Create your first search solution using Azure Search

By Heidi Steen
Last updated: 11/04/2015

In this article:

Prerequisites

Create an Azure Search index

Explore CatalogIndexer

Build an MVC4 Application using Azure Search

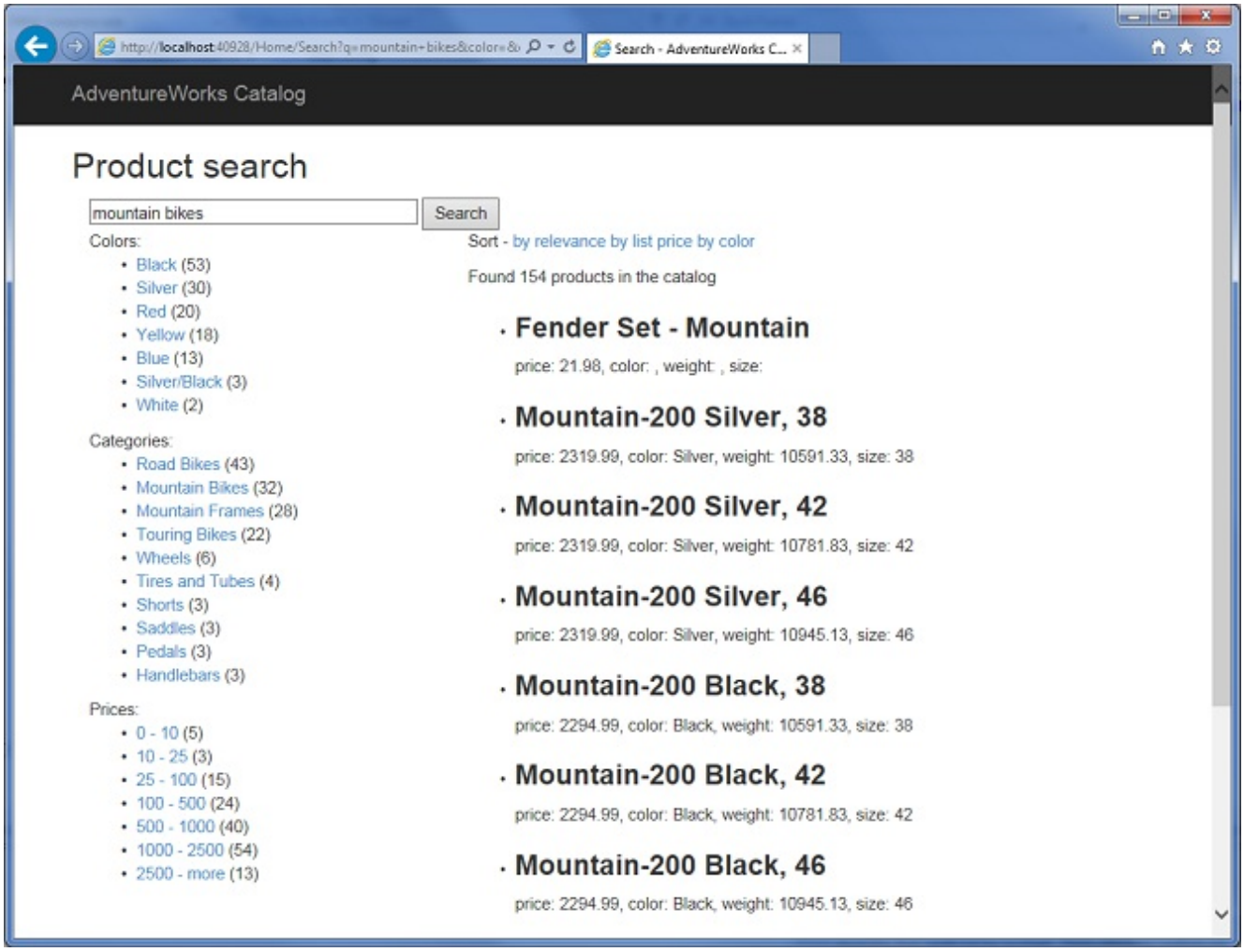
Explore AdventureWorksWeb

How to resolve "Could not load file or assembly 'System.Web.Mvc"

Next steps

11 Comments

This sample demonstrates a search application for the Adventure Works bicycle company. After completing this tutorial, you'll have an online product catalog with a rich search experience that includes faceted navigation, multiple sort options for your search results, and type-ahead query suggestions.



