Intro to Azure

I've been using Azure to handle LOB and SaaS applications for some time now, and without doubt it's a massively powerful, expansive and complicated system. The aim of this series of posts is to provide an introduction to the Azure offering, break it down into sizable chunks and more or less, cover the areas that I think might be of interest to "small time" developers on how to leverage the power of Azure without having scores of IT professionals with seemingly unending resources to learn and apply everything that Azure offers.

Who this is for

I intend that this series of posts be aimed towards developers that are looking to move from the area of desktop development into cloud development. Essentially this is for people that know database development from a desktop perspective, and want to expand their skills to the Azure platform in order to be able to provide quality, performant cloud based solutions for clients.

Additionally, technologically adept clients may find the series helpful in understanding some of the underlying composition of a given possible solution.

What we'll solve

An increasingly common request from business clients has been that of making data readily available regardless of physical location. Accessing data and features of their software without necessarily being tied to the office and having to work through a terminal service or VPN. Additionally, the desire to have web-based access to mainframe data is a definite requirement, and one that I've found Azure can handle quite well.

Azure in itself is massive, with service offers that go far further than what we as relatively small development shops might require in order to provide these client requirements. Thus, we'll pick out a handful of core Azure services (SQL Database, ASP.NET Web Apps, Web APIs, file storage and a few other supporting components) for a focus area with the intent to see about how Azure can help us devise a solution.

The general idea is to provide a cloud based database, a desktop application that can make use of the cloud database, and an ASP.NET web application to provide mobile access to feature subsets.

What we won't solve

As much as I've come to love Azure and what I can do with it, it's not the cure-all end-all for every scenario (as nothing is). Some companies, for example, may be more suited to a full-fledged web based management application for their business. There's any number of ASP.NET development shops that specialize in this area, and we won't presume to replace that possible solution with Azure (we could, but for the purpose of this series, there's really no reason to).

What's the real advantage for people like us?

Pretty much everything that we can do on Azure, we can do elsewhere also, so… why use it? For me, it boils down to stability of the underlying infrastructure (Microsoft has invested an untold amount into Azure, and it works, and it's not going to fall apart), and from a more practical standpoint: we don't have to manage the implementation details. E.g., we don't have to administer servers. We can have a high level of control over various services without having to be/hire a DBA, or play with IIS, or manage a server OS (unless we want to… there's that option as well). While this can be true with 3rd party hosting, it's very rare to find *good* third party hosting solutions that offer everything we need wrapped up in one provider where we continue to have enough control over things to make them work for us. No getting around it, cloud applications do require significant detail to performance considerations, and one area that Azure excels at that is difficult to find elsewhere is the scalability of the resources. If we're having bottlenecks on our database, we can increase the amount of resources provisioned for it without having to configure anything. Same for Web Apps, or API services, or file storage or cache or anything else. It's very easy to manage high level resources without having to touch a SQL Server configuration, or IIS, or a VM OS.

Performance

I'm putting this way up front (along with security), because it's the single, number one most expressed concern about moving to the cloud. Everyone worries about it... it's going to be slow! Users are going to hate it! I don't believe it can be done… build me a prototype so I can see how the performance will be before I commit to developing a solution (for what it's worth, myself and many other developers have found that performance is easier to manage on an Azure SQL Database than it is on a run of the mill 3rd party hosted or remote VM SQL Server instance).

This section isn't here to describe those in detail at the moment (indeed, much like desktop work, performance is very much a baked in set of practices across a wide spectrum of contexts rather than a strict set of rules to follow, and so it continues to be in the cloud). Instead, what this section is here for is to say *Yes! I hear you. We will address this, and it will work. I'll prove it! ☺ Please be patient, take my word on it for the moment… we'll get there.*

Security

Security gets a special placement here as well. It doesn't get nearly as much concern as performance does, but still it is a top concern for many. Much as described for performance, security is not so much a strict set of actions, but rather a conglomeration of various practices that get baked into our solutions in a variety of contexts. Cloud security is really no different, and for the most part you'll carry on as usual, with perhaps some other special considerations which I'll describe in their appropriate places (for example, some areas have seen "optional" improvement, such as using the Azure Web SDK in Visual Studio and having sensitive information automatically removed from the web.config file and placed elsewhere… I say "optional" because the Azure SDK is not a requirement, but more on this later).

