Azure DocumentDB

Overview

Whether it targets businesses, consumers, or both, an app is only as meaningful as the data that drives it. With consumer and organizational requirements changing constantly, as well as the need to store, index, and optimize data and structures as they change, the need for a more open, flexible, and schema-agnostic data solution has been become essential. Azure DocumentDB addresses these challenges and makes it easy to adjust and adapt data models on the fly, as business logic and application scenarios change.

DocumentDB is a fully managed NoSQL database service built for fast performance, high availability, elastic scaling, and ease of development. As a schema-free NoSQL database, DocumentDB provides rich and familiar SQL query capabilities over JSON data, ensuring that 99% of your reads are served under 10 milliseconds and 99% of your writes are served under 15 milliseconds. These unique benefits make DocumentDB a great fit for Web, mobile, gaming, IoT, and many other applications that need seamless scale and global replication.
In this lab, you will deploy an Azure DocumentDB database to store customer and product order information for the fictitious company *Adventure Works*, and you will connect it to Azure Search to index the data and implement auto-suggest. You will also write a Web app that uses the database and demonstrates how easily applications can consume data from DocumentDB.

Objectives

In this hands-on lab, you will learn how to:

- Create an Azure DocumentDB account
- Create DocumentDB collections and populate them with documents
- Create an Azure Search service and and use it to index DocumentDB data
- Access Azure DocumentDB collections from your apps
- Query the Azure Search service connected to a DocumentDB database

Prerequisites

The following are required to complete this hands-on lab:

- An active Microsoft Azure subscription. If you don't have one, sign up for a free trial.
- Visual Studio 2015 Community edition or higher

Exercises

This hands-on lab includes the following exercises:

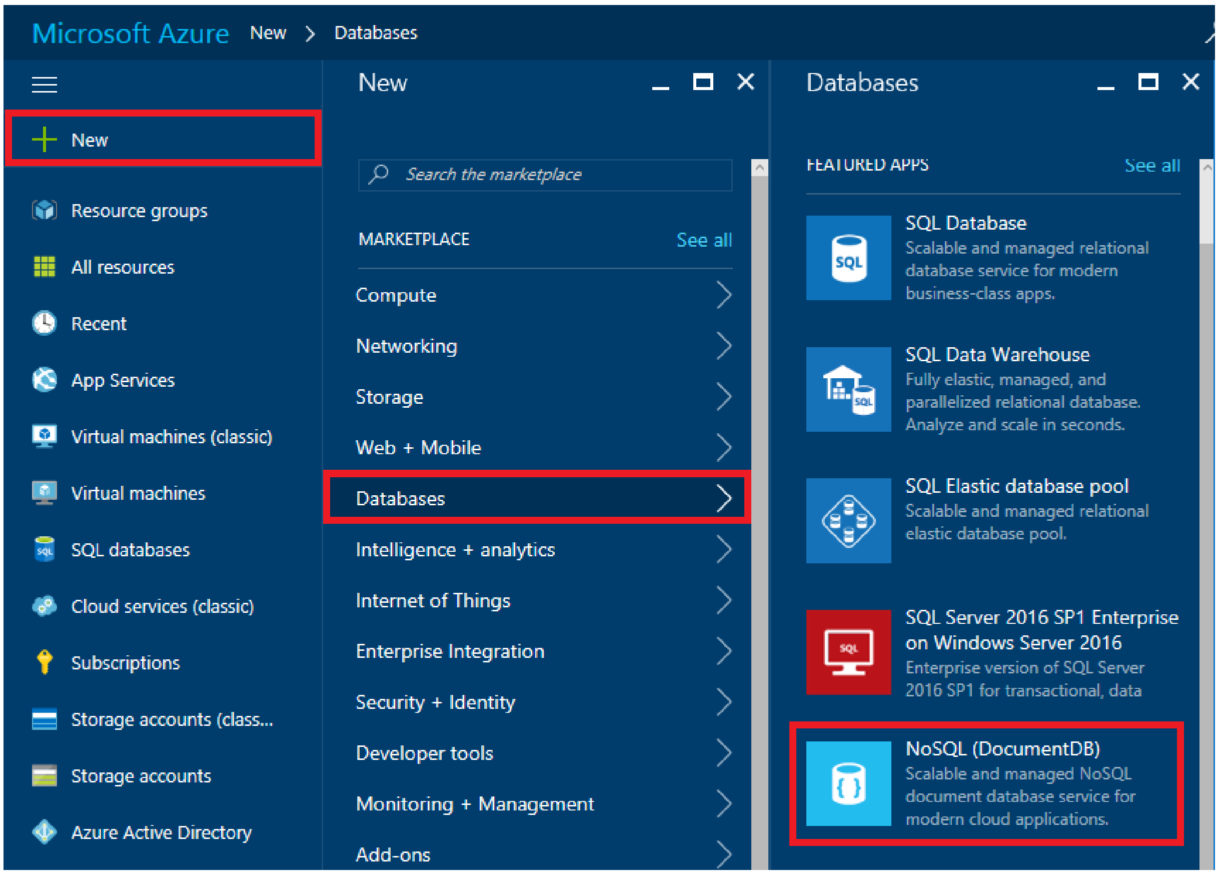
- Exercise 1: Create a DocumentDB account
- Exercise 2: Create a database and collections
- Exercise 3: Populate collections with documents
- Exercise 4: Connect Azure Search
- Exercise 5: Build an Azure Web App
- Exercise 6: Add auto-suggest

Estimated time to complete this lab: **60** minutes.

Exercise 1: Create a DocumentDB account

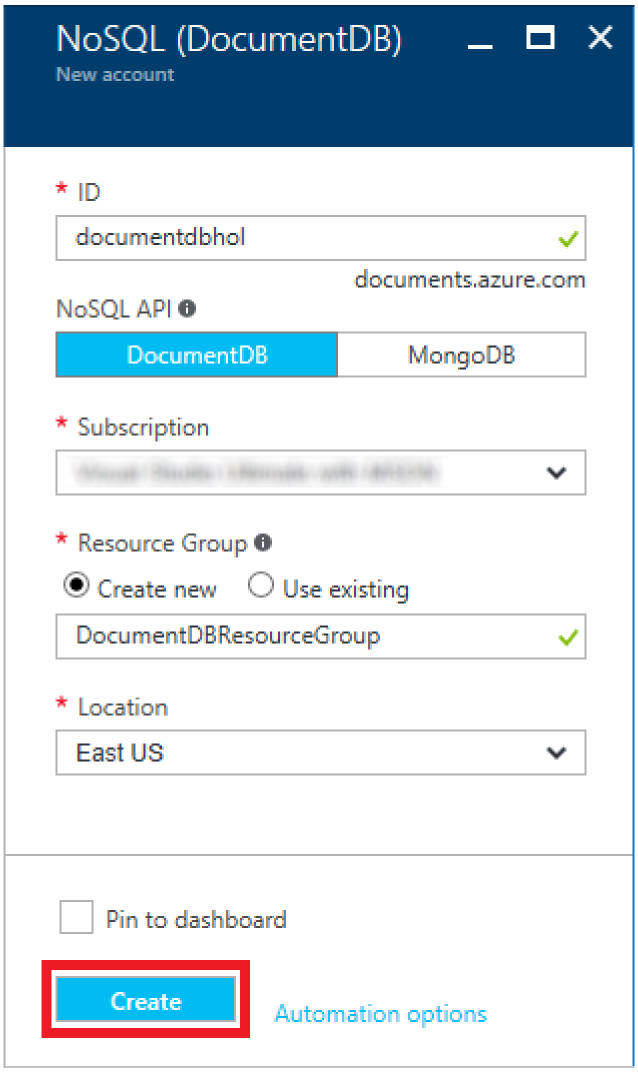
The first step in working with Azure DocumentDB is to create a DocumentDB account to hold databases, collections, and documents. In this exercise, you will create a DocumentDB account using the Azure Portal.

1. Open the Azure Portal in your browser. If you are asked to sign in, do so with your Microsoft Account.
2. Click **+ New**, followed by **Database** and **DocumentDB (NoSQL)**.


Microsoft Azure
New > Databases
New
Resource groups
All resources
Recent
App Services
Virtual machines (classic)
Virtual machines
SQL databases
Cloud services (classic)
Subscriptions
Storage accounts (class...
Storage accounts
Azure Active Directory
New
Search the marketplace
MARKETPLACE
See all
Compute
Networking
Storage
Web + Mobile
Databases
Intelligence + analytics
Internet of Things
Enterprise Integration
Security + Identity
Developer tools
Monitoring + Management
Add-ons
Databases
FEATURED APPS
See all
SQL Database
Scalable and managed relational database service for modern business-class apps.
SQL Data Warehouse
Fully elastic, managed, and parallelized relational database. Analyze and scale in seconds.
SQL Elastic database pool
Scalable and managed relational elastic database pool.
SQL Server 2016 SP1 Enterprise on Windows Server 2016
Enterprise version of SQL Server 2016 SP1 for transactional, data
NoSQL (DocumentDB)
Scalable and managed NoSQL document database service for modern cloud applications.

Creating a DocumentDB account

3. In the "NoSQL (DocumentDB)" blade, give the account a unique name such as "documentdbhol" and make sure a green check mark appears next to it. (You can only use numbers and lowercase letters since the name becomes part of a DNS name.) Make sure **DocumentDB** is selected for **NoSQL API**. Select **Create new** under **Resource group** and name the resource group "DocumentDBResourceGroup." Select the **Location** nearest you, and then click the **Create** button.



NoSQL (DocumentDB)
New account

* ID
documentdbhol
documents.azure.com

NoSQL API
[✓ DocumentDB] MongoDB

* Subscription
[redacted]

* Resource Group
[✓ Create new] Use existing
DocumentDBResourceGroup

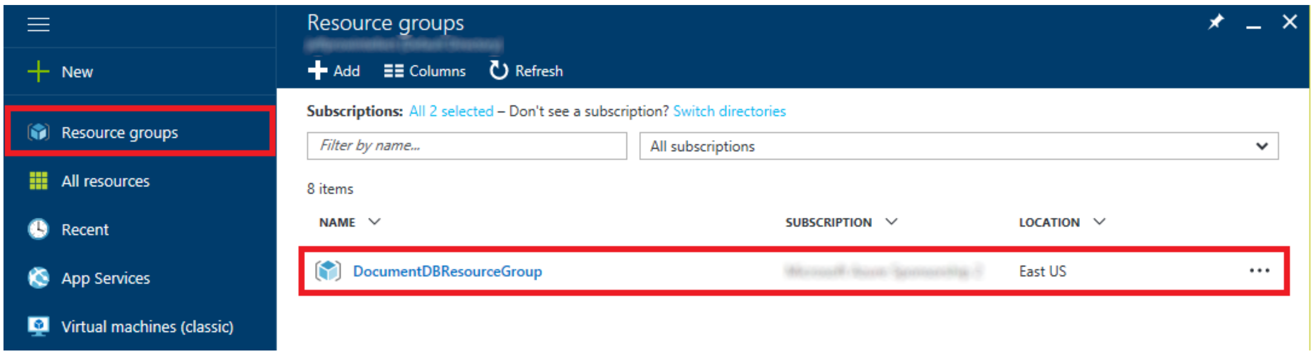
* Location
East US

Pin to dashboard

Create Automation options

Specifying DocumentDB parameters

4. Click **Resource groups** in the ribbon on the left side of the portal, and then click the resource group created for the DocumentDB account.