The demo gets you started with Azure Search by walking you through these exercises:

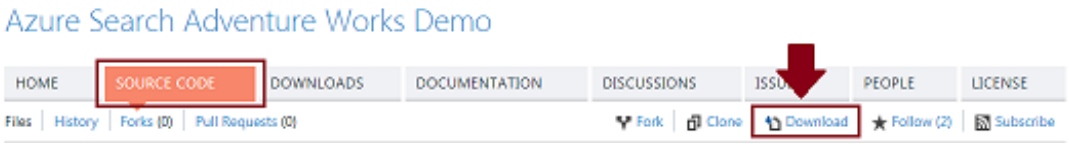
- Create an Azure Search index.
- Load documents (data) into the Azure Search index from a SQL Server database.
- Build an ASP.NET MVC4 Application that utilizes Azure Search to:
  - Search and display results from an Azure Search index
  - Show type-ahead suggestions in a Search box while entering a search term
  - Filter search results using faceting

Prerequisites

- An Azure subscription. If you aren't ready to sign up for a trial subscription, you can skip this tutorial and opt for Try Azure App Service instead. This alternative option gives you Azure Search with an ASP.NET Web app for free - one hour per session - no subscription required.
- Visual Studio 2012 or higher with ASP.NET MVC 4 and SQL Server installed. You can download the free Express editions if you don't already have the software installed: Visual Studio 2013 Express and Microsoft SQL Server 2014 Express.
- An Azure Search service. You'll need the Search service name, plus the admin key. See

Create an Azure Search service in the portal for details.

- Adventure Works Azure Search Demo project on CodePlex. On the Source tab, click **Download** to get a zip file of the solution.



This solution contains two projects:

- **CatalogIndex** creates an Azure Search index and loads data from a SQL Server database included with the project.
- **AdventureWorksWeb** is an MVC4-based application that queries the Azure Search index. This tutorial assumes that you have a working knowledge of ASP.NET MVC.

Create an Azure Search index

In this step, you will use **CatalogIndex** to create a new Azure Search index called "catalog" based on data in an AdventureWorks SQL Server database.

1. In Visual Studio, open up the Azure Search Demo solution named **AdventureWorksCatalog.sln**.
2. Right-click **CatalogIndexer** in Solution Explorer and select **Set as startup project** so that this application runs, rather than the **AdventureWorksWeb** project, when you press **F5**.
3. Open **App.config** in this project and update the values for "SearchServiceName" and "SearchServiceApiKey" to those of your Azure Search service. For Search service name, if your service is "mysearch.search.windows.net", you would enter "mysearch".
4. Optionally, App.config includes an entry for "SourceSqlConnectionString" that assumes SQL Server 2014 Express LocalDB (Server=(LocalDB)\v11.0). If you're using a different edition of SQL Server, update the server name accordingly. For example, if you have a local default instance, you can use (local) or localhost.
5. Save **App.config**.
6. Press **F5** to launch the project.

A command prompt should open with the following text: "Creating index..."

