone / Android / Win Phone apps
- Web sites that will get spikes in traffic
- **HPC:** Simulations, modeling, etc.
- A "managed home" for Access databases

THREE "ROLES" BRING YOUR SERVICES TO LIFE

The Fabric Controller controls all of the servers running in Windows Azure. To deploy code, you have to tell the Fabric Controller how you want the code to be hosted. Roles are server templates for your code to run in. There are three server templates, or roles, that you can use to host and run your code.

All roles are stateless in nature. While changes can be made to your server at runtime (under some circumstances), all changes will be lost during a reboot or a server move.

- **Web Role:** The web role is just like a normal web server. It runs IIS7 and allows you to host up to five HTTP/S ports. You can host several web applications with the same role using host headers. This role runs Windows Server 2008.
- **Worker Role:** The worker role is a lot like the web role. It runs Windows Server 2008, but it does not run IIS. You can host any number of services using any protocol that uses TCP. Worker roles are commonly used for back-end processes and for hosting many web services.
- **VM Role:** The VM Role is a little different from the web role and worker role. The VM role is any server image that you create and upload. It must run Windows Server 2008 R2. You can customize the server image to your needs. Windows licensing is included in the Windows Azure pricing, so you don't have to use your licenses on servers you are running in Windows Azure. This role type is used most often to host software that otherwise wouldn't work in Windows Azure.

Instances: How to scale out

When you deploy a role, you determine how many servers should run that role template. These are called instances. You control how many instances are running and what size they are.

You select the size of the instance for your role in the ServiceDefinition.csdef file. While you can dynamically change how MANY instances you have, you cannot dynamically change the SIZE of your instances.

The sizes available follow a similar pattern, with one exception. All dedicated cores start with Small instances. This is one dedicated CPU core with about 2GB of RAM. The next three sizes just double the number of cores and RAM available.

They are Medium (2core/4GB), Large (4 core/7GB), and Extra Large (8 core/14GB).

The one exception is the new Extra Small instance. This is a shared core, running at 1GHz with 768MB for RAM. It is much cheaper than the other sizes.

CONFIGURING YOUR CLOUD SERVICE

Configuration of your cloud application is handled with two separate files:

- **ServiceDefinition.csdef:** Tells the Fabric Controller the shape of your service ("service" is a generic term for web app, application, web service, etc.), including what ports to allow, how the instances are allowed to communicate, what code they should each run, etc. If you change this file, you must redeploy your application to the cloud.
- **ServiceConfiguration.cscfg:** Includes settings that are needed by the Fabric Controller at runtime to set up and run your instances. You can also store configuration data in this file. You can update this file at any time by providing a new file that includes the changes you want. Depending on what settings you change, you may cause an app restart of your servers.

Dialing up (and down) the number of instances

A deployed application can include several roles. Instances of each role are created by the Fabric Controller according to your ServiceConfiguration.cscfg file. You can change how many instances you want by changing the config file. Each role in your application can have a different amount of instances assigned to it.

When you increase the number of instances (perhaps during a busy period for your application), the Fabric Controller will bring it online. This will not impact any of the other already running instances. Bringing an instance online takes about 10 - 20 minutes.

When you decrease the number of instances (perhaps the busy time has passed), the Fabric Controller will select some instances

to be stopped. The OnStop() method of your RoleEntryPoint class (typically found in webrole.cs) will be called once the server has been taken offline, but before it is shut down. This will give you an opportunity to clean up any last-minute issues before the instance is fully terminated.

When to allocate storage "local" to your compute Instances

You can configure each role to allocate a certain amount of local storage. This will be storage space located on the server instance itself. You can define how much space you want to allocate, and you can configure more than one allocation per role.

Local storage is most often used to allow for legacy code that needs to use the local file system to work. The limitation is that it is considered volatile; any data put there is likely to be lost in the event of a server move or restart. You should only put data in local storage that you can lose. Most applications will rely on the Windows Azure Storage system (blobs, queues, and tables, which are explained later in this Refcard).