Resource groups
[redacted]

Subscriptions: All 2 selected – Don't see a subscription? Switch directories

Filter by name... All subscriptions

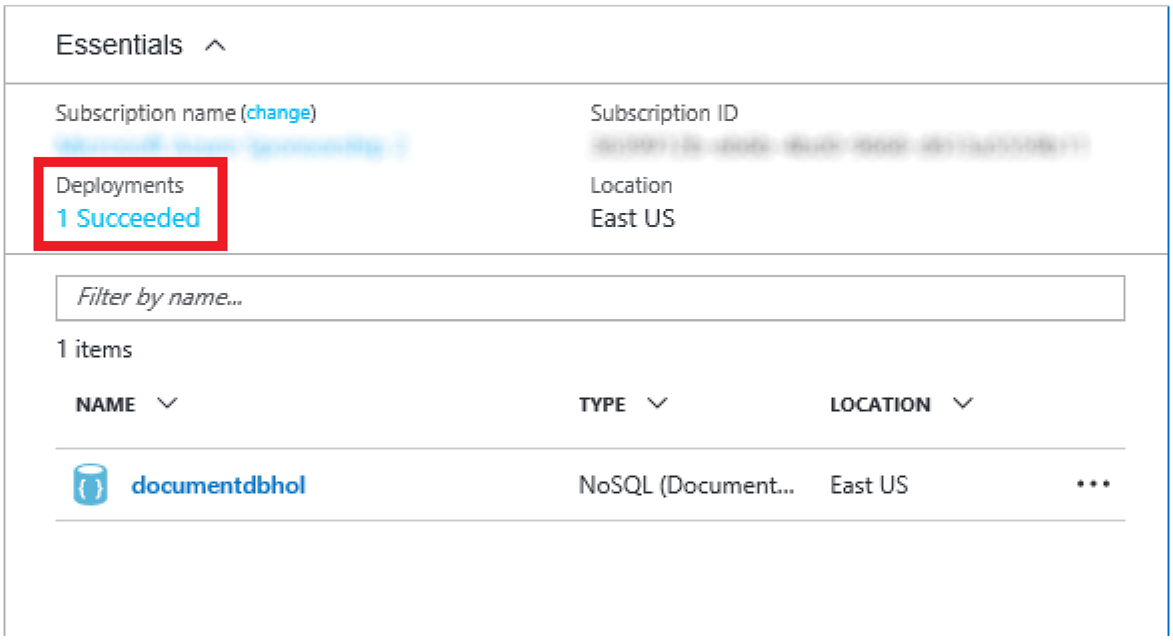
8 items

NAME	SUBSCRIPTION	LOCATION
DocumentDBResourceGroup	[redacted]	East US

Opening the resource group

5. Wait until "Deploying" changes to "Succeeded," indicating that the DocumentDB account has been deployed.

Refresh the page in the browser every now and then to update the deployment status. Clicking the **Refresh** button in the resource-group blade refreshes the list of resources in the resource group, but does not reliably update the deployment status.



Essentials

Subscription name (change)
[redacted]

Subscription ID
[redacted]

Deployments
1 Succeeded

Location
East US

Filter by name...

1 items

NAME	TYPE	LOCATION	
documentdbhol	NoSQL (Document...	East US	•••

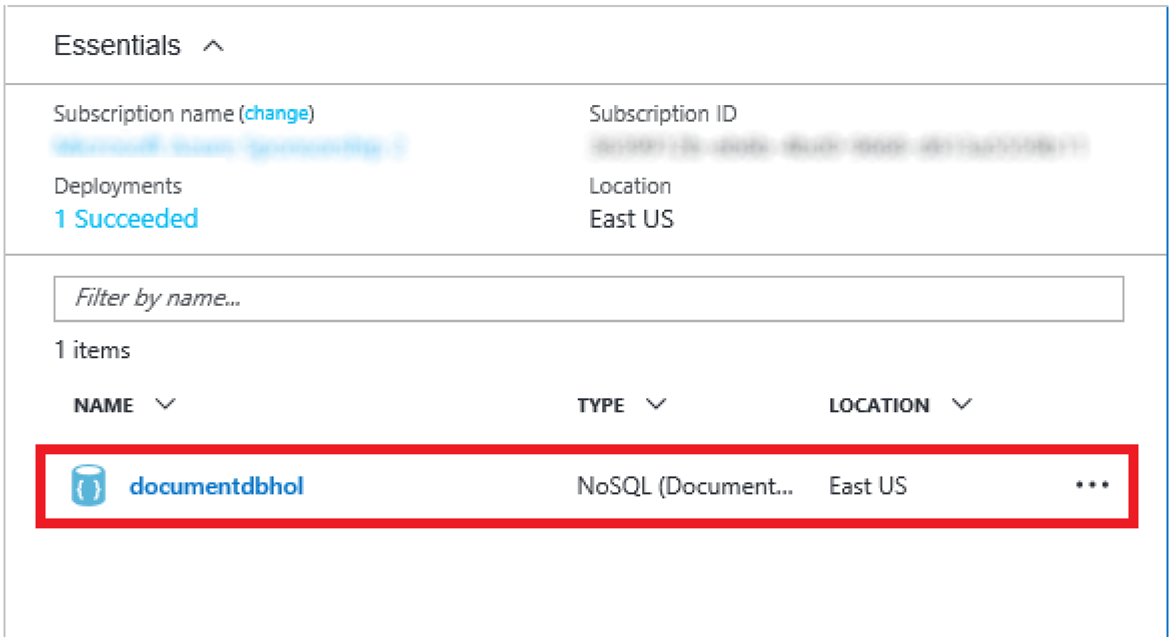
Viewing the deployment status

Your DocumentDB account is now provisioned and ready for you to start working with it.

Exercise 2: Create a database and collections

Now that you've deployed a DocumentDB account, the next step is to create a database and add collections to it in preparation for storing documents. In this exercise, you will create a database and add three collections to it for storing information about customers, products, and orders.

1. Click the DocumentDB account that you deployed in Exercise 1.



Essentials

Subscription name (change)
[redacted]

Subscription ID
[redacted]

Deployments
1 Succeeded

Location
East US

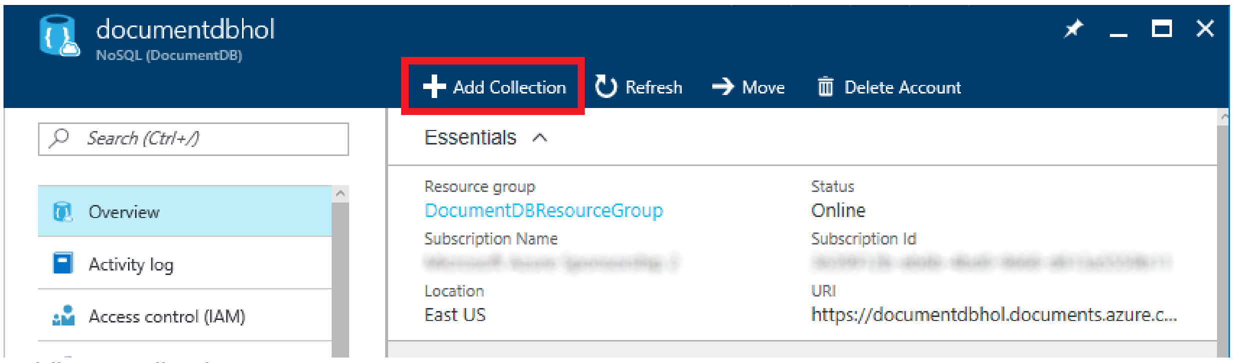
Filter by name...

1 items

NAME	TYPE	LOCATION	
documentdbhol	NoSQL (Document...	East US	•••

Opening the DocumentDB account

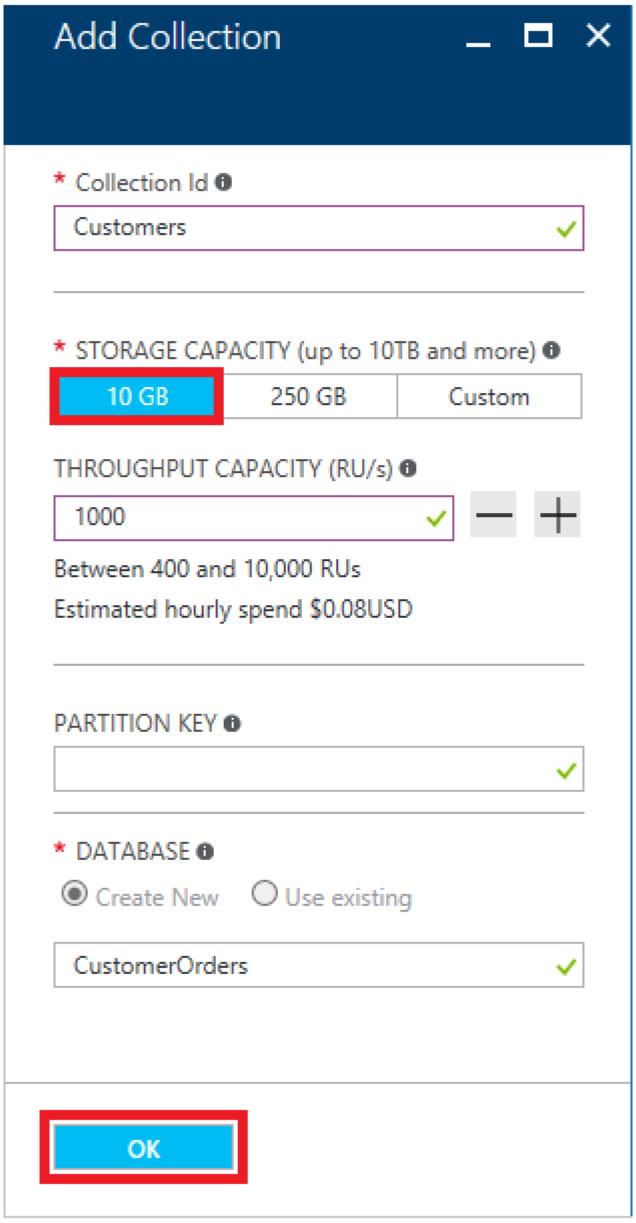
2. Click **+ Add Collection**.



Resource group: DocumentDBResourceGroup
Status: Online
Subscription Name: [redacted]
Subscription Id: [redacted]
Location: East US
URI: https://documentdbhol.documents.azure.c...

Adding a collection

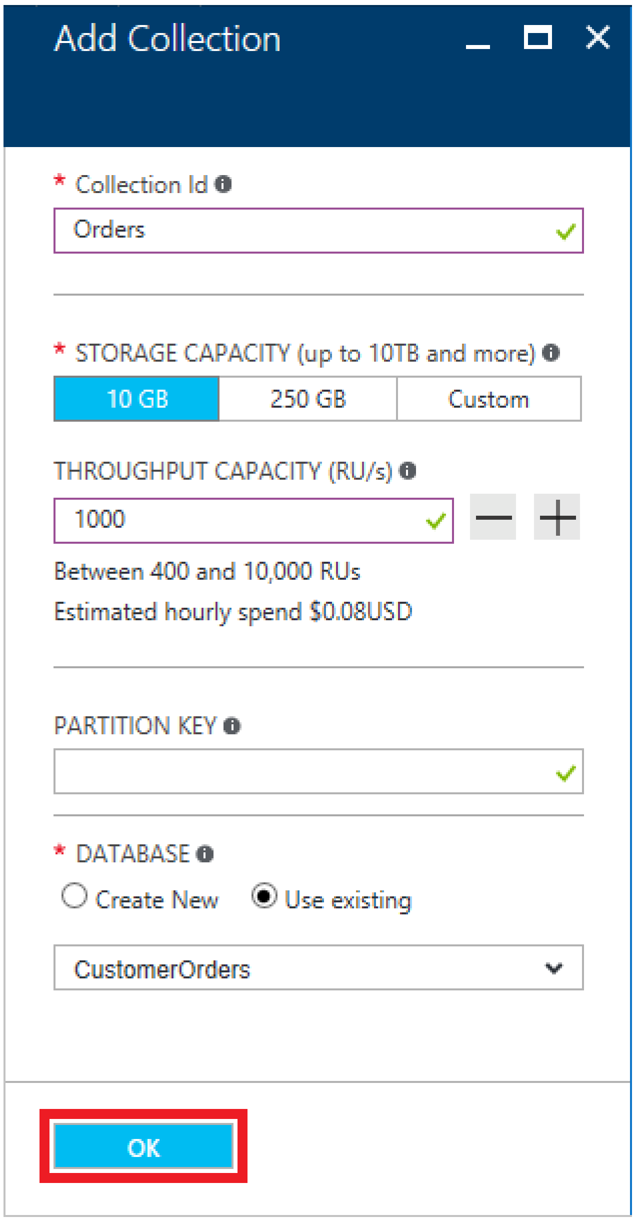
3. Enter "Customers" (without quotation marks) as the **Collection Id** and select **10 GB** as the **STORAGE CAPACITY**. Select **Create New** under **DATABASE** and specify "CustomerOrders" as the database name. Then click the **OK** button.