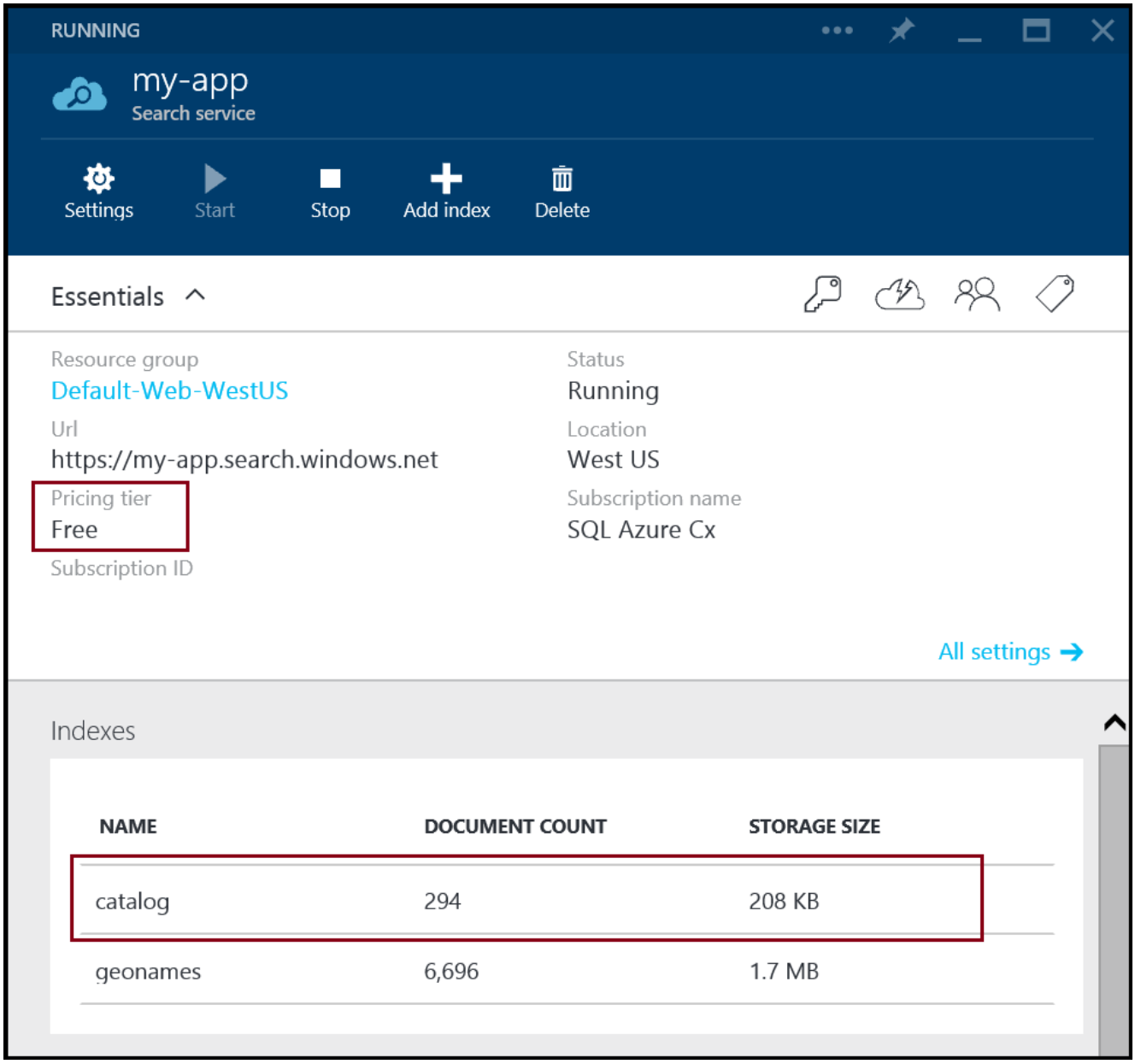
After a few moments, it should complete with the text: "Complete. Press to continue:"

Press **Enter** to close the application. At this point, you have successfully created the Azure Search index.

> NOTE:
>
> If you get errors that include "invalid value for key 'attachdbfilename'" or some other database attachment error, you might be running into UAC conflicts. For the purposes of this demo, work around these errors by doing the following: Copy the solution to a new or existing folder (such as Temp) that provides access to authenticated users. Use **Run as Administrator** to start Visual Studio. Open the solution, build it, and then press **F5** to create the index.

To verify index creation and document upload, go to your Search service dashboard in the Azure management portal. In Usage, the index count should be up by one, and you should have 294 documents, one for each product in the database.

Click the **Indexes** tile to show the index list. The indexes list slides out to show the new index and document count. Note that you can have up to three indexes at the Free pricing tier. If you already had three indexes, you would need to delete one to free up space for any new ones.



Explore CatalogIndexer

Let's take a closer look at the **CatalogIndexer** project to understand how it works.