You must declare local storage in your ServiceDefinition.csdef file. This sample defines a 20MB storage area named "TempFiles". This local storage area will be cleaned out if the server is rebooted.

```
<LocalResources>
  <LocalStorage name="TempFiles" cleanOnRoleRecycle="true"
                sizeInMB="20" />
</LocalResources>
```

When your instance is started up, a folder will be created for you with a storage limit set to what you defined in your configuration. You will need to get a reference to the path this folder was created in.

```
LocalResource myStorage = RoleEnvironment.GetLocalResource("TempFiles");
String localPath = myStorage.RootPath;
```

From here, you can use the localPath variable as a path in your normal file-handling code.

Declare Endpoints

Roles can declare endpoints so that they can receive network traffic. Web roles have a single endpoint configured with HTTP over port 80 as a default. You can also enable an endpoint for HTTPS over port 443.

A worker role doesn't have any endpoints defined by default. There are two types of endpoints.

- **Input endpoint:** When the instance is started, an input endpoint will be published to the load balancer and configured on the firewall. This will allow traffic from the Internet with the right protocol on the right port to be routed to your instances.

  To declare an input endpoint, you need to define it in your ServiceDefinition.csdef.

```
<Endpoints>
  <InputEndpoint name="Endpoint1" protocol="tcp" port="10000" />
</Endpoints>
```

- **Internal endpoint:** The internal endpoint will not be published to the load balancer and is used for direct communication to your role instances. This endpoint can only be used by other instances in your application.

  You declare an internal endpoint in much the same way as an input endpoint, but you do not declare a port number. The port number will be defined at runtime.

```
<Endpoints>
  <InternalEndpoint name="Endpoint2" protocol="tcp" />
</Endpoints>
```

You reference both endpoints in a worker role the same way. You have to create a serviceHost and then configure it at runtime by reading the RoleEnvironment configuration to get the endpoint details.

```
RoleInstanceEndpoint myInputEndPoint = RoleEnvironment.CurrentRoleInstance.
InstanceEndpoints["Endpoint1"];

serviceHost.AddServiceEndpoint(
            typeof(IContract),
            binding,
            String.Format("net.tcp://{0}
             /MyService", myInputEndpoint.IPEndpoint));
```

The IPEndpoint property of the RoleInstanceEndpoint class represents both the IP address and port number of the endpoint.

NON-RELATIONAL STORAGE FOR MASSIVE DATA

Windows Azure Storage provides three ways to store your unstructured data: BLOBs, queues, and tables.

All three forms use the same backend infrastructure. They are all accessible through the .NET Storage Client Library (provided with the SDK) or directly with REST. Because you can access all storage with REST, the code using the storage doesn't have to be running in Windows Azure. It could be running on a phone, in a browser, or on a server in your data center.

Windows Azure stores all data in triplicate. When data is written, a response is not returned until two replicas write the data successfully. The third replica is written asynchronously.

All data lives in a storage account, which can contain any combination of storage types. A single storage account can hold up to 100TB of data and has an account name and two account keys. The account name is like your user name, and the account key is like your password. You should never share these with anyone.

If you are using the .NET Storage Client Library, then you can store your credentials in your ServiceConfiguration.csdef and it will be automatically picked up.

```
<Setting name="DataConnectionString" value="DefaultEndpointsProtocol=https;
AccountName=[YOUR_ACCOUNT_NAME];AccountKey=[YOUR_ACCOUNT_KEY]" />
```

Whenever you add something to your ServiceConfiguration.cscfg file, you need to declare it in your ServiceDefinition.csdef file.

```
<ConfigurationSettings>
   <Setting name="DataConnectionString"/>
</ConfigurationSettings>
```

The first object you will work with is CloudStorageAccount. This represents the storage account you created in the portal and contains the credentials needed to use storage. From the storage account, you create a client object. There is one type of client object for each storage type.

Use BLOBs for video, image and other digital files

BLOB stands for Binary Large Object. You can think of BLOB storage as a file system in the cloud. A storage account can contain any number of BLOB containers. Each container is like a root folder. Containers do not contain child containers. A container will have one of several different access levels set:

- **Private:** This is the default setting. All reads and writes must use the account name and account key.
- **Full public read:** This provides full anonymous read permissions. The reader can read BLOBs and list the container's contents.
- **Public read only:** This is similar to Full Public Read, but the user does not have permissions to list the contents of the container.