* Collection Id: Customers
* STORAGE CAPACITY (up to 10TB and more): [✓ 10 GB] 250 GB Custom
THROUGHPUT CAPACITY (RU/s): 1000
Between 400 and 10,000 RUs
Estimated hourly spend $0.08USD
PARTITION KEY:
* DATABASE: [✓ Create New] Use existing
CustomerOrders

Creating a Customers collection

4. Click **+ Add Collection** again. Fill in the form as shown below to create a second collection named "Orders." Be sure to add it to the existing database ("CustomerOrders") rather than create a new database. Then click **OK**.



* Collection Id: Orders

* STORAGE CAPACITY (up to 10TB and more): [✓ 10 GB] 250 GB Custom

THROUGHPUT CAPACITY (RU/s): 1000

Between 400 and 10,000 RUs

Estimated hourly spend $0.08USD

PARTITION KEY:

* DATABASE: Create New [✓ Use existing]

CustomerOrders

Adding an Orders collection

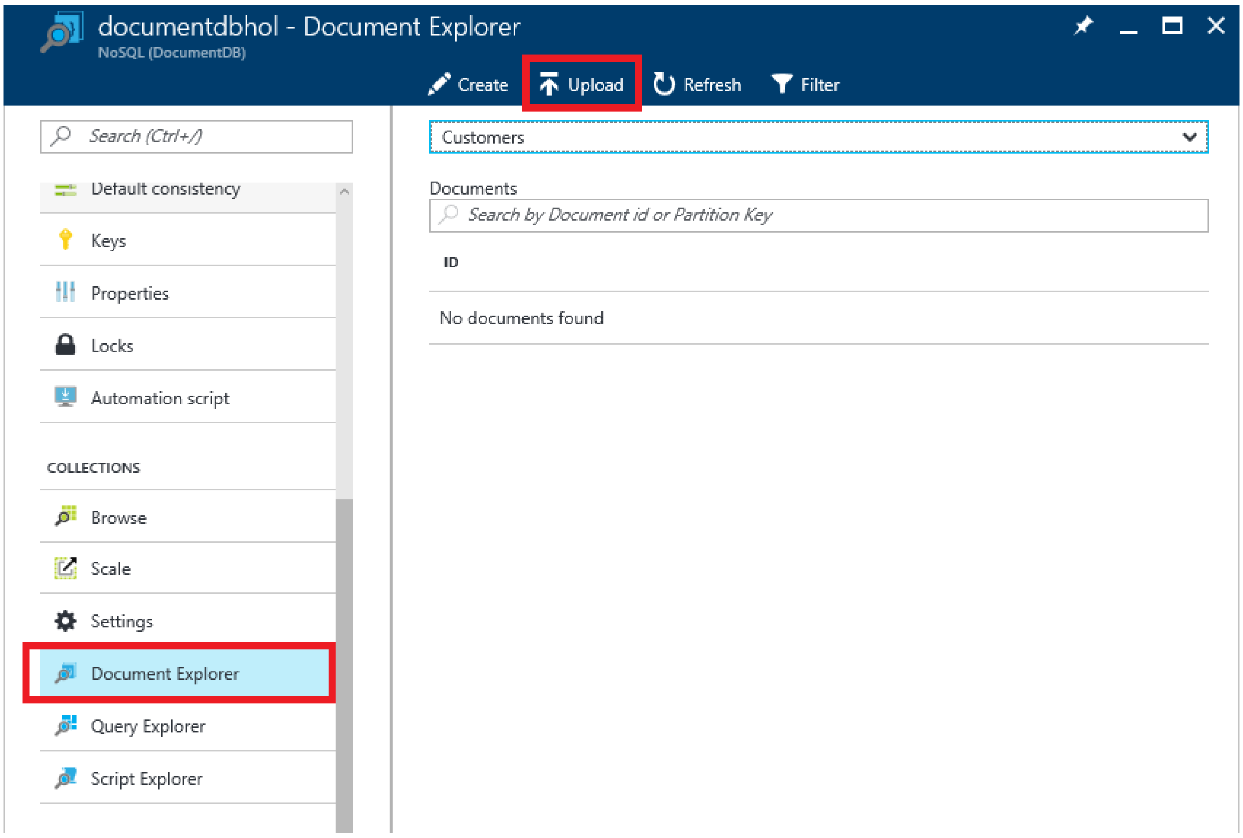
5. Click **+ Add Collection** again. Create a third collection named "Products" with the same settings as the "Customers" and "Orders" collections. Once more, be sure to add it to the existing database ("CustomerOrders") rather than create a new database.

The next step is to upload documents containing data regarding customers, products, and orders to the collections you created.

Exercise 3: Populate collections with documents

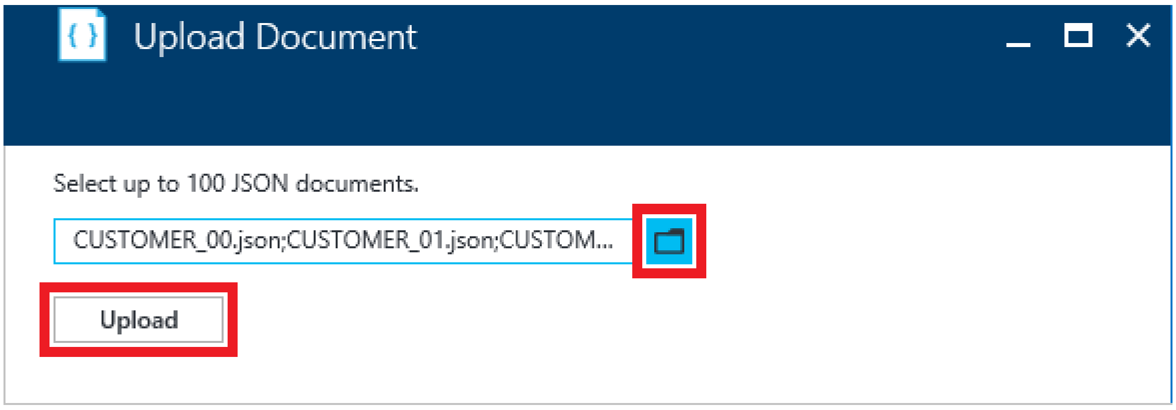
There are several ways to populate DocumentDB collections with documents, including programmatic import via the Azure SDK and the Microsoft DocumentDB Data Migration Tool. In this exercise, you will populate your collections with data by uploading JSON documents through the Azure Portal.

1. Click **Document Explorer** in the menu on the left. Make sure **Customers** is selected in the drop-down list of collections, and click **Upload**.



Opening Document Explorer

2. Click the folder icon. Select all of the files in this lab's "Resources/Customers" folder, and then click the **Upload** button.


Upload Document
Select up to 100 JSON documents.
CUSTOMER_00.json;CUSTOMER_01.json;CUSTOM...
Upload

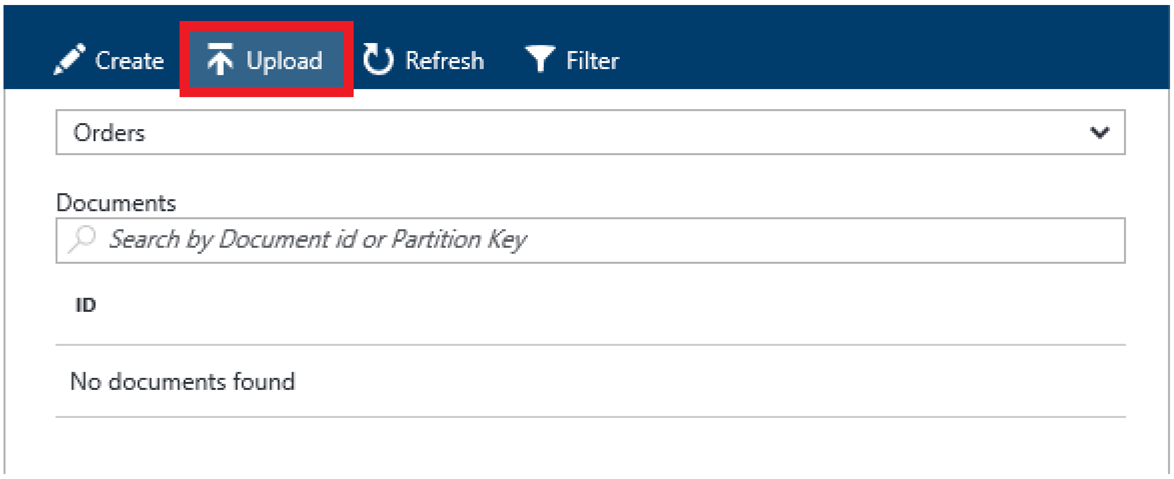
Uploading customer data

Each of the files you uploaded is a JSON document containing information about one customer. Here is one of those files:

```
{

  "CustomerID": "ALFKI",
  "CompanyName": "Alfreds Futterkiste",
  "ContactName": "Maria Anders",
  "ContactTitle": "Sales Representative",
  "Address": "Obere Str. 57",
  "City": "Berlin",
  "Region": null,
  "PostalCode": "12209",
  "Latitude": 52.54971,
  "Longitude": 13.49041,
  "Country": "Germany",
  "Phone": "030-0074321",
  "Fax": "030-0076545"
}
```

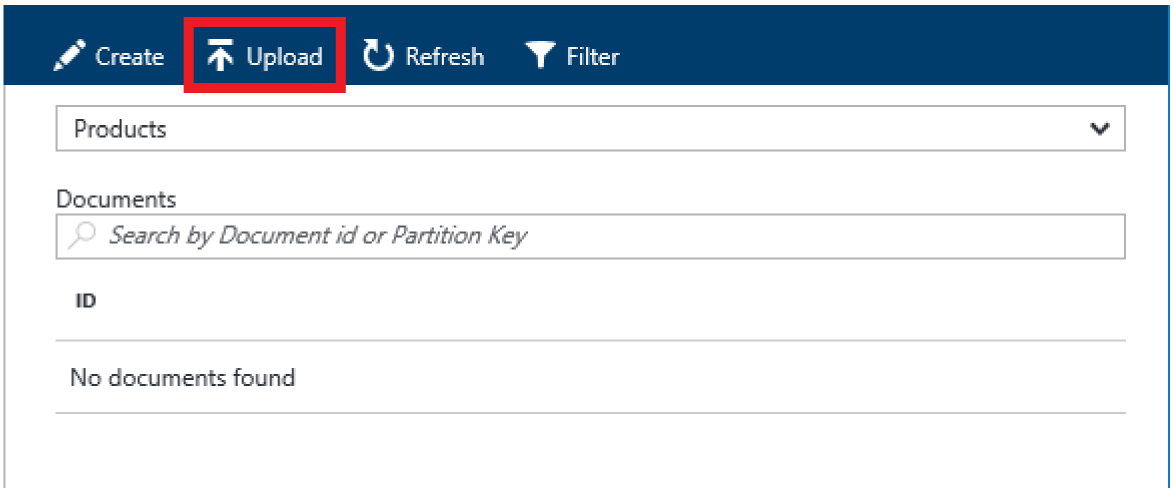
There are 91 files in all, so the Customers collection is now populated with data for 91 customers.

3. Close the "Upload Document" blade and return to the "Document Explorer" blade. Select **Orders** from the drop-down list of collections and click **Upload**. Then upload all of the files in this lab's "Resources/Orders" folder to the Orders collection.