1. Open **Program.cs** from the Solution Explorer and go to the `Main(string[] args)` function.

   Notice how this function builds up a Uri called `_serviceURI` based on your specific Azure Search service and then creates an HttpClient called `_httpClient` that makes use of your API Key to authenticate to this Search service.

2. `ChangeEnumeratorSql` and `ChangeSet` are used to enumerate the data from the Products table in your SQL Server AdventureWorks database.

3. Based on the data that is collected from this table, `ApplyChanges` is then called to apply this data to your Azure Search index.

4. Move to `ApplyChanges` in the same file. Notice how this function deletes the index if it already exists (`DeleteCatalogIndex`) and then creates a new index called "catalog"

(`CreateCatalogIndex`).

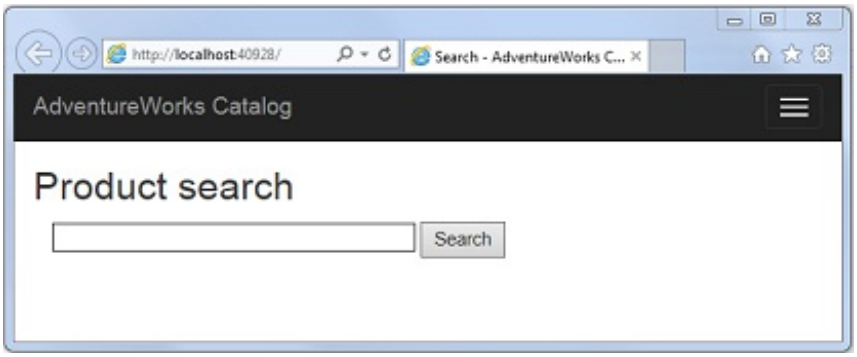
5. Move to the `CreateCatalogIndex` function, and notice how the index is created with a schema that matches the columns from the Products table in SQL Server. Each field has a Type (i.e., `Edm.String` or `Edm.Double`) as well as attributes that define what these fields are used for. Refer to the Azure Search REST API documentation for more details on these attributes.
6. Go back to the `ApplyChanges` function. Notice how this function loops through all of the data in the enumerated changes `ChangeSet`. Rather than applying the changes one by one, they are batched into groups of 1000 and then applied to the Search service. This is much more efficient than applying the documents one by one.

Now that you've seen how to create and populate an index using relational data from SQL Server, let's take a look at how to build a product catalog that uses our search data.

Build an MVC4 Application using Azure Search

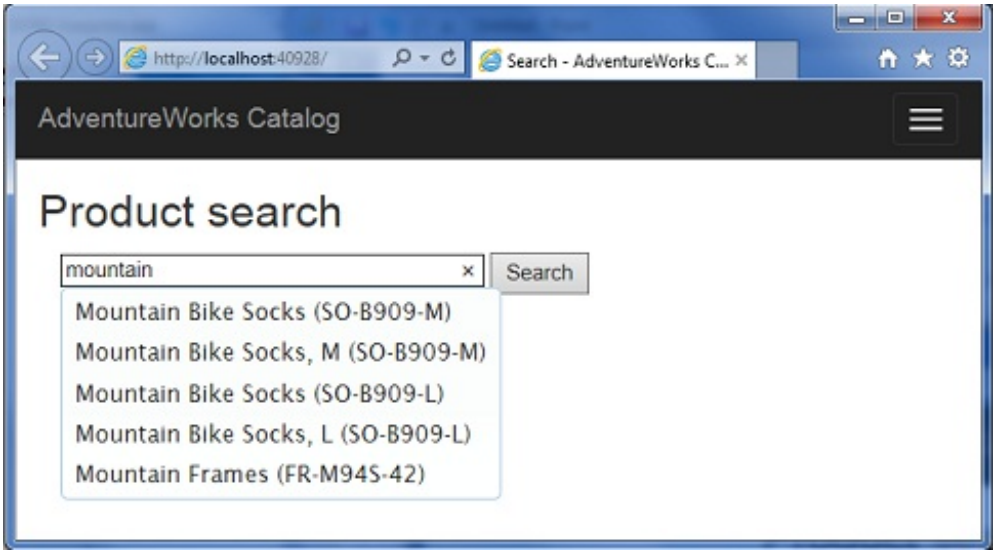
In this step, you'll build and run the search application in a web browser.