To connect to and work with BLOBs, you need to create a CloudStorageAccount object and then a CloudBlobClient object. This will let you get a reference to a BLOB container, which is like a root folder. All of the storage client objects have you get references to objects before they are created in the cloud. For example, you will make a reference to a container and then call container.CreateIfNotExists() to create the container. To upload a file, you would create a CloudBlockBlob object from your

container reference and then call one of the upload methods. As an example, use the following code to create a container and upload a local file.

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
using Microsoft.WindowsAzure.ServiceRuntime;

CloudStorageAccount storageAccount = CloudStorageAccount.FromConfigurationSe
tting("DataConnectionString");

CloudBlobClient blobStorage = storageAccount.CreateCloudBlobClient();

CloudBlobContainer container = blobStorage.GetContainerReference("mydocs");

container.CreateIfNotExist();

CloudBlockBlob blob = container.GetBlockBlobReference("birthday.mpg");

blob.UploadFile(@"c:\tempfiles\birthday.mpg");
```

Queues will decouple front end from back end

Windows Azure Storage queues are very similar to queues you have probably used before. They are most commonly used to communicate from a front-end server to a back-end server. Queues are very handy in decoupling the front end from the back end of your system.

Queues allow you to send small 8KB messages from producers (commonly the front end) to consumers (commonly the back end). Queues are First In, First Out (FIFO). The first message in is the first message out.

Queues can store an essentially unlimited number of messages at one time. A queue can have any number of producers submitting messages and any number of consumers taking messages.

Because of the size limitation of a queue message, you will usually use a work ticket pattern. You will store the real data to be worked on in a BLOB container, or a table, and put a pointer to the data in the message.

Messages go through a specific lifecycle in the queue. A consumer will read a message and provide a timeout (defaults to 30 seconds and can be as long as 2 hours). The queue will mark the message as invisible and lock it for this period of time. Before the timeout expires, the reader of the message must call back with a delete message command. If it doesn't, then the queue assumes the consumer has failed, cancels the lock, and marks the message as visible on the queue again.

To work with a queue, you need to create a CloudStorageAccount and a CloudQueueClient object to manage queues and messages. To connect to and create a queue:

```
CloudStorageAccount storageAccount = CloudStorageAccount.FromConfigurationSe
tting("DataConnectionString");
CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();

CloudQueue q = queueClient.GetQueueReference("newordersqueue");
q.CreateIfNotExist();
```

To add a message to the queue, you simple call the AddMessage method of the queue object and pass in the string value of the message you want to send.

```
CloudQueueMessage theNewMessage = new CloudQueueMessage("shoppingca
rt:1337");
q.AddMessage(theNewMessage);
```

To get the next message off of the queue, you simply call GetMessage. You can get more than one message at a time. If the queue is empty, GetMessage will return a null.

```
CloudQueueMessage currentMsg;
currentMsg = q.GetMessage();
```

Finally, to delete a message you have finished working with, call DeleteMessage.

```
q.DeleteMessage(currentMsg);
```

Tables: up to 100TB of non-SQL data

You can have several tables per storage account. Each table can hold up to 100TB of data and is meant for highly scalable non-relational data. Each table is comprised of entities (like rows in a normal database), and each entity is comprised of many properties. Entities in the same table can have a different schema, giving Windows Azure tables a great deal of flexibility. Tables do not have any relationships with other tables; there are no joins or foreign keys.

Each entity in a table must have a RowKey and a PartitionKey property. These two properties together act as a sort of composite primary key for the entity. A property named TimeStamp is also required.

The entities in tables are grouped into partitions using the PartitionKey. Partitions are how the table service scales. As a particular partition becomes busy, it is spun out to a new storage server so that it has more resources to handle the requests. This could happen to all of your partitions at once, fanning out to different machines to make sure that the system is scaling to the demands put on it. Choosing a PartitionKey strategy is important when designing any tables that might require high performance and scale.

The RowKey is a unique row identifier for that row in its partition. Both the RowKey and the PartitionKey can be anything you want them to be, but it is best to keep them simple INTs or strings.

Tables can be accessed directly with REST or through the .NET Storage Client Library. Tables present an OData endpoint, which makes it easy to work with them as a data source.

To work with tables, you need a few working pieces. You need a class that represents your data; and in most modern architectures, this is the data transfer object (DTO) or plain old .NET object (PONO) that has just properties (i.e., no methods). This class needs to inherit from Microsoft.WindowsAzure.StorageClient. TableServiceEntity and must provide for the RowKey and PartitionKey values.