Uploading order data

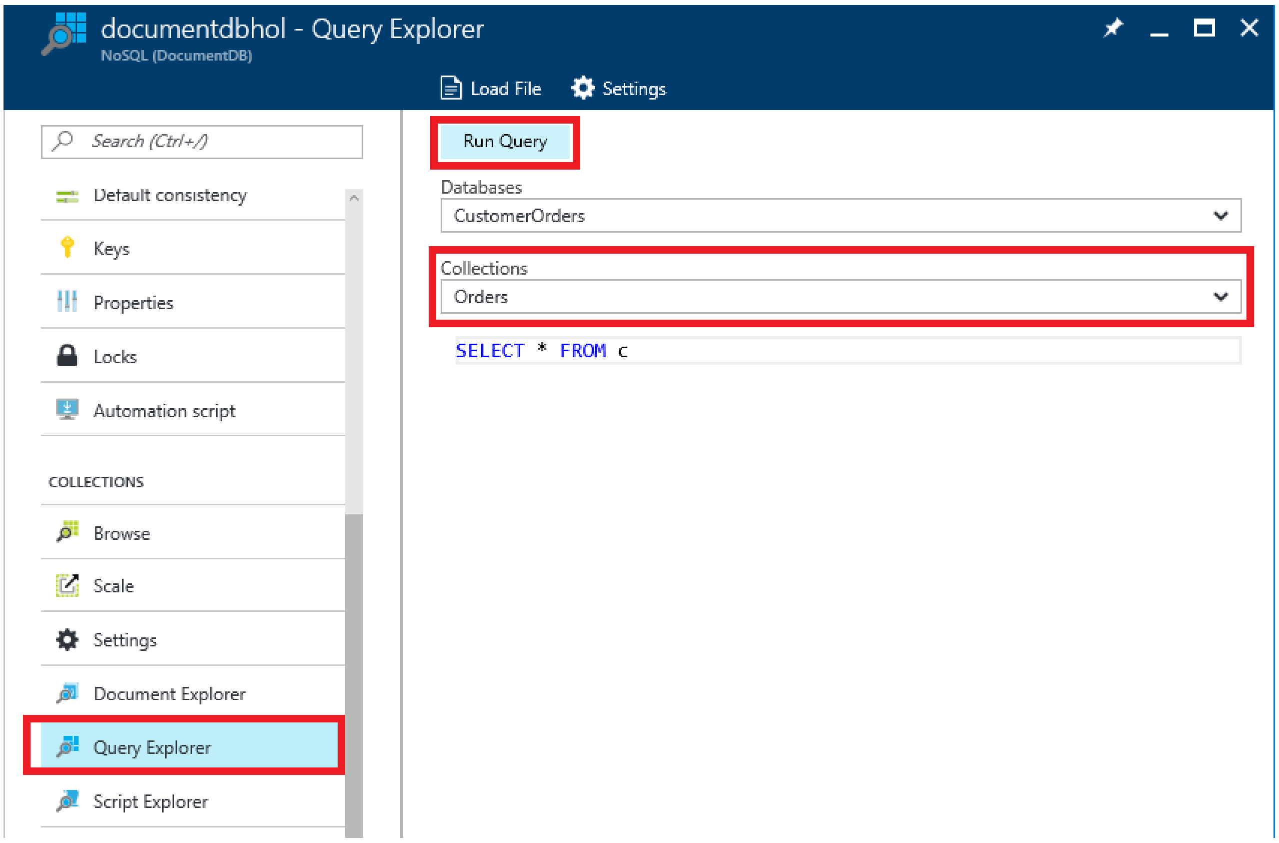
4. Repeat this process to upload all of the files in this lab's "Resources/Products" folder to the Products collection.



Uploading product data

5. The next step is to validate the document uploads by querying one or more of the collections. Click **Query Explorer** in the menu on the left. Select **Orders** in the drop-down list of collections, and then click the **Run Query**.

   Although you won't use DocumentDB's rich query capabilities directly in this lab, be aware that DocumentDB supports a variation of the SQL query language for extracting information from JSON data. For more information, and plenty of examples, see https://docs.microsoft.com/en-us/azure/documentdb/documentdb-sql-query.

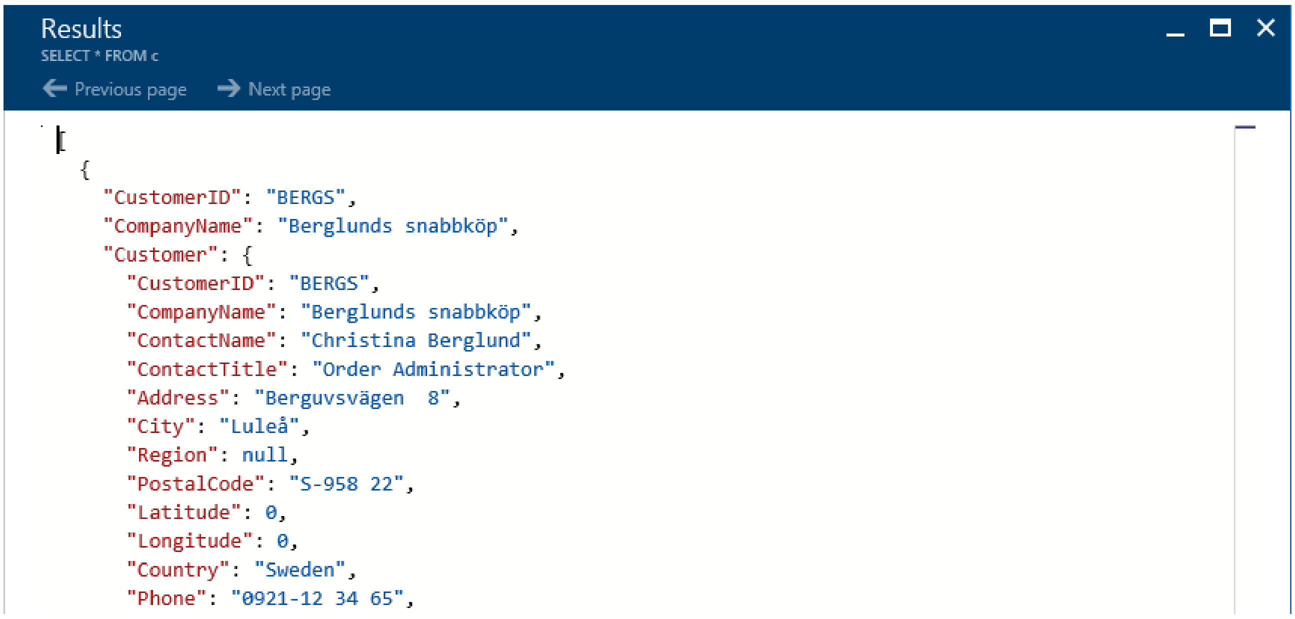


Databases: CustomerOrders
Collections: Orders

```
SELECT * FROM c
```

Querying the Orders collection

6. Confirm that you see the query results below.



```
[
  {
    "CustomerID": "BERGS",
    "CompanyName": "Berglunds snabbköp",
    "Customer": {
      "CustomerID": "BERGS",
      "CompanyName": "Berglunds snabbköp",
      "ContactName": "Christina Berglund",
      "ContactTitle": "Order Administrator",
      "Address": "Berguvsvägen  8",
      "City": "Luleå",
      "Region": null,
      "PostalCode": "S-958 22",
      "Latitude": 0,
      "Longitude": 0,
      "Country": "Sweden",
      "Phone": "0921-12 34 65",
```

Query results

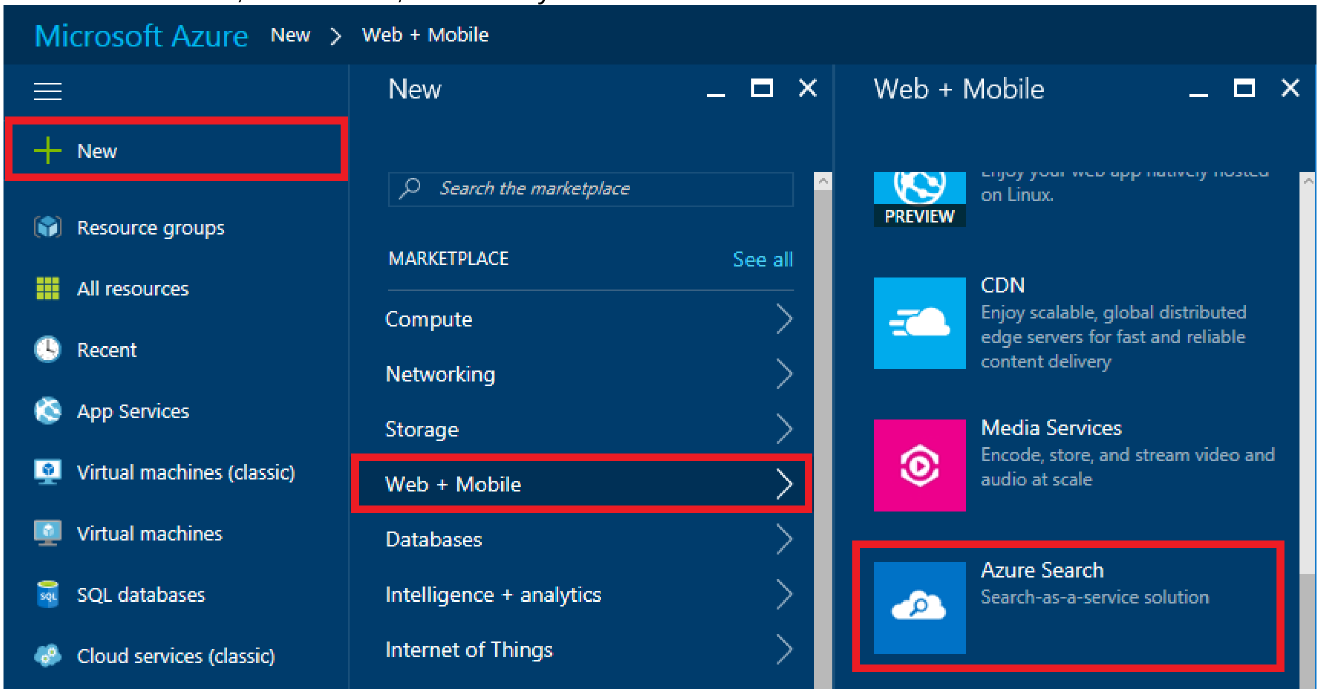
7. If you would like to confirm that the product and customer uploads succeeded too, run the same query against the Products and Customers collections.

   The database that you created now has three collections that are populated with data. Now let's index the data so searches can be performed quickly.

Exercise 4: Connect Azure Search

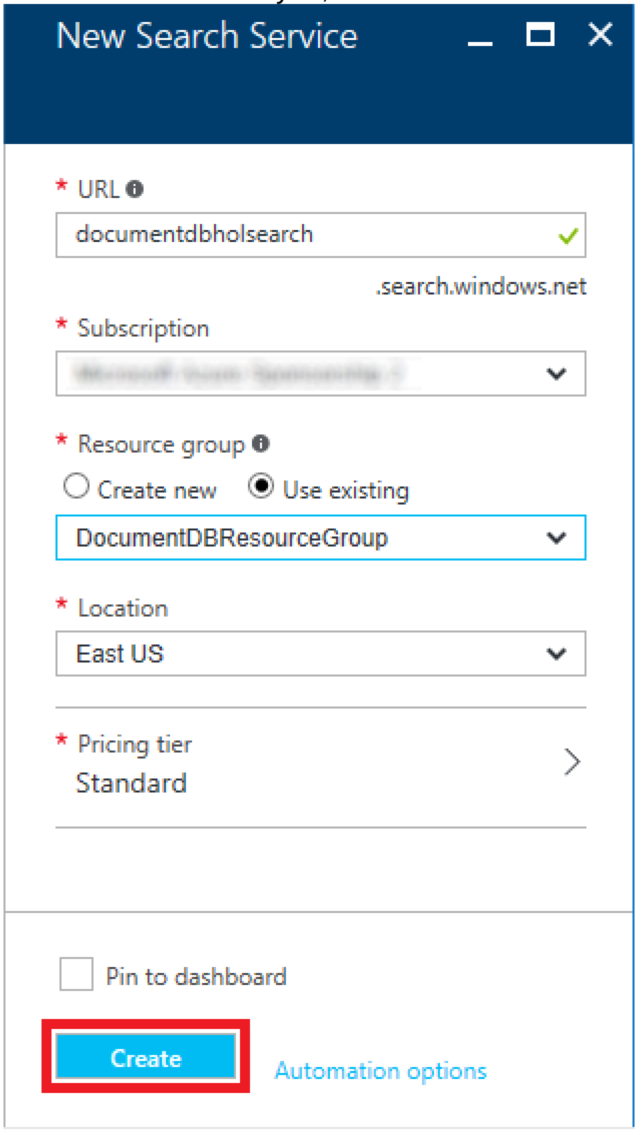
One of the benefits of using DocumentDB is that it integrates easily with Azure Search. Azure Search is a managed Search-as-a-Service solution that delegates server and infrastructure management to Microsoft and lets you index data sources for lightning-fast searches. Search can be accessed through a simple REST API or with the Azure Search SDK, enabling you to employ it in Web apps, mobile apps, and other types of applications. In this exercise, you will deploy an Azure Search service and connect it to the DocumentDB database that you created in Exercise 2.

