Scalability and Performance of Cloud Hosted Web Apps –DEMOS

The source code for the demos is published on Git Hub repository <https://github.com/morcinim/Azure-PA-DEVELOPER>

# Autoscaling Compute Demo

Requirement: Azure Web App + SQL Database + CloudShop web application +VS 2015+SQL Management Studio

Description: The demo code is in 1-Autoscaling Compute

1. Create SQL Azure Database on V12 server (if not possible create V2 server then upgrade). Create new database using the option to provision demo database (reduced version of Adventure Works
2. Connect to the SQL Database and use 1-Autoscaling Compute\ConfigureFullTextSeach.sql to configure Full text search for the Products table.
3. Open the CloudShop.sln in VS2015 and adjust the connection strings: DefaultConnection and AdventureWorksEntities web.config
4. Deploy the Application to the Azure Web App and demonstrate scaling configuration

<https://azure.microsoft.com/en-us/documentation/articles/web-sites-scale/>

TODO:

Investigate powershell RM script for configuring auto-scaling Web App/Cloud Service.

Investigate if scale up event could be produced by test loading from VSO.

# Scaling Database

Requirement: SQL Azure databases + sample console application + Azure Automation service

Vertical scaling:

1. Run powershell script 2-Scaling database\Vertical Scaling\ScaleSQLDatabase.ps1 to show how to scale down SQL Azure database. To make demo more interesting if you are familiar with Azure Automation is to setup this script as a runbook on a schedule <https://azure.microsoft.com/en-us/documentation/articles/automation-configuring/>

and scale down one of the demo databases (AdventureWorks) to Basic tier at night.

In the actual demo you review the logs and comment on the powershell commands.

Horizontal scaling preparation

1. Demonstrate database sharding for scenario of one tenant per database, demonstrate querying across databases. The code can be found in 2-Scaling database\Horizontal Scaling and is based on the sample described <https://code.msdn.microsoft.com/vstudio/Elastic-Scale-with-Azure-bae904ba>
2. Open the solution located at 2-Scaling database\Horizontal Scaling\ElastingDBTools\EFIntegration\C#
3. Open Program.cs and provide the server name and three database names. These have to be pre-created, the code does not create them. Execute the code to create the schemas and sample entries.

Horizontal scaling demo:

1. Explain that the scenario is similar to that of SaaS application that uses a separate database per tenant. The application stores blob list in the database that is determined by the tenant ID.
2. Explain Nuget packages used by solution –In Visual Studio Tools-Nuget Package Manager-Manage Nuget Packages for Solutin- comment on Transient Fault Handling for SQL Azure database and Elastic Scale client API
3. Open SqlDatabaseUtils.cs comment on the retry policy for transient errors that can occur with SQL Azure db, go back to Program.cs point out how EF code to write blogs is passed as delegate to ExectuteAction of the retry policy.
4. Comment on the First lines of code where connection is established to Shard management database and shards are registered, follow by F12 to code in Sharding.cs
5. Point out the constructor where shard map manager is created with the List shard map where the relationship between the tenatIDs and databases will be maintained
6. The actual registration of the shard occurs in RegisterNewShardFunction
7. Put break point on //MultiShard query, run the code.
8. Explain Blog entries are stored in respective SQL Azure databases
9. When breakpoint is hit comment on the ability to query across shards (databases) using the MultiShard/Connection, Command and DataReader objects
10. Optionally comment PG is adding TSQL commands to do the same (open QUERY\Elastic Query.sql) based on the [PolyBase](https://msdn.microsoft.com/en-us/library/mt143171.aspx) approach in SQL 2016 where you define external datasource painting to the Shared management database and the shard map and then define external tables that map to actual tables in the shards. Then you can launch adhoc TSQL queries to query across the shards as we saw with the Elastic client API.

If you want to demo this query the ShardMapManagement database has to be set to Premium tier.

# Session Affinity

Requirement: Azure Web App + SQL Database + CloudShop web application (same as 1)

<http://azure.microsoft.com/blog/2013/11/18/disabling-arrs-instance-affinity-in-windows-azure-web-sites/>

NOTE: Instead of the step by step demo, to save time you can setup the demo with modified web.config and deployed to separate website (cloudshop-affinity) and just demonstrate diffecence in behavior with the original one.

1. Before the demo make sure you are on either Basic or Standard plan and that you have 2 instances running. Also in classic portal make sure you enabled KUDU editor (Edit in Visual Studio Online set to ON in the configuration section).
2. Examine session cookie data in IE/Edge F12 dev. Tools: Launch Edge browser navigate to the web application you deployed to Azure Web App Service. Press F12 to Launch Developer Tools
3. Click Network Tab, Cookies to the Left and click the URL of the web application (reload if required)- Point out the ARRAffinity Cookie
4. In browser windows select product from the product list and click “Add item to cart”. Back in Dev tools window click the URL entry with Name “Add” (you may need to click other entry and then back that one) – see that the Cookies includes ARRAffinity (in addition to cookies representing the ASP.Net session that stores the product you added to the cart).
5. Now we will disable the affinity
6. Go to Dashboard, click Edit in VSO Online, click web.config to edit it
7. CRTL+F – search for system.webServer element then paste following below the system.webServer element

<httpProtocol>

<customHeaders>

<add name="ARR-Disable-Session-Affinity" value="true"/>

</customHeaders>

</httpProtocol>

1. Switch to Azure management portal, click Restart button
2. Close previous instances of Edge browser, start new instance and point it again at website. Start Developer tools
3. Click URL entry for the application – click Headers and point out ARR-Disable-Affinity set to true in the response headers section. Click Cookies – point out there is no ARR Affinity cookie
4. Add now two or three products the go to Checkout page- Note that not all products are present in the cart. Press CRTL+F5 to refresh the checkout page – you should see that the machine name RD has changed. If you look into the list of products in the cart you will see that the missing products were.