Prerequisites

Before digging in, let's cover a few assumptions I'll make about some technical abilities of the reader. First, I'll assume that you're either a programmer or have a sufficient idea of basic programming practices. I'll assume that you're somewhat experienced in building business applications and working with databases. You'll have some familiarity with SQL Server, an idea of how a website works, what a network is and have an understanding of performance implications. You'll have a rough idea of how standard user permission systems work and possibly some experience with Active Directory.

Generally speaking, I'll assume that you have a decent high level understanding of the various isolated components that make up any given business solution. What I assume you'll need to know is a) how to find, choose and provide these isolated components within Azure, and b) how to tie them together and make them work as a whole.

Azure Services in a Nutshell

Ok, so let's take a 30,000ft view of Azure and its services, and some brief descriptions of the kind of stuff it can do and a few other trivial metrics that might help put things in scope.

A service listing and descriptions of what they do, in terms that we might actually care about. Additionally, note that sometimes there is overlap of services… service A might cover part of service B's offerings as well. Also consider that the services are meant to be mixed and matched and used together in order to provide an overall solution. A Web App, for example, will make use of a SQL Database, etc. Also note that some services seem to exist as subsets of other services, and as such any given list of services (such as the one below), might not contain *everything* that's available.

First let's get a glance at those that we're likely to wind up using for our purposes:

- **SQL Database:** our SQL Server in the cloud
- **Web Apps:** our ASP.NET web applications, including the ability to support lightweight web service APIs
- **WebJobs:** a lightweight version of the Scheduler service that can run a file either continuously, on demand or on a schedule (applicable files types are exe, cmd, bat, sh, php, py, js)
- **Storage** (optional): store files (update distributions, installation files, application images, client images & documents, etc)
- **CDN** (optional): for speeding up websites if there's a lot of usage that's not in the same region as the datacenter
- **Redis Cache** (optional): caching static tables, images, data etc to in-memory caches that SQL Database and ASP.Net can utilize for performance.

And here's a not-quite-but-almost-full list of the current service offerings:

(an updated list of these can be found here: http://azure.microsoftcom/en-us/services, and the documentation home page – as of today – provides documentation links by service based on category: http://azure.microsoft.com/en-us/documentation/ (categories: Compute, Web & Mobile, Data & Storage, Analytics, Networking, Media & CDN, Hybrid Integration, Identity & Access Management, Developer Services and Management))

- **Azure Active Directory (AAD)**: Provides user management within Azure. Technically required for *any* azure work, but for the most part it sets up the minimum requirements for us upon account creation and we can often not touch it again. The required portion of this to use Azure services is simply that the master account is entered into a default directory. It's worth noting that a) AAD can integrate with on-premise AD, b) we can use AAD to provide Single Sign-On features to applications (SSO, e.g., use one account to sign into windows, web apps, desktop apps, 3rd party resources, etc).

- **API Management**: This isn't something we really need to concern ourselves with. While it does make a lot of sense for us to use some basic web service (API) features, we're going to do so in the context of an ASP.NET Web App rather than through this service. This service is more geared toward large, "professional" API services such that eBay or UPS or Amazon may expose to allow third party vendors, users, developers and such to pull and place information within their application. The pricing for this is pretty high and we'd need a substantial amount of web service support requirements to even consider it.
- **Application Insights**: This is basically just a convenient means of monitoring various web app things. Internal analytics, more geared toward developers than to marketers (think Google Analytics but for devs instead of marketers and advertisers). It's nice, it's integrated into the Azure portal and Visual Studio, and does, like Google Analytics, require that we toss a bit of JavaScript into our web pages, so there may be some slight performance implications (though likely not too much – I haven't yet done any serious benchmarking on this). Take it or leave it (it can be easily added later if you opt to leave it at first).
- **App Service/App Service Plan**: Previously called Web Hosting, this is the service plan that allows you to put apps on: this can be web Apps, Mobile Apps and a few others. We'll use this to host our ASP.NET Web App.
- **Automation**: This is above and beyond us. It's geared towards larger shops with the intent to automate builds, deployments, provisioning, etc. Not anything we're going to worry about.
- **Backup**: This is a full-on backup option for entire operating systems (either on premise Windows servers, or VMs). We'll disregard it for our purposes.
- **Batch**: Another large scale service we don't need. Essentially for parallel batch runs of significant resource requirements. We'll ignore for our purposes.
- **BizTalk Services**: At the time of writing, I haven't looked too deeply into this yet, though I'm curious to get more into it as time allows. Basically this allows us to take two external applications and integrate communication between them. For example, if External App A outputs data in an XML format, and External App B imports data in a JSON format, the BizTalk service would allow us to create the required data adapters for that conversion and integrate the two accordingly.
- **CDN (Content Delivery Network)**: CDN is a common service not exclusive to Azure. It's a cache system for websites, basically. It takes your server's static files (namely images, JavaScript and CSS files, etc.) and distributes them to multiple servers (called Edge Servers) worldwide, thus making it much faster for people who aren't necessarily in the same region as the physical server to be served static content without having to go to the other side of the world to get it.
- **Cloud Services**: this one's a bit beyond us as well, for the most part. It more or less allows us to run services on the cloud to help support our solutions. Think of it perhaps like a cloud-based Windows Service engine that we can deploy service application to. We have a few other lighter-weight options for this type of work, so for all but the most demanding of applications, this can be generally disregarded.
- **Data Factory**: Somewhat similar to SSIS for the cloud (SQL Server Integration Services), allows you to pull together data from varying sources: SQL Database (Azure), Blob storage on Azure, on-premise databases, etc. Might be relevant, but we won't go into detail for now. Here's a decent post explaining: http://www.jamesserra.com/archive/2014/11/what-is-azure-data-factory/

