Scalability and Performance of Cloud Hosted Web Apps -DEMOS

# 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/>

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

1. Show though powershell script how to scale up/down SQL Azure database (can include Automation services to execute scaling runs couple of days before delivery
2. Demonstrate database sharding for scenario of one tenant per database, demonstrate querying across databases

# 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)

Plan:

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

# 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 routing.

1. Configure Performance profile for traffic manager.

# 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 asynch 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 asynch and uses await keyword to await on the result of the calls to product repository