```
public class ShoppingCartEntry :
  Microsoft.WindowsAzure.StorageClient.TableServiceEntity
    {
        public ShoppingCartEntry(int _shoppingCartID)
        {
            PartitionKey = "carts";
            RowKey = _shoppingCartID;
        }

        public int CustomerID { get; set; }
        public string Sku { get; set; }
        public int quantity { get; set; }
    }
```

The Client Storage Library uses WCF Data Services to work with Windows Azure Tables, which means you will need a context class. This is a class that sits between the entity class (shown above) and the table itself. This is like any other WCF Data Services context class. This class must inherit from TableServiceContext. This base class provides all the plumbing you need.

```
public class ShoppingCartDataContext : TableServiceContext
    {
     public ShoppingCartDataContext  (string baseAddress,
StorageCredentials credentials)
      : base(baseAddress, credentials)
     { }

     public IQueryable<ShoppingCartEntry> ShoppingCartEntry
     {
      get
       {
        return this.CreateQuery<ShoppingCartEntry>("ShoppingCartEntry");
       }
     }
    }
```

Once you have these two classes, you can start working with the data you have in your table. For example, we will add a shopping cart called aNewShoppingCart. To start, you need to provide a storage account and a data context object. You should keep the data context class around as much as you can instead of creating a new object on each call.

```
CloudStorageAccount storageAccount = CloudStorageAccount.FromConfigurationSe
tting("DataConnectionString");
ShoppingCarDataContext context = new ShoppingCarDataContext
(storageAccount.TableEndpoint.AbsoluteUri, storageAccount.Credentials);
context.AddObject("ShoppingCartEntry", aNewShoppingCart);
context.SaveChanges();
```

You must always remember to call SaveChanges on the context object once you have made changes. If you don't, the changes are never sent to the cloud.

A similar approach is used for updating data.

```
aNewShoppingCart.Sku = "31415";
context.UpdateObject(aNewShoppingCart);
context.SaveChanges();
```

You can batch many operations against the context object before calling SaveChanges.

Querying against the table is easy using the context object and LINQ.