- **DocumentDB:** NoSQL databases (essentially used for large amounts of schema-less data: key value pairs with some querying ability to tap into the values side of things). While this hasn't been traditionally a part of LOB applications in the past, it's a relatively new paradigm that could replace a traditional RDMS. Relational vs. NoSQL database comparisons are beyond the scope of this article. Suffice to say that the intended audience of this article will have no need of it.
- **ExpressRoute:** An interesting service, though one which we won't worry about. From MS's landing page: *Azure ExpressRoute enables you to create private connections between Azure datacenters and infrastructure that's on your premises or in a colocation environment. ExpressRoute connections do not go over the public Internet, and offer more reliability, faster speeds, lower latencies and higher security than typical connections over the Internet.*
- **HDInsight:** Hadoop-based service that allows for management of Big Data. Out of scope for us.
- **Key Vault:** High level certification and key management. We won't worry about it for now.
- **Load Balancer:** As the name indicates, this is a load balancer. It is applied to virtual networks and allows for load balancing for network traffic. Typically not applicable to us, as our intended solution range won't generally include having a virtual network.
- **Machine Learning:** I'll just skip this one altogether…
- **Managed Cache:** MS recommends using the newer Redis Cache instead (later in this list)
- **Media Services:** handles management, encoding/decoding, format conversion of various media files. Can help prevent people from ripping videos from your site, etc. Probably not applicable to most of us.
- **Mobile Engagement:** Mobile app usage analytics on steroids I guess is how you'd describe this one. We'll leave it alone for our purposes here.
- **Mobile Services:** Provides a cloud based backend of applications that will go to Windows Store, Windows Phone, iOS, Android, etc. Seems to encompass much of the requirements (databases, etc) that come with this type of solution in isolation, though I think for our purposes we can put a focus on using AS.NET in general rather than trying to build mobile applications with this service.
- **Multi-Factor Authentication:** Adds another layer of security by registering various devices as authenticated devices for login purposes. For some projects this might apply, but for the most part we'll consider it an edge case and leave it out.
- **Notification Hub:** provides the ability to handle push notifications. These can go to subscriber lists containing millions of people to targeted per-user messages. Could be useful depending on various integration/load requirements, but I tend to feel that often times we can use our own solutions to accomplish this on a scale that's more suitable for what we tend to find need for in our contexts.
- **Operational Insight:** Analytics for complex cloud infrastructures. We'll ignore it.
- **Redis Cache:** In-memory cache service that's available to all Azure services. Could be useful for databases, web apps, etc. We'll touch base more on this in a further article.
- **RemoteApp:** Run applications that reside on a VM without having to log into the VM to do so. Could be applicable in some edge cases, but for the most part we'll leave this out (similar to previous Citrix offerings).
- **Scheduler:** As you would guess, a scheduler service. Could be considered similar to the Windows Task Scheduler, with a few twists. Can invoke actions that call HTTP/S endpoints or post messages to a Storage Queue (more on that later), or call other services either inside or

outside of Azure. For our purposes, we have another lighter-weight service that we can use instead (WebJobs).