1. In Visual Studio, open up the Azure Search Demo Solution named **AdventureWorksCatalog.sln**.
2. Right-click **AdventureWorksWeb** in the Solution Explorer and select **Set as startup project**.
3. Open **Web.config** in this project and update the values for "SearchServiceName" and "SearchServiceApiKey" to those of your Azure Search service. For Search service name, if your service is "mysearch.search.windows.net", you would enter "mysearch".
4. Save **Web.config**.
5. Press **F5** to launch the project. Follow these Troubleshooting steps if you get a build error.

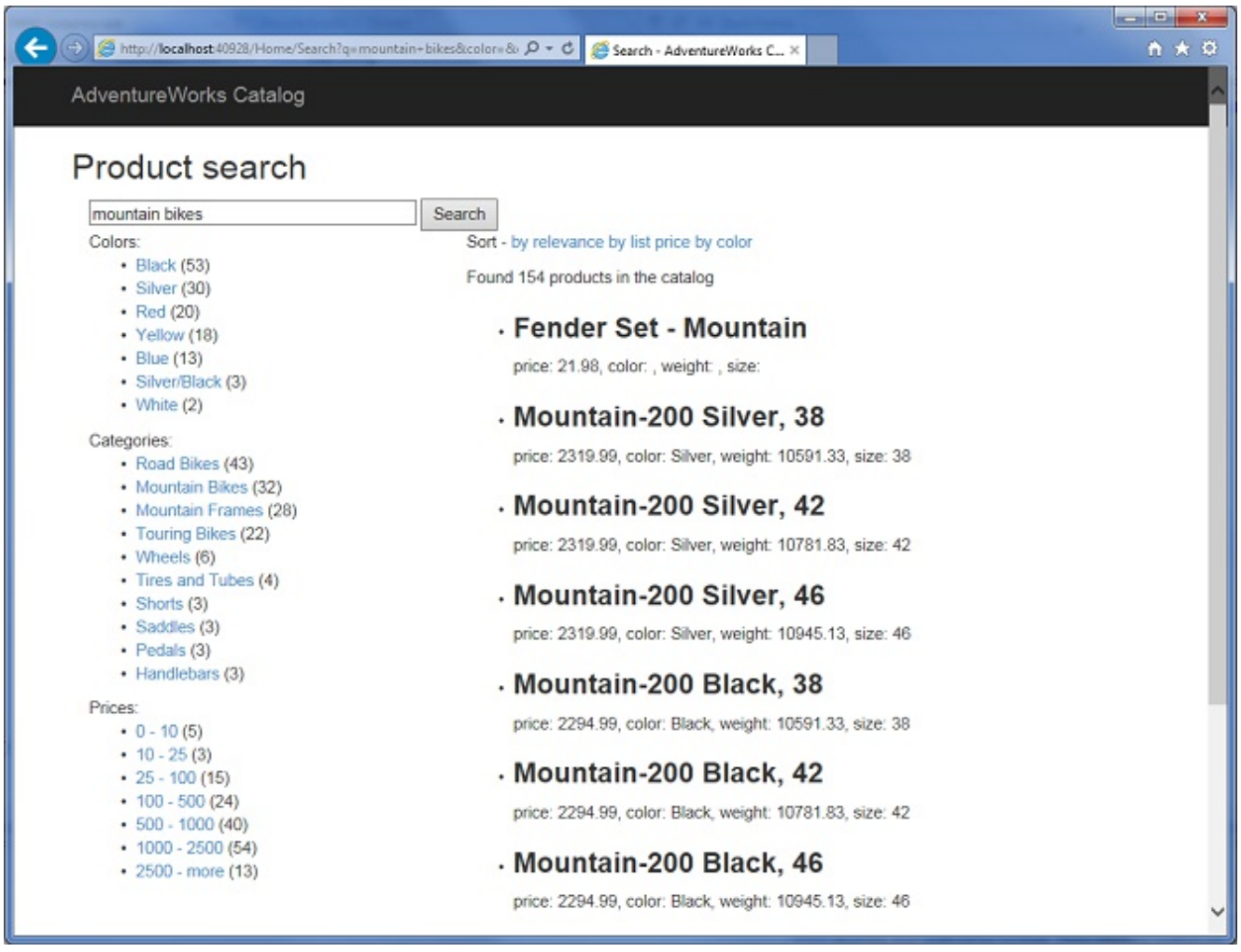


6. Leave the search box empty, and click **Search** to return all 294 products.
7. Notice the list of facets along on the left side. Try clicking one of the facets. Search is re-executed, but this time it adds the facet you chose to filter the results. You may want to add a breakpoint to the first line in the **HomeController.cs** `Search` function to step through (F11) each line.

8. Enter a search term, such as "mountain bikes". As you type, notice the drop-down list of suggested queries.



The screenshot from the start of this demo appears again below, as a reminder of what was covered. In the previous sections, we discussed faceted navigation (on the left), the document count at the top of the page, suggestions, and the sort options for sorting by relevance, list, or price.



Explore AdventureWorksWeb

The project AdventureWorksWeb shows us how ASP.NET MVC 4 can be used to interact with Azure Search from a web application. In this section, we'll review individual parts of the application code to see what they do.

1. In Solution Explorer, expand **AdventureWorksWeb** | **Controller** and open **HomeController.cs**

   This is the MVC controller that manages the interaction from the Index view. At the top of the controller, you will notice a reference to `CatalogSearch _catalogSearch` which creates a HttpClient connection object to your Azure Search service. The code for this `CatalogSearch` is located in **CatalogSeach.cs**

2. Within **HomeController.cs**, there are two main functions:

   Search: When the user clicks on the search button or chooses a facet, this function is called to retrieve the results and send them back to the Index view to be displayed.

   Suggest: As the user is typing in the Search box, this function is called to return a list of suggestions based on the content in your Azure Search index.

Let's drill into these two functions in more detail.