1. In the Azure Portal, click **+ New**, followed by **Web + Mobile** and then **Azure Search**.



Creating a new Azure Search service

2. In the "New Search Service" blade, give the service a unique name in the **URL** box and make sure a green check mark appears next to it. (You can only use numbers and lowercase letters since the name becomes part of a DNS name.) Select **Use existing** under **Resource group** and select the resource group you created for the DocumentDB account in Exercise 1. Select the **Location** nearest you, and then click the **Create** button.


New Search Service

* URL: documentdbholsearch .search.windows.net ✓

* Subscription: [illegible]

* Resource group: Create new [✓ Use existing]

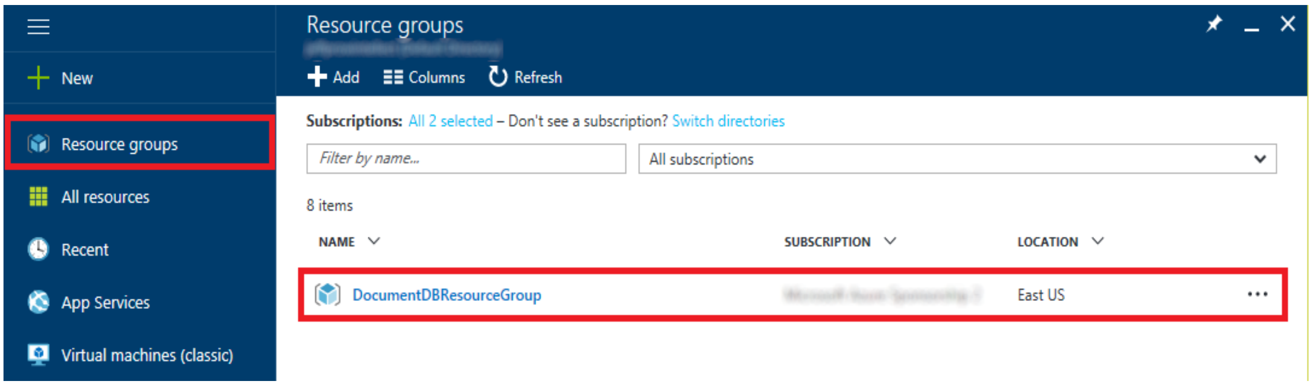
DocumentDBResourceGroup

* Location: East US

* Pricing tier: Standard

Specifying Search parameters

3. Click **Resource groups** in the ribbon on the left side of the portal, and then click the resource group containing the DocumentDB account and Search service.



Subscriptions: All 2 selected – Don't see a subscription? Switch directories

Filter by name... | All subscriptions

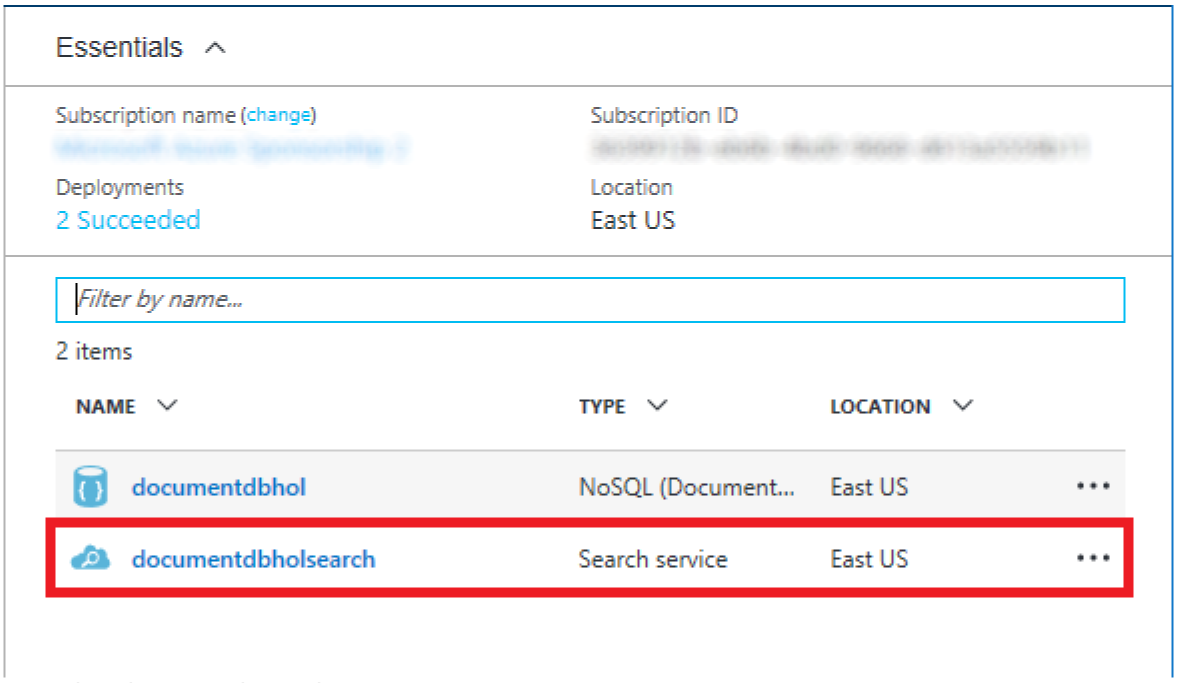
8 items

NAME	SUBSCRIPTION	LOCATION
DocumentDBResourceGroup		East US

Opening the resource group

4. Wait until "Deploying" changes to "Succeeded," indicating that the Search service has been deployed. Then click the Search service.

   Refresh the page in the browser every now and then to update the deployment status. Clicking the **Refresh** button in the resource-group blade refreshes the list of resources in the resource group, but does not reliably update the deployment status.


Essentials

Subscription name (change)

Subscription ID

Deployments
2 Succeeded

Location
East US

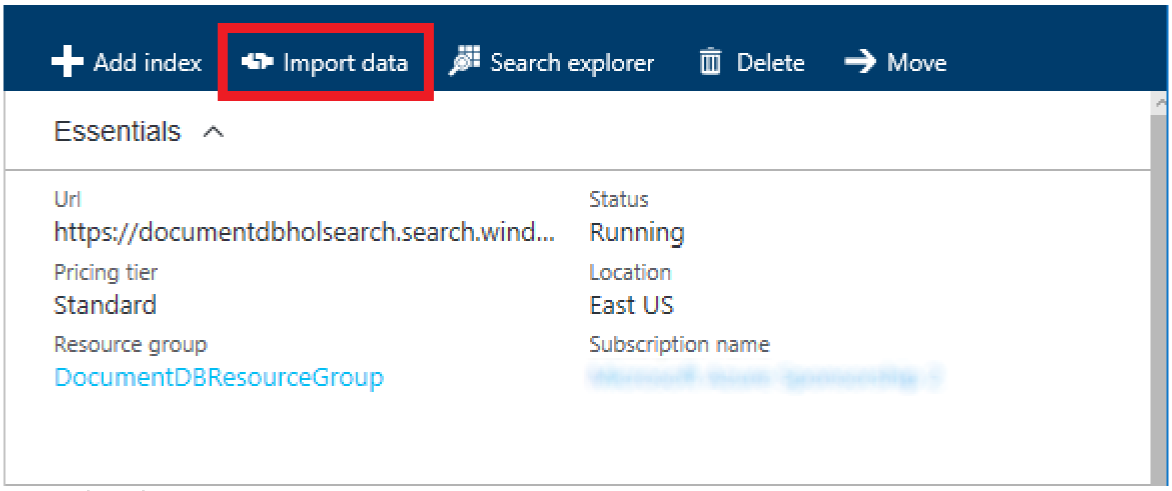
Filter by name...

2 items

NAME	TYPE	LOCATION
documentdbhol	NoSQL (Document...	East US
documentdbholsearch	Search service	East US

Opening the Search service

5. Click **Import data**.

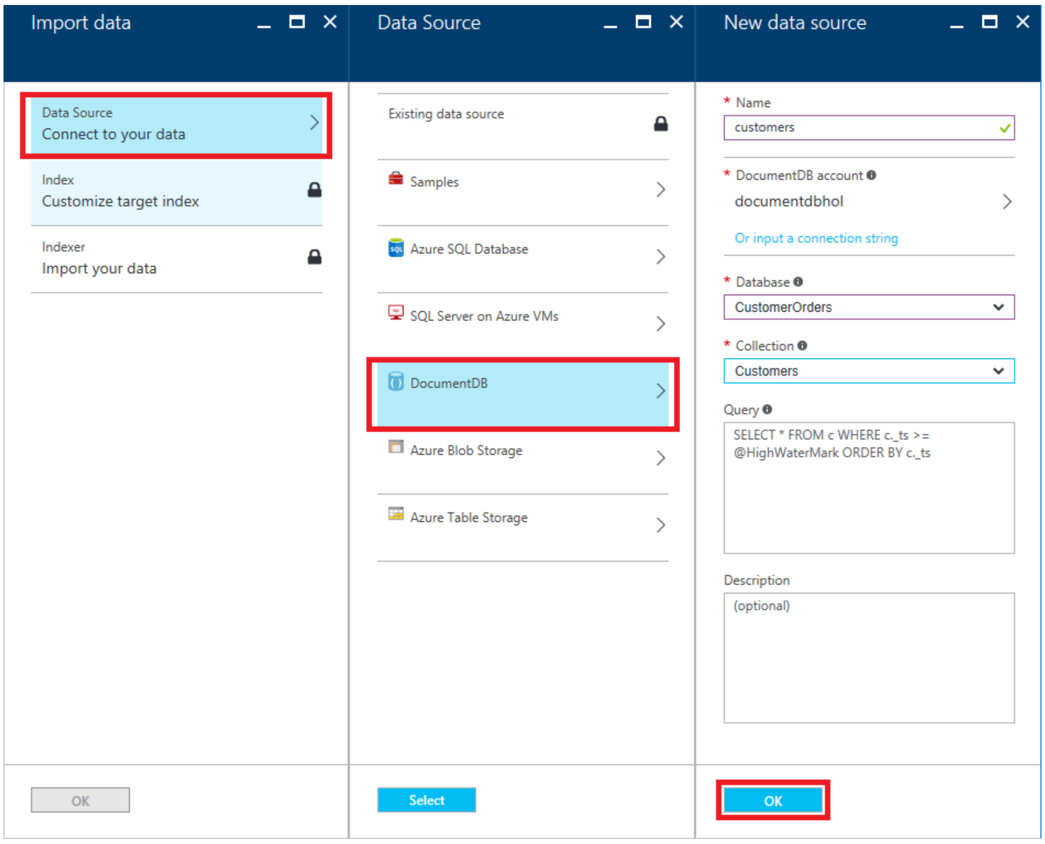


Essentials

Url
https://documentdbholsearch.search.wind...
Status
Running
Pricing tier
Standard
Location
East US
Resource group
DocumentDBResourceGroup
Subscription name

Importing data

6. Click **Data Source**, followed by **DocumentDB**. In the "New data source" blade, type "customers" (without quotation marks) into the **Name** field. Click **Select an account** and select the DocumentDB account you created in Exercise 1. Select the **CustomerOrders** database and the **Customers** collection. Then click **OK**.