You can see that disabling the affinity causes the user session to be connected to any of the two machines. Because the session object is stored by default in memory of the web server you now have two shopping carts. In order for the application to work correctly you need to ensure that the session is stored out of process, for example in the Redis Cache that we talk about later in the presenation.

# Redis Cache

Requirement: Azure Web App + SQL Database + CloudShop web application (same as 3)

Demo Preparation:

Integrate CloudShop with Redis (Session state) (optionally) reference data (product list)

1. Create Redis Cache following <https://azure.microsoft.com/en-us/documentation/articles/cache-dotnet-how-to-use-azure-redis-cache/> and enable session state provider for the CloudShop application. C1 Basic 1GB cache should be sufficient. When creating enable Diagnostics as it has advantage to show get/set access to cache in the management portal
2. Modify the web.config of the CloudShop application to in connection string element RedisConnection and the <sessionState> element to include your Redis cache connection parameters
3. You can test locally, then deploy the code the website making sure it is in the same location as the Redis Cache you created.

Actual demo:

1. Run the application from the website (it is recommended to have Standard tier with Always On enabled not to waste time on website warm up).
2. Point out to the timer in the bottom of the page that tracks time to access products repository
3. Type “LL” into the seach box click Search – the results should come back in the order of seconds
4. Click search again now the results come in miliseconds.
5. Explain we use cash aside pattern for accessing products repository both when searching it or returning complete list.
6. Show implementation of 4-Redis Cache\CloudShop\Services\ProductsRepository.cs .The cach Set and Get methods are implemented in the extension static class. Global.asax.cs contains the code to connect to Redis cache on application startup.
7. If you showed Session Afinity Demo and have two instances you can also demo that now the shopping cart works correctly as it is stored out of process in Redis cache thanks to the provider we configured in web.config

# Azure CDN

Requirement: same as 1) but deployed to two datacenters – for SQL Azure could use same database

Plan:

1. Integrate CloudShop web application with Azure CDN (graphics + script bundles+ Controllers).

# Traffic Manager

Requirement: Azure Web App + SQL Database + CloudShop web application (same as 1) + but web app deployed to two datacenter locations (single database OK) + Traffic Manager + VM in secondary datacenter with RDP connection enabled to use as client to demonstrate performance based routing.

https://azure.microsoft.com/en-us/documentation/articles/traffic-manager-configure-performance-load-balancing/

Demo preparation steps:

1. Publish the 1-Autoscaling Compute\CloudShop web application to a secondary datacenter. If you did previous demos from West Europe you deploy it to North Europe.
2. Set the app setting on both websites to different values to distinguish which one is served by Traffic manager by running provided powershell script 7- .Make sure both websites are accessible and display datacenter location in the title as expected.
3. Configure Traffic Manager performance profile as per <https://azure.microsoft.com/en-us/documentation/articles/traffic-manager-configure-performance-load-balancing/>
4. Verify you can access from traffic manager DNS name (cloudshop.trafficmanager.net) your demo machine. Note to which datacenter location you are routed, this is your primary location.
5. Create an Azure VM in the secondary location. We will only use the Browser to access the traffic manager URL from it. Configure RDP connection and make sure you can access the website through traffic manager DNS name (cloudshop.trafficmanager.net)

Actual demonstration:

1. Navigate from your demo machine to the website using traffic manager DNS name (cloudshop.trafficmanager.net). Call out that we get routed to nearest location and call out the title of the website reflecting that.
2. Run a command window with nslookup cloudshop.trafficmanager.net to show how the DNS name gets resolved to actual website DNS name
3. RDP into the Azure VM in a secondary location. Open IE and navigate through traffic manager DNS name (cloudshop.trafficmanager.net). Show different title of the website as we connect from machine closer to secondary datacenter we got routed there
4. Time permitting – navigate to <http://manage.windowsazure.com> and review the Traffic Manager profile configuration

# Async Programming

Requirement: Azure Web App + SQL Database + CloudShop web application (same as 1)

Description: The demo code is in 7-Async Programming

<http://www.asp.net/mvc/overview/performance/using-asynchronous-methods-in-aspnet-mvc-4>

The application has been modified to access the repository of products using asynchronous code. The implementation of the controller now uses asynchronous methods and the products repository uses the asynchronous version of LINQ queries to access the database.

1. Open solution in Visual Studio 2015
2. Examine Controllers\HomeController.cs. Note that the controller actions Index and Search are now marked as asynch methods and return a Task<ActionResult> object instead of Action result.
3. Examine Services\ProductsRepository.cs and point out GetProductsAsync() and SearchAsync() menthods which use ToListAsynch() extension method to execute the query to the database. The first method just lists all products in the database whereas the second uses a string to execute a full text search to find products with matching names.
4. The repository functions are called from the Search() function in HomeController.cs that is called with no argument when you first load the page or with a string that you specify in the search textbox of the page.
5. The function will make async call to get either the entire list of products or the ones that match search string entered. It then removes from the list the products that user has previously added to their sopping card. Notice the function is marked as async and uses await keyword to await on the result of the calls to product repository