1. In **HomeController.cs** `Search` function, you will notice a call to `_catalogSearch.Search`, which includes the connection object to your Azure Search service. When you call the Search function located in **CatalogSearch.cs**, you can see that it utilizes these parameters to build out an Azure Search query. The results of the query are stored in a JSON object called `result` and then passed back to the `Index` view where the results are parsed and displayed in the web page.
2. Open **Index.cshtml** under Views | Home, you will notice the code that is used to parse these results.
3. Stop the application if it's still running, and open **Index.cshtml** file under Views | Home. At the end of this file, you will see a JavaScript function that uses `JQuery $(function ())`. This function is called when the page loads. It uses the JQuery autocomplete function and links this function as a callback from the search text box, identified as "q". Whenever someone types into the text box, this autosuggest function is called which in turn calls /home/suggest with what is entered. `/home/suggest` is a reference to the function in **HomeController.cs** called `Suggest`.
4. Open **HomeController.cs** and move to the Suggest function. This code is very similar to the Search function that uses the `_catalogSearch` object to call a function in **CatalogSearch.cs** called `Suggest`. Rather than making a search query, the `Suggest` function makes a call to the Suggestions API. This uses the terms entered in the text box and build out a list of potential suggestions. The values are returned to the **Index.cshtml** file that are automatically listed in the Search box as type-ahead options.

You might be asking yourself at this point how Azure Search knows what fields to build

suggestions over. The answer to this is back when you created the Index. In the `CreateCatalogIndex` function within the file Program.cs of the **CatalogIndexer** project, there is an attribute called `Suggestions`. Whenever this attribute is set to `True`, it means that Azure Search can use it as a field for retrieving suggestions

Let's give this a try.

1. Once again, build the application and then press **F5** to run the application.
2. In the search box type in the word "Road". After a second, you should see a list of items display below the text box with Suggestions that a user could potentially choose.

You might want to add a breakpoint to the `Suggest` function within **HomeController.cs** and step through (F11) each call that is made to build out the Suggestion list.

This concludes the demo. You have now walked through all of the main operations that you'll need to know before building out an ASP.NET MVC4 application using Azure Search.