* Name
customers

* DocumentDB account
documentdbhol
Or input a connection string

* Database
CustomerOrders

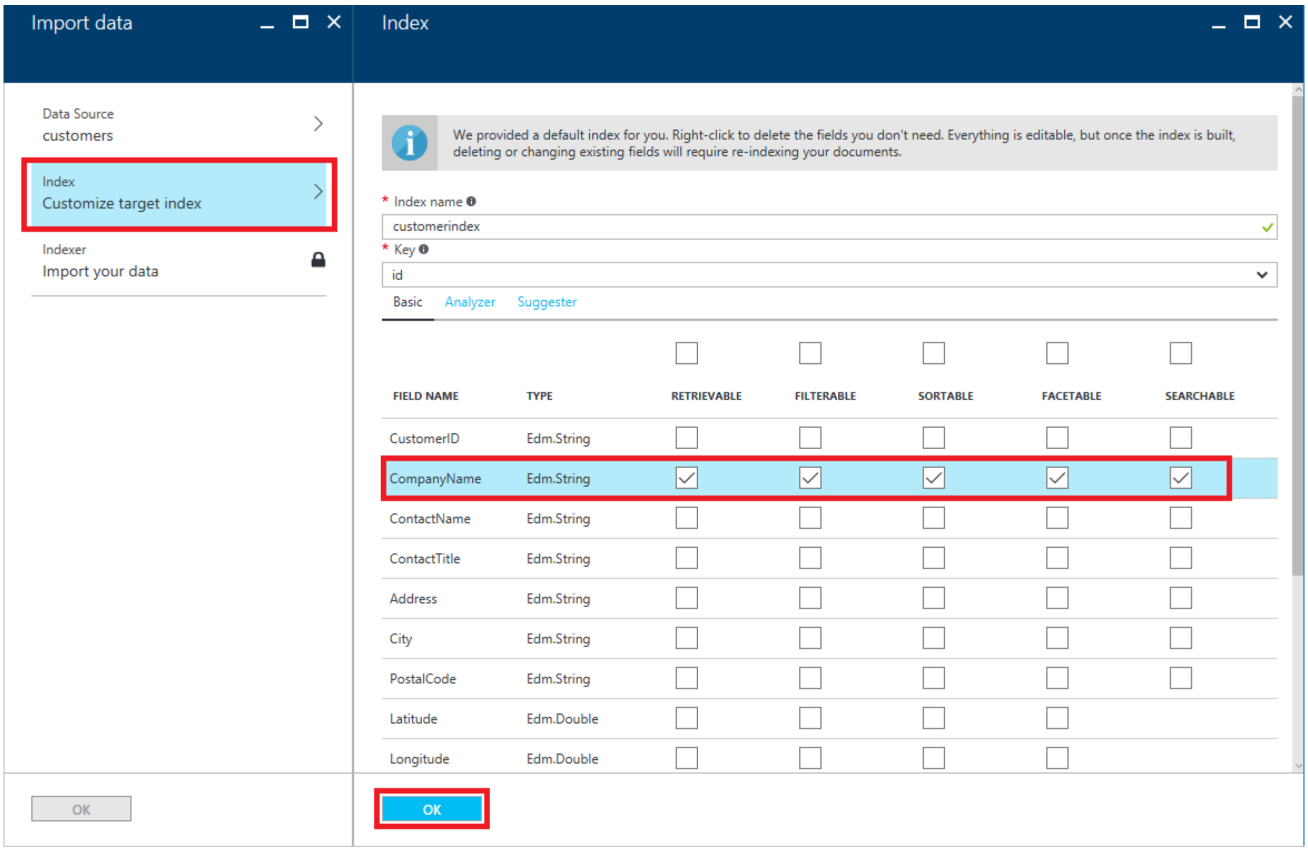
* Collection
Customers

Query
SELECT * FROM c WHERE c._ts >= @HighWaterMark ORDER BY c._ts

Description
(optional)

Connecting to a data source

7. Click **Index** in the "Import data" blade. In the "Index" blade, type "customerindex" (without quotation marks) into the **Index name** field, and then check all five boxes in the CompanyName row. Then click **OK**.



We provided a default index for you. Right-click to delete the fields you don't need. Everything is editable, but once the index is built, deleting or changing existing fields will require re-indexing your documents.

* Index name: customerindex

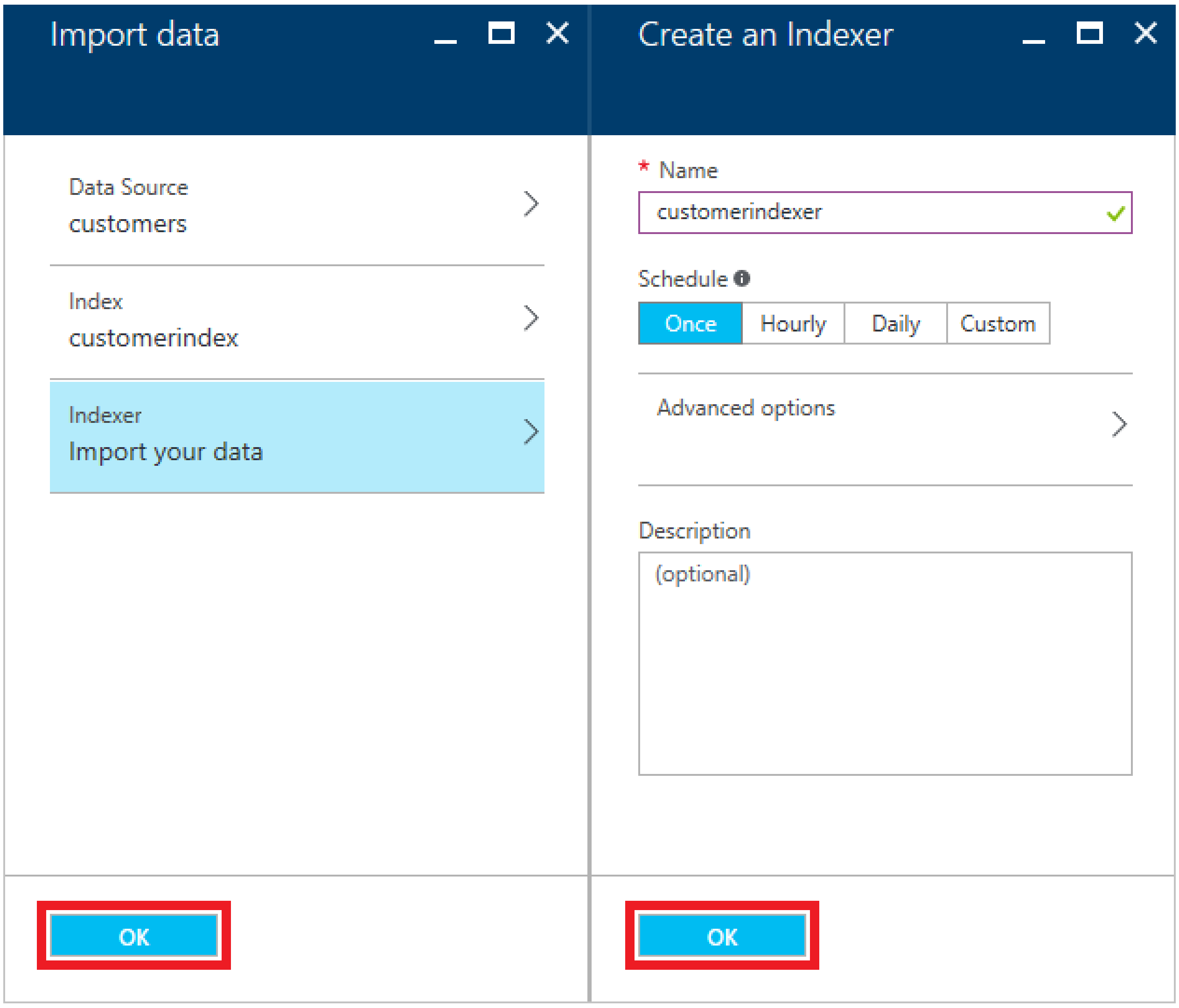
* Key: id

Basic | Analyzer | Suggester

FIELD NAME	TYPE	RETRIEVABLE	FILTERABLE	SORTABLE	FACETABLE	SEARCHABLE
CustomerID	Edm.String	☐	☐	☐	☐	☐
CompanyName	Edm.String	✓	✓	✓	✓	✓
ContactName	Edm.String	☐	☐	☐	☐	☐
ContactTitle	Edm.String	☐	☐	☐	☐	☐
Address	Edm.String	☐	☐	☐	☐	☐
City	Edm.String	☐	☐	☐	☐	☐
PostalCode	Edm.String	☐	☐	☐	☐	☐
Latitude	Edm.Double	☐	☐	☐	☐	
Longitude	Edm.Double	☐	☐	☐	☐	

Configuring a search index

8. Click **Indexer** in the "Import data" blade. In the "Indexer" blade, type "customerindexer" into the **Name** field and click **OK**. Finish up by clicking the **OK** button at the bottom of the "Import data" blade.



Import data

Data Source: customers

Index: customerindex

Indexer: Import your data

Create an Indexer

* Name: customerindexer

Schedule: [✓ Once] Hourly Daily Custom

Advanced options

Description: (optional)

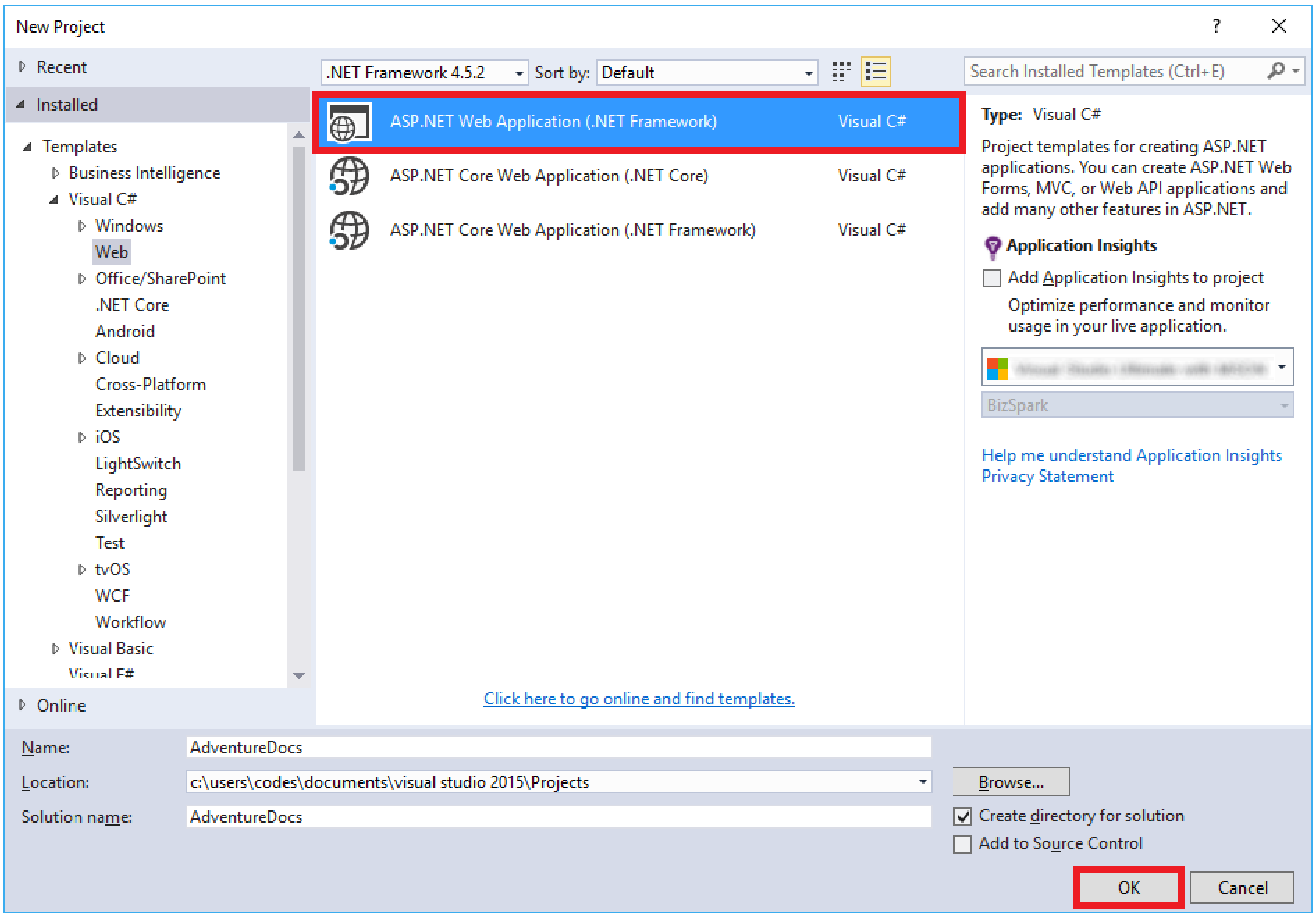
Configuring a search indexer

With the DocumentDB database deployed and an Azure Search service connected to it, it is time to put both to work by building a Web app that uses them to display customer, product, and order information.

Exercise 5: Build an Azure Web App

Azure Web Apps allow you to quickly and easily deploy Web sites built with tools and languages you're familiar with. In this exercise, you will build an ASP.NET MVC Web app with Visual Studio and configure it so that it can be deployed to the cloud as an Azure Web App. The app will connect to the DocumentDB database deployed and populated with data in previous exercises and provide a browser-based front-end for viewing and searching the data.