```
var results = from g in this.context.ShoppingCartEntry
              where g.Sku == "31415"
              select g;
```

This will return a list of objects that have their SKU numbers set to 31415.

ACCESS CONTROL AND SERVICE BUS

Windows Azure AppFabric is a set of services that help you connect and consume other services. It has its own SDK, separate from the Windows Azure SDK.

ACS

The Access Control Service (ACS) provides an easy-to-use authorization service for use in your application. You can use ACS instead of embedding code to authenticate users in your application.

ACS makes it easy to not only use your own authentication (by federating with your old on-premises identities) but also to authenticate against Google, Live, Yahoo, and Facebook.

Authentication tends to happen at the edge of an application, and authorization tends to happen in depth. Because of this model, it is usually easy to add ACS to your current authentication mechanism or to replace it outright.

The new code would authenticate to ACS and then produce the same output (a token, cookie, etc.) that your downstream authorization code already expects.

Service Bus

Another challenge with moving applications to the cloud is connecting them together with services or applications that are still on premises. A central key point to the Windows Azure platform is that you don't have to move everything to the cloud. It is easy to move only the parts to the cloud that make sense and provide ways to securely connect them to your on-premises components. This can be challenging at times because of network topologies. There are always a series of firewalls, proxies, and other networking gear that get in the way of this communication.

The Service Bus provides a way for any service consumer to connect with any service, regardless of network topology. A matching WCF relay binding is provided for almost every out-of-the-box WCF binding. By adding a new endpoint to your service with a relay binding, you will be publishing your service endpoint to the Service Bus in Windows Azure. Any service consumers that are configured to use this same relay binding will then be able to connect and use a Service Bus-based endpoint.

In the cloud, the Service Bus provides a relay. Think of this like bouncing a signal off a satellite. Both ends don't have to know how to reach each other; they just have to know how to reach the relay (or the satellite).

When services or consumers try to connect to the Service Bus, they have to authenticate with ACS to make sure that only people you want to connect are connecting with their services. Your service would still provide for whatever security concerns it might have.

RELATIONAL STORAGE FOR BUSINESS INSIGHTS

SQL Azure is essential SQL Server in the cloud. While this is not a perfect statement, it is the shortest explanation.

A SQL Server usually equates to a physical server, and that physical server is hosting several SQL databases. With Windows Azure, your subscription can have one SQL Azure server, which can hold as many databases as you want. But this SQL Azure server doesn't really exist; it is a simulated SQL Server that manages your service and the databases you want hosted.

To all normal SQL Server clients, the SQL Azure server appears to be a normal SQL Server. The SQL Azure service runs thousands of servers and manages your databases across this swarm of servers, moving and maintaining them as needed.

Each database can be up to 50GB in size. Many see this as a limitation, but it doesn't need to be. Many on-premises databases soon exceed the amount of resources available to them (perhaps they grew too big for their environment) and soon have to turn to partitioning or sharding to manage this growth. You would use these strategies to overcome this apparent limitation. Microsoft has also raised this limit on database size several times and could do so again.

As with the data in Windows Azure storage, your data in SQL Azure is stored in triplicate. This ensures that there is always a safe copy of your data.

You can create the following editions of an SQL Azure database:

- **Web Edition:** the smallest. You are limited to a 1GB or 5GB database.
- **Business Edition:** the bigger brother to the Web Edition. You can have databases that range from 1GB to 50GB in max size.

The only difference between the editions at this time is the maximum size of each database. In the future, Microsoft expects to add additional features to the business edition to provide more value.

Backup

Windows Azure does not provide any backup mechanisms for you. It does provide a highly available datacenter, but there isn't any cross-data center mechanisms for your data. There is currently no way to easily back up and restore your data. You have to rely on some other mechanisms to accomplish this. For example you could utilize the Bulk Copy Program (BCP) to copy your data once a night.

You could also use the Data Sync Service to synchronize your SQL Azure database with an on-premises SQL Server database over a secure connection.

Database Copy

Database Copy makes a transactionally consistent copy of your database. A copy (with a new name that you provide) will be made of your database on the same SQL Azure server. The copy can take a few moments to complete. This is a good approach to backing up a database before an upgrade, making it easy to restore in case of a failure or rollback. To start the copy, use the following command (or you can start it in the portal).

```
CREATE DATABASE ShoppingCartBackup AS COPY OF ShoppingCart
```

To monitor how the copy is going, which can take a while with a very large database, use the following command.