- **Azure Search:** Could be relevant to us. It's an integrated search services for web apps that aims to give all of the niceties of awesome searching (relevance, suggestions, linguistics, spelling mistakes, etc). Based on Bing and Office search. More on this later.
- **Service Bus:** Messaging queues and notifications that span the majority of Azure's service offerings to help integrate various applications. Can also integrate with on-premise and other applications. For example, you toss a batch email process into the service bus queue for async processing to offload if from a web app.
- **Site Recovery:** This seems to be another VM cloud recovery service. At present, I'm unclear about the differences between this and the Backup service, as both seem to accomplish the same thing.
- **SQL Database:** previously called SQL Azure, it's a SQL Server instance (or instances) in the cloud. Most of the underlying DBA-style management (that of which pertains to physical resources namely: files, hardware, etc) has been stripped away. Retains nearly all of the practical abilities of SQL Server.
- **Storage:** non-relational storage. Files, logs, whatever else that wouldn't go into a database really. Comes in a couple possible flavors: Blob, Table, Queue and Drive. We'll cover in a bit more detail later.
- **StorSimple:** Enterprise level for "massive data growth." I'll leave this alone for now.
- **Stream Analytics:** Real time analytics from a variety of sources: apps, devices, sensors, cars… an Internet of Things analytics solution, it appears to be… we'll leave it alone.
- **Traffic Manager:** Probably not for us in most cases. Allows us to route traffic between multiple datacenters, regions, geo areas, if we should happen to be replicating to multiple geos (e.g, hosting a resource in both US and Europe)
- **Virtual Machines:** Virtual Machines… what more need I say? Ok, well we can provision a virtual machine and set it up with IIS or SQL or whatever. Use it like we would any other server. Turn it into a terminal, set it up on a VPN… whatever. We'll avoid this, because (to me, at least) one of the big advantages of Azure is that we can get what we need without having to screw around with a VM and all the maintenance that goes along with it, both on the OS and on the services levels.
- **Virtual Network:** again, not much to be said. We'll avoid it as one of the primary intents is to get away from these details. But, if we need to, we can get full control over IPs, DNS servers, security rules, traffic flows, etc.
- **Visual Studio Online:** One would think that this is Visual Studio, online. Not quite though… it integrates with VS as we currently know it, and provides cloud collaboration and a few other specialized features. It is *not* an IDE. Version control, agile management, build/deploy tools, performance testing of Azure applications, etc.
- **VPN Gateway:** More or less allows creating a VPN to connect an Azure network to an on-premise network.

How often do they change?

I'm can't really say how often, but the service offerings do tend to change, yes. There's new ones being added on a regular basis, and revisions to existing ones. Keep an eye on the services you decide to use, but generally speaking we'll be using ones that have been around longer and are more stable and less likely to have changes.

Another pertinent point here is that one should be wary of jumping in head first to use whatever catches the eye. Azure has been in a constant state of flux, and oftentimes services are made available through a public preview that may not quite be ready for production use yet. For our purposes, this usually isn't too bad though: all Azure services are built because there's a business need for it somewhere, and a database application is a core business need, thus the service and its related services have been around for a long time (in terms of Azure at least) and are reasonably matured. Specialized services (dynamic data masking, etc) may be a bit finicky here and there, but for the most part the services that we'll be considering tend to be on the safe side.

For keeping up with it, it tends to not be as difficult as one might think once they're "in the mix" of things. My foremost advice here would be to head over to the Azure blog and subscribe via RSS (listed first in the links below). This will get the greater majority of information published to it, covering all service areas. Thus I do recommend at least a perfunctory glance at the full list of services noted above, just so when you see certain listings/posts, you'll have an idea of whether they're important to you or not:

http://azure.microsoft.com/blog/

Next up is the official Updates page. You can sign up for RSS on this as well (much recommended):

http://azure.microsoft.com/en-us/updates/

Scott Gutherie's Blog (ScottGu). Very good stuff:

https://weblogs.asp.net/scottgu

Scott Hanselman has some good stuff at his blog as well (must be something about people named Scott):

http://www.hanselman.com/blog/archives.aspx#Azure

Azure Friday usually just has a video weekly that highlights some feature or another. Sometimes out of our scope but sometimes it's in… worth checking into once in a while anyway:

http://azure.microsoft.com/en-us/documentation/videos/azure-friday/

A Few Last Notes

It’s also worth noting that websites we can create on Azure aren’t limited to ASP.NET and SQL Server as a backend. You can host WordPress, Joomla!, Drupal, node.js, custom PHP sites, MySQL databases, etc. (this is without having to provision a Linux server, which is another option as well).

June 22nd, 2015
Jack D. Leach
Dymeng SERVICES
http://www.dymeng.com/azure