1. Start Visual Studio 2015 and use the **File -> New -> Project** command to create a new Visual C# ASP.NET Web Application project named "AdventureDoc" (short for "Adventure Works Documents").



New Project

- Recent
- Installed
  - Templates
    - Business Intelligence
    - Visual C#
      - Windows
      - Web
      - Office/SharePoint
      - .NET Core
      - Android
      - Cloud
      - Cross-Platform
      - Extensibility
      - iOS
      - LightSwitch
      - Reporting
      - Silverlight
      - Test
      - tvOS
      - WCF
      - Workflow
    - Visual Basic
    - Visual F#
- Online

.NET Framework 4.5.2 Sort by: Default

Search Installed Templates (Ctrl+E)

ASP.NET Web Application (.NET Framework)	Visual C#
ASP.NET Core Web Application (.NET Core)	Visual C#
ASP.NET Core Web Application (.NET Framework)	Visual C#

Click here to go online and find templates.

Type: Visual C#

Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

☐ Add Application Insights to project

Optimize performance and monitor usage in your live application.

BizSpark

Help me understand Application Insights
Privacy Statement

Name:	AdventureDocs
Location:	c:\users\codes\documents\visual studio 2015\Projects
Solution name:	AdventureDocs

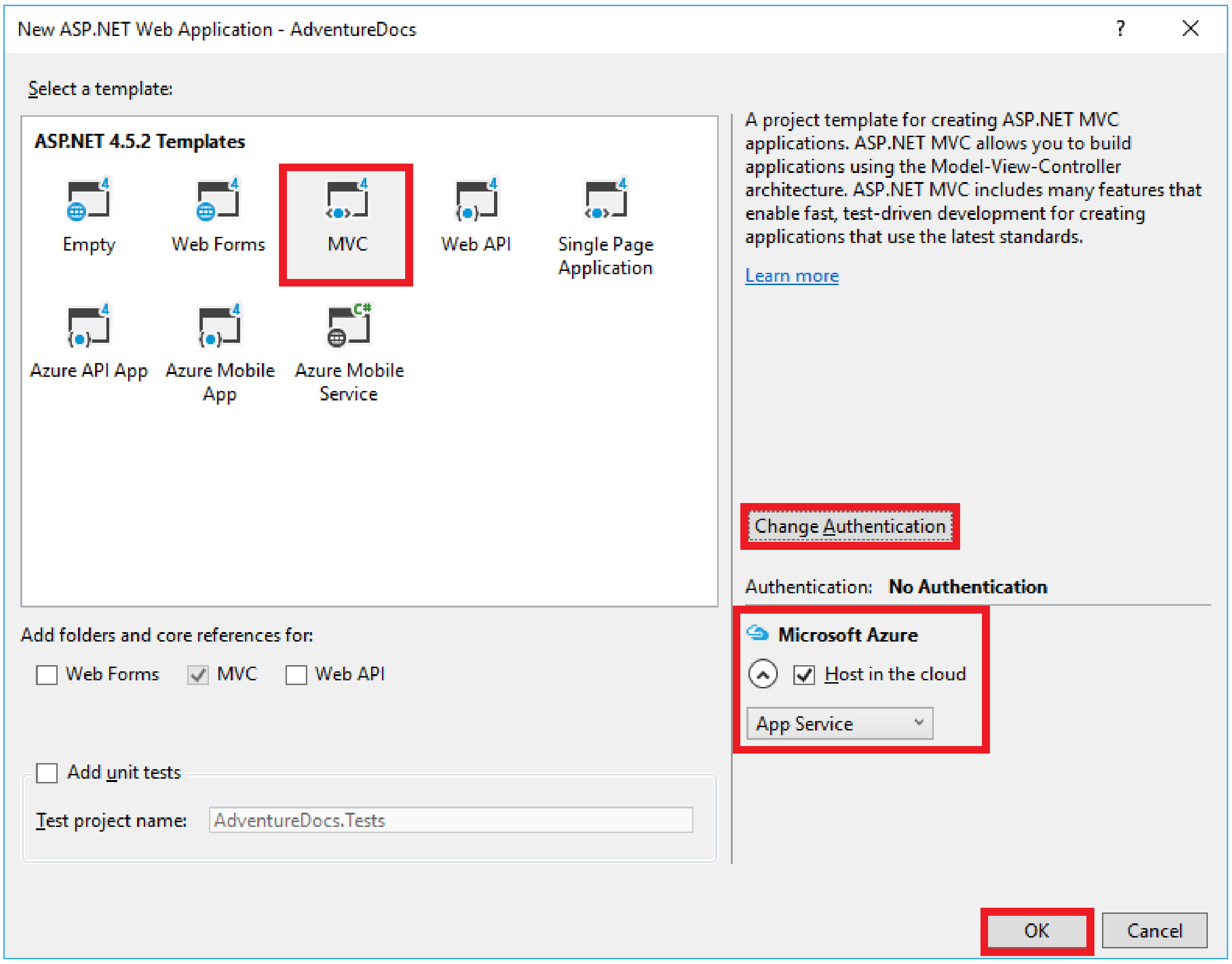
Browse...

☑ Create directory for solution
☐ Add to Source Control

OK Cancel

Creating a new Web Application project

2. In the "New ASP.NET Web Application" dialog, select the **MVC** template. Then click the **Change Authentication** button and select **No Authentication**. (This simplifies the app by omitting authentication infrastructure.) Next, make sure the **Host in the cloud** box is checked and that **App Service** is selected in the drop-down list below the check box. Finally, click **OK**.


New ASP.NET Web Application - AdventureDocs

Select a template:

ASP.NET 4.5.2 Templates

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Authentication: **No Authentication**

Microsoft Azure

Add folders and core references for: Web Forms [✓ MVC] Web API

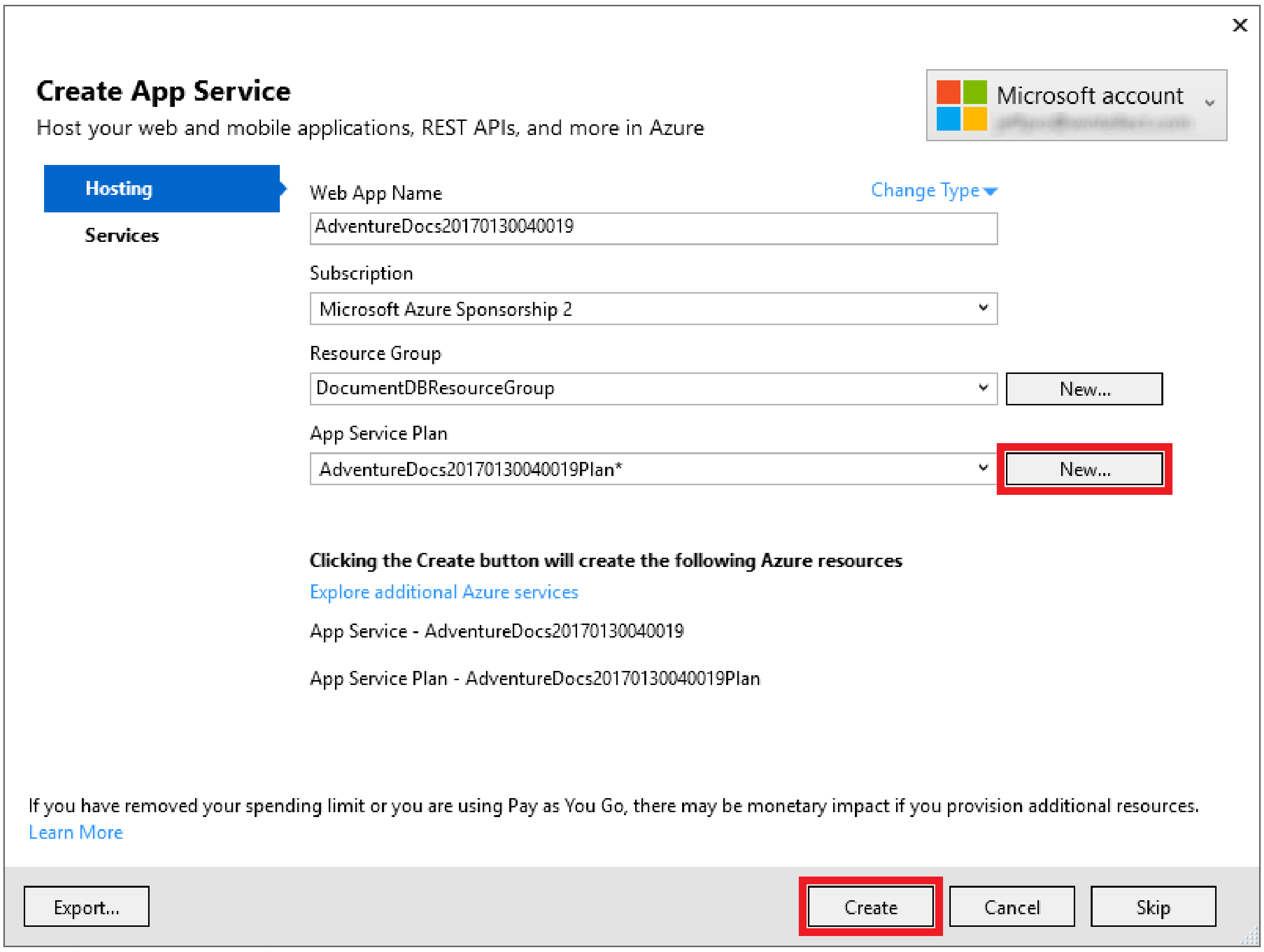
[✓ Host in the cloud]: App Service

Add unit tests

Test project name: AdventureDocs.Tests

Configuring the project

3. In the "Create App Service" dialog, make sure **DocumentDBResourceGroup** is selected under **Resource Group**. (This will add the Azure Web App to the same resource group as the DocumentDB account and the Azure Search service, which is handy because deleting the resource group will delete all three.) Then click the **New** button next to **App Service Plan** and select the location nearest you for hosting the Web App, and **Free** as the **Size.** Click **OK** to dismiss the "Configure App Service Plan" dialog. Then click **Create** at the bottom of the "Create App Service" dialog.


Create App Service

Host your web and mobile applications, REST APIs, and more in Azure

Web App Name: AdventureDocs20170130040019

Subscription: Microsoft Azure Sponsorship 2

Resource Group: DocumentDBResourceGroup

App Service Plan: AdventureDocs20170130040019Plan*

Clicking the Create button will create the following Azure resources

Explore additional Azure services

App Service - AdventureDocs20170130040019

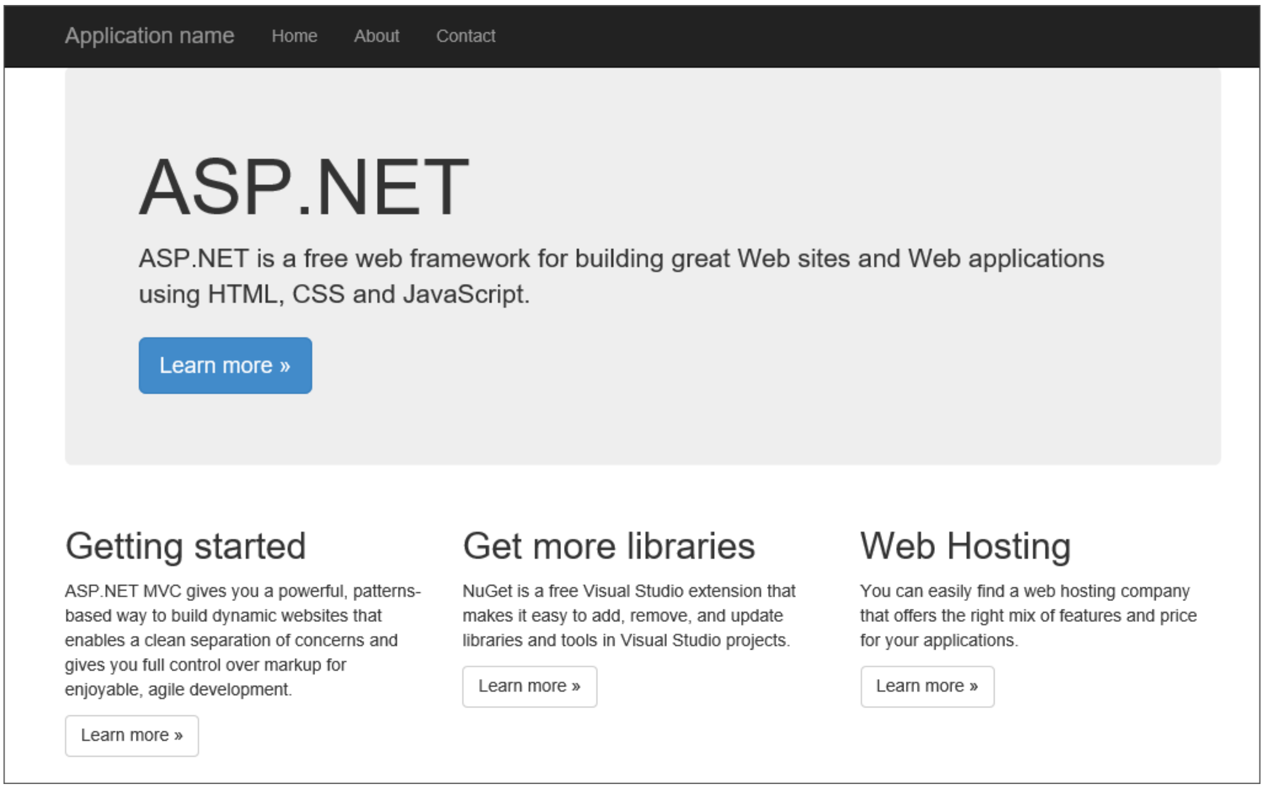
App Service Plan - AdventureDocs20170130040019Plan