How to resolve "Could not load file or assembly 'System.Web.Mvc"

When building AdventureWorksWeb, if you get "Could not load file or assembly 'System.Web.Mvc, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35' or one of its dependencies", try these steps to resolve the error.

1. Open the Package Manager Console: **Tools** | **NuGet Package Manager** | **Package Manager Console**
2. At the PM> prompt, enter "Update-package -reinstall Microsoft.AspNet.Mvc"
3. When asked to reload the file, choose **Yes to All**.
4. Rebuild the solution and try **F5** again.

Next steps

For additional self-study, consider adding a Details page that opens when a user clicks one of the Search results. To prepare, you could do the following:

- Read up on the Lookup API that allows you to make a query to Azure Search to bring back a specific document (for example you could pass the productID).
- Try adding a new function in the **HomeController.cs** file called Details. Add a corresponding **Details.cshtml** view that receives the results of this Lookup and displays the results.
- Check out this additional code sample and video on geospatial search: Channel 9 - Azure Search and Geospatial Data and CodePlex: Azure Search GeoSearch Sample

You can also review the Azure Search REST API on MSDN.

Need help?

Go to an MSDN forum ▶

StackOverflow discussion ▶

11 Comments Azure Site 1 Login

Recommend Share Sort by Newest

Join the discussion…

Dan Friedman • 3 months ago

I tried this in VS2015 and got errors connecting to the DB file, even in admin mode. This helped: https://stackoverflow.com/ques...

• Reply • Share ›

Jussi Palo • a year ago

Hi,
Does the index automatically drop out the oldest items when I add an item that exceeds the storage size or documents hosted limit, or do I need to check the current index size before each insert?

UPDATE: Looks like I have to take care of that myself: "Status code: 429 indicates that you have exceeded your quota on the number of documents per index. You must either create a new index or upgrade for higher capacity limits."

• Reply • Share ›

Heidi Steen [heidist@MSFT] ➚ Jussi Palo • 10 months ago

You can now delete indexes from the portal, which is much easier and faster than the API approach.

• Reply • Share ›

Liam • a year ago

Hi Brian,

You certainly could use Azure Search as a tool for searching content on your site, however, this is really not a core scenario that Azure Search targets

(http://azure.microsoft.com/blo.... One of the biggest issues you will have is in the indexing of the content. Since Azure Search does not have a crawler for indexing the content in your website, you would need to write the code that takes this content and pushes it to the Azure Search API for indexing.
Liam

Reply • Share ›

brian • a year ago

Can this be used to search sites that do not store content in a database?

Reply • Share ›

Liam • a year ago

Aref, there is a good discussion about this error here: http://stackoverflow.com/quest...
Does this help?

Reply • Share ›

Aref Kashani • a year ago

If anyone is facing issue "Could not load file or assembly 'System.Web.Mvc, Version=4.0.0.0, Culture=neutral, PublicKeyToken=*' or one of its dependencies. The system cannot find the file specified."
To resolve: in VS Package Manager Console run command: "Update-Package -reinstall Microsoft.AspNet.Mvc"

Reply • Share ›

Dustin ➚ Aref Kashani • a year ago

Did you ever resolve this? I'm experiencing the same issue.

Reply • Share ›

Aref Kashani ➚ Dustin • a year ago

As liam pointed out:http://stackoverflow.com/quest...

Or like I said -
To resolve: in VS Package Manager Console run command: "Update-Package -reinstall Microsoft.AspNet.Mvc"

Reply • Share ›

Dustin ➚ Aref Kashani • a year ago

totally somehow missed that you said that. Thanks!

Reply • Share ›

Liam ➚ Aref Kashani • a year ago

I think Aref's suggestion is the best way to resolve this. By the way, you can find the VS Package Manager Console in Visual Studio under Tools -> Nuget Package Manager -> Package Manager Console

Reply • Share ›

Go Social

Microsoft Azure +

Community +

Support +

Account +

Trust Center +

Hello from Seattle.

English (US)

Euro (€)

Contact Us Feedback Trademarks Privacy & Cookies

Microsoft