```
select name, state, state_desc from sys.databases
where name = 'ShoppingCartBackup'
select * from sys.dm_database_copies
where database_id = DB_ID('ShoppingCartBackup')
```

You can then download the databasecopy or move it to another SQL Azure server in a different Windows Azure datacenter.

SQL Azure Migration Wizard

The SQL Azure Migration Wizard is an open-source tool that can help you migrate your SQL Server database to SQL Azure. It will inspect your schema for data types and other features not supported in SQL Azure and help you migrate your data. You can find it at http://sqlazuremw.codeplex.com/. It is based on a highly configured system that makes it easy to tune it to your needs.

Harness one of the Planet's Largest CDNs

Windows Azure has 20 Content Delivery Network (CDN) cache servers around the world, and they leverage the Microsoft highly connected and fast global network to deliver that data to local users. CDN is the way that high-scale websites and providers cache their data around the globe. When a user tries to download some cached data, it is downloaded from the most local cache server instead of from a main server. This gives the user a faster download experience.

The Windows Azure CDN can cache the contents of your public BLOB container around the world. When you store a file in your public BLOB container, it will have a URL like the following: http://mydocs.blob.core.windows.net/videos/birthday.mpg

If your container is public, you can turn on CDN caching on the Windows Azure portal. Once you do this, you will receive a new URL for your BLOB container, such as: http://<guid>.vo.msecnd.net/videos/birthday.mpg

You will then use this new cached URL in your application. When the user tries to download the file, the CDN service will determine which cache server is closest and download it from there.

Files are not pushed to the cache servers until they are needed at that particular cache server. This reduces your costs. If the file is not in the cache when it is needed, it is retrieved over the network from your BLOB container.

The CDN servers have algorithms on when a file should be refreshed from your BLOB container. By default, a check is made on the first access after 72 hours. If the file has changed, a new file is downloaded. If a file hasn't changed, it will respond with the current file. You can control the frequency of these cache refreshes by setting a cache-control header on your BLOB.

SEEMLESS DEPLOYMENT FROM VISUAL STUDIO 2010

At some point, you will want to deploy your cloud application to Windows Azure. There are three ways to do this, and all three approaches take about the same amount of time.

- **Visual Studio:** You can use the Visual Studio tools to deploy from right within Visual Studio. This is the best choice when you want to easily deploy to a developer's subscription or to a test environment subscription. You will need to open the Server Explorer and configure the Windows Azure section with a management certificate and your Windows Azure subscription ID. Once this is configured, you can right click on the Windows Azure project in your Visual Studio solution and choose "publish." Visual Studio will then start the application. You can watch this process in the Visual Studio window titled "Windows Azure Activity Log".
- **Windows Azure Portal:** To deploy with the portal log in at http://windows.azure.com. You will need to navigate to your service account for your application and click the "deployment" button. You will be asked to provide a name and upload both the package and the service configuration. Once the upload is complete, you will need to click the "Start" button.
- **Management API:** You can deploy with the management API, which is the underlying API that both Visual Studio and the portal use. This approach works well when you want to script or automate your deployments. You will need to have registered a management certificate. You can call the REST service management endpoint directly or use one of the published PowerShell cmdlet packages. You can download them at http://code.msdn.microsoft.com/azurecmdlets.

TWO STYLES OF UPGRADE

There are two key styles of performing an upgrade to your deployed application.

- **VIP Swap:** To use this method, you must have version 1 deployed to the production slot and version 2 (the new version) in the staging slot. Once you have tested the staging environment and are ready to deploy it, you simply click the "Swap VIP" button. This will immediately swap the Virtual IP addresses in the load balancers. Your production slot becomes staging, and staging becomes production. This method leads to a very little amount of possible downtime, but it presents you with a great Plan B option. If the new version isn't working out well, you can click the button again to swap the slots, placing the old production back in the production slot again.
- **Rolling upgrade:** In this process, you click the "Upgrade" button on the slot you want to upgrade, and then you upload a new package. The Fabric Controller will take down one third of your servers, deploy the new code, and bring them back online. Once the servers are back online, the Fabric Controller will proceed to upgrade the second third and then the last third.

  This process results in no downtown. Unfortunately, there is no simple way to roll back to a prior version. To do that, you would have to deploy a second rolling upgrade with the old code. Another drawback is that a rolling upgrade cannot change the shape of the service you are upgrading. For example, you can't add a role.