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources. Learn More

Creating an App Service

4. Take a moment to review the project structure in the Solution Explorer window. Among other things, there's a folder named "Controllers" that holds the project's MVC controllers, and a folder named "Views" that holds the project's views. You will be working with assets in these folders and others as you implement the application.

5. Use Visual Studio's **Debug -> Start Without Debugging** command (or simply press **Ctrl+F5**) to launch the application in your browser. Here's how the application looks in its present state:



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Get more libraries

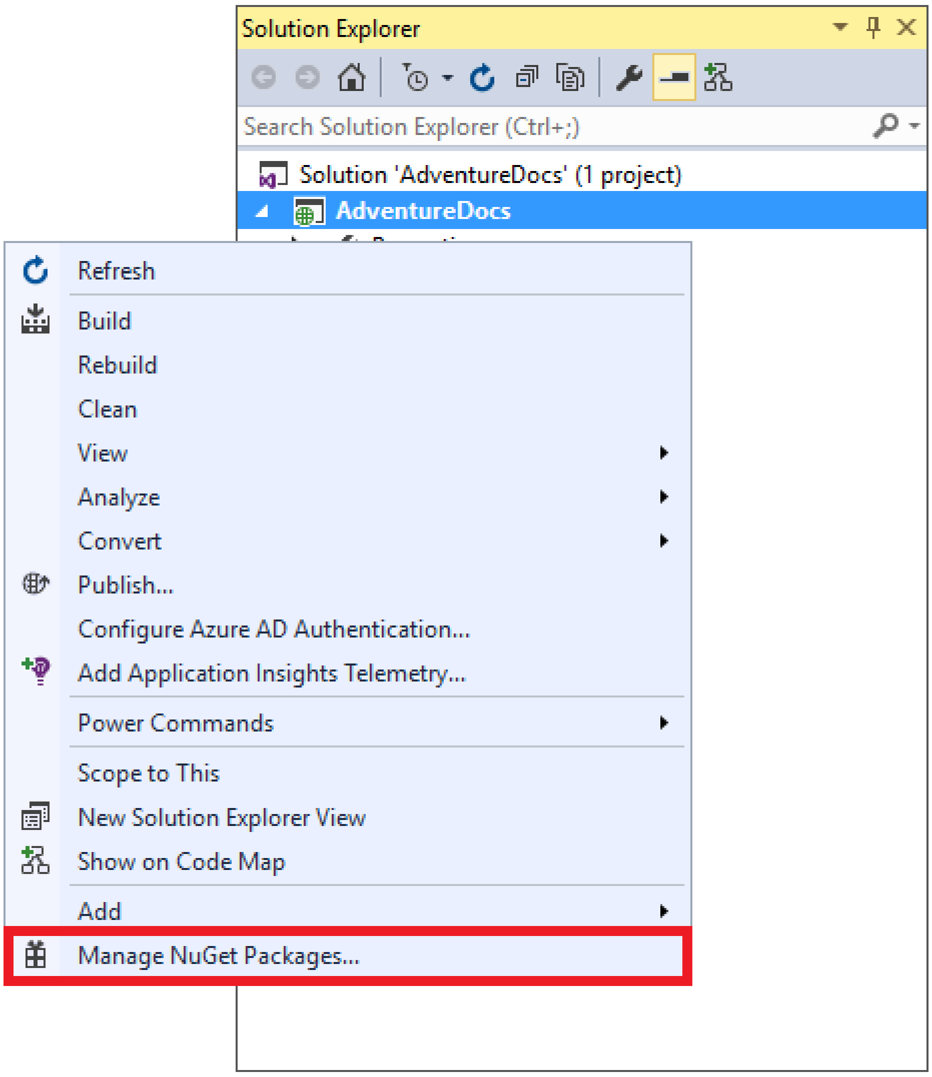
NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

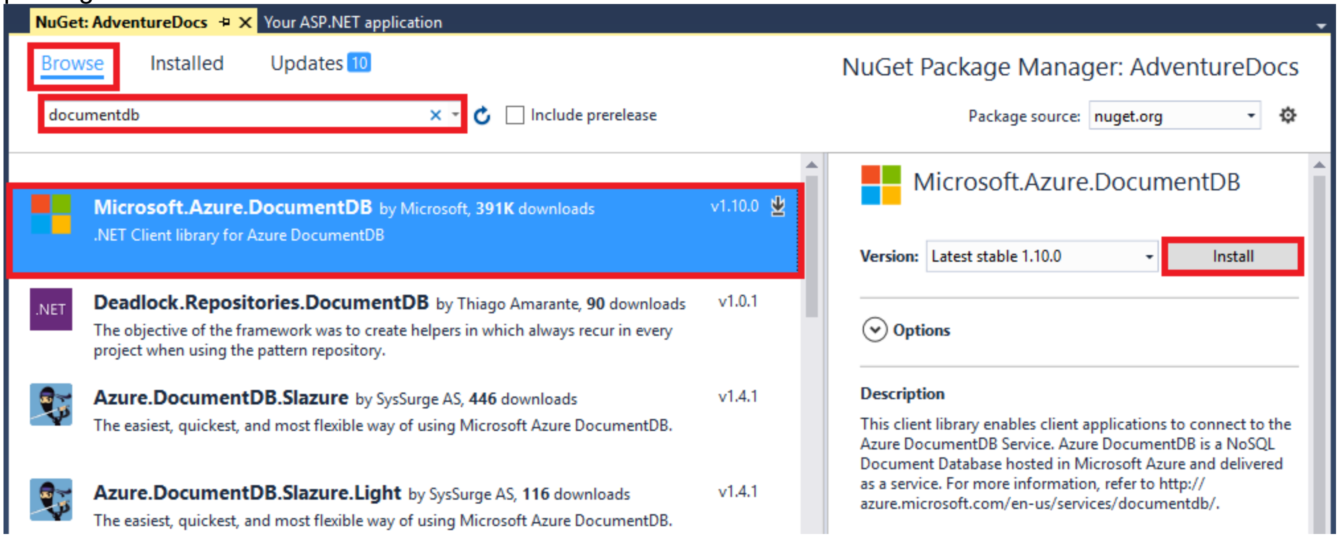
The initial application

6. Close the browser and return to Visual Studio. In the Solution Explorer window, right-click the **AdventureDocs** project and select **Manage NuGet Packages...**.


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'AdventureDocs' (1 project)
AdventureDocs
Refresh
Build
Rebuild
Clean
View
Analyze
Convert
Publish...
Configure Azure AD Authentication...
Add Application Insights Telemetry...
Power Commands
Scope to This
New Solution Explorer View
Show on Code Map
Add
Manage NuGet Packages...

Managing NuGet packages for the project

7. Click **Browse**. Then type "documentdb" (without quotation marks) into the search box. Click **Microsoft.Azure.DocumentDB** to select the Azure DocumentDB .NET Client Library from NuGet. Finally, click **Install** to install the latest stable version of the package. This package contains APIs for accessing Azure DocumentDB from .NET applications. Click **OK** if you're prompted to review changes, and **I Accept** if prompted to accept licenses for downloaded packages.



documentdb

Package source: nuget.org

Microsoft.Azure.DocumentDB by Microsoft, **391K** downloads v1.10.0
.NET Client library for Azure DocumentDB

Deadlock.Repositories.DocumentDB by Thiago Amarante, **90** downloads v1.0.1
The objective of the framework was to create helpers in which always recur in every project when using the pattern repository.

Azure.DocumentDB.Slazure by SysSurge AS, **446** downloads v1.4.1
The easiest, quickest, and most flexible way of using Microsoft Azure DocumentDB.

Azure.DocumentDB.Slazure.Light by SysSurge AS, **116** downloads v1.4.1
The easiest, quickest, and most flexible way of using Microsoft Azure DocumentDB.

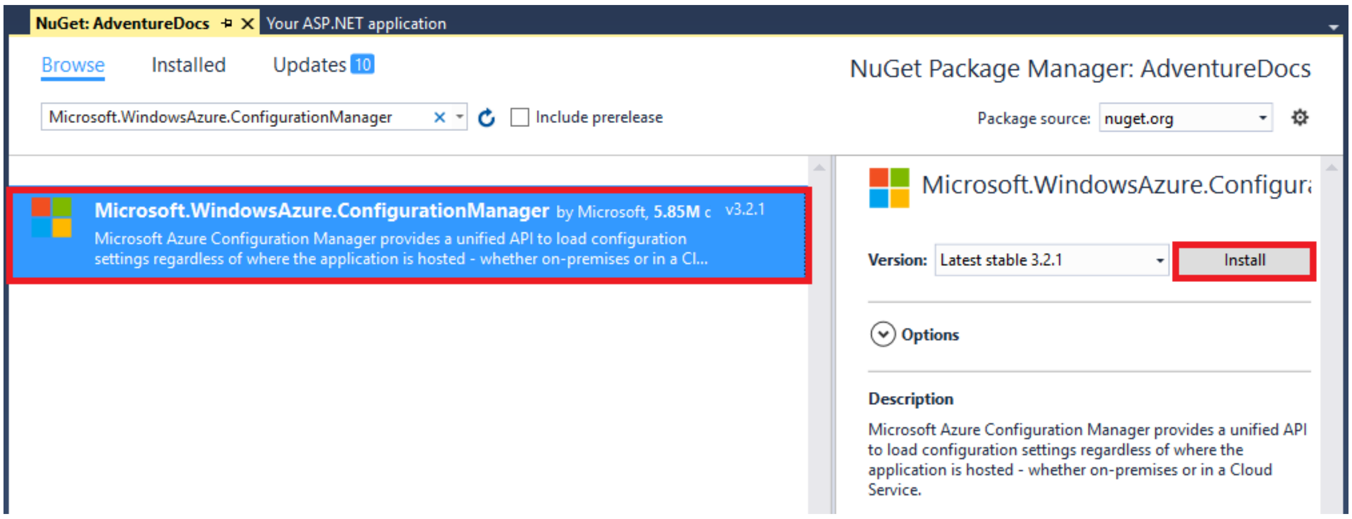
Microsoft.Azure.DocumentDB

Version: Latest stable 1.10.0

Description
This client library enables client applications to connect to the Azure DocumentDB Service. Azure DocumentDB is a NoSQL Document Database hosted in Microsoft Azure and delivered as a service. For more information, refer to http://azure.microsoft.com/en-us/services/documentdb/.

Installing Microsoft.Azure.DocumentDB

8. Repeat this process to add the NuGet package named **Microsoft.WindowsAzure.ConfigurationManager** to the project. This package contains APIs for loading configuration settings. Once more, OK any changes and accept any licenses presented to you.