USE INTELLITRACE TO DEBUG YOUR CLOUD APP

At some point, you might need to debug your application. The easiest way to do this is locally in the emulator. This lets you debug a cloud application like any other application, with support for multiple instances.

Debugging an application that runs in the cloud will rely on using tracing or IntelliTrace. You can use the normal tracing APIs and then pick those up through the diagnostic API. The diagnostic API will pick up your traces and move them to your storage account so you can analyze them offline.

If you have Visual Studio 2010 Ultimate Edition, you can use IntelliTrace. To use this with Windows Azure your project has to use .NET 4 and deploy with Visual Studio 2010 Ultimate Edition. There will be a check box in the deployment wizard to add the IntelliTrace components.

SERVICE MANAGEMENT API

The Service Management API allows you to do almost everything the Windows Azure portal can. You can access the REST API directly or use a set of PowerShell cmdlets.

To access the Service Management API, you will have to register a certificate. Each command to the API must be signed with a registered certificate. You can have up to five certificates registered at one time, and you can revoke a certificate at any time.

The following actions must be done by hand in the Windows Azure portal:

- Access billing data
- Creating a Windows Azure subscription
- Creating a storage or compute service
- Deploying management certificates

AZURE SUBSCRIPTIONS AND ACCOUNTS

You will need a Windows Azure subscription to use Windows Azure. This subscription includes your service contract and payment terms. You will be assigned a subscription ID, which is the equivalent to your account number.

Many people choose to have a different subscription for each deployment region they might want to manage (for example, a subscription for production and a second for testing).

A subscription contains one or more projects. Don't confuse subscription projects with Visual Studio projects; they don't have anything to do with one another. A subscription project is a way to group your Windows Azure resources together. Projects can also help in tracking costs for different projects without having to have separate subscriptions.

How you setup your subscription can affect how you are charged. You can read about all of the pricing for each service on www.azure.com.

GREYBOX—TO KNOW WHEN RESOURCES ARE USED

One way to make sure that you haven't left a demo up and running, which would accidentally burn your allocated hours of compute, is to install GreyBox on your computer. This open-source software will monitor your environment in Windows Azure and remind you if you have left any servers up and running. You can download this free tool at http://greybox.codeplex.com/.

DEVELOPMENT SDKs

The right SDK depends on the technologies you use now:

- Install Windows Azure Tools 1.3 (November 2010) for Microsoft Visual Studio: covers Windows Azure Compute and Storage and the Visual Studio tools
- Install the Windows Azure SDK for PHP and the Windows Azure Companion

When you install the Windows Azure SDK, it will install the compute and storage emulators, which are local simulations of the real cloud.

- **Compute Emulator:** A local and development-only way to host your role instances. The compute emulator is limited to hosting five instances when running your projects.
- **Storage Emulator:** A simulation of the Windows Azure storage services (BLOB, queue, and tables). The simulation is backed by a local instance of SQL Server.

Developers working with other platforms can download an SDK only package (PHP, Ruby, Python, Java, etc.) from the Interoperability Bridges site.

LEARN ABOUT AZURE SECURITY

Pretty early into your 1st "real project" someone will ask about security capabilities. Here are some resources you will love:

- Security for the Windows Azure Platform forum, with experts who respond in less than 72 hours: http://social.msdn.microsoft.com/Forums/en-US/windowsazuresecurity/threads
- Visit the Security Talk Series—all you need to know about security capabilities of the Windows Azure Platform: http://www.microsoft.com/events/series/securitytalk.aspx?tab=videos

ABOUT THE AUTHOR

Brian H. Prince is an Architect Evangelist with Microsoft focused on building and educating the architect community in his district. Prior to joining Microsoft in March 2008, he was a Senior Director, Technology Strategy for a major midwest partner.

Further, he is a co-founder of the non-profit organization CodeMash (www.codemash.org). He speaks at various regional and national technology events including TechEd.

Brian is the co-author of *Azure in Action*, published by Manning Press.

Brian holds a Bachelor of Arts degree in Computer Science and Physics from Capital University, Columbus, Ohio. He is also an avid gamer.

RECOMMENDED BOOK

Azure in Action is a fast-paced tutorial that introduces cloud development and the Azure platform. The book starts with the logical and physical architecture of an Azure app, and quickly moves to the core storage services—BLOB storage, tables, and queues. Then, it explores designing and scaling frontend and backend services that run in the cloud. Through clear, crisp examples, you'll discover all facets of Azure, including the development fabric, web roles, worker roles, BLOBs, table storage, queues, and more.

BUY NOW
www.manning.com/hay/

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-03-5
ISBN-10: 1-936502-03-8
50795
9 781936 502035
$7.95

Version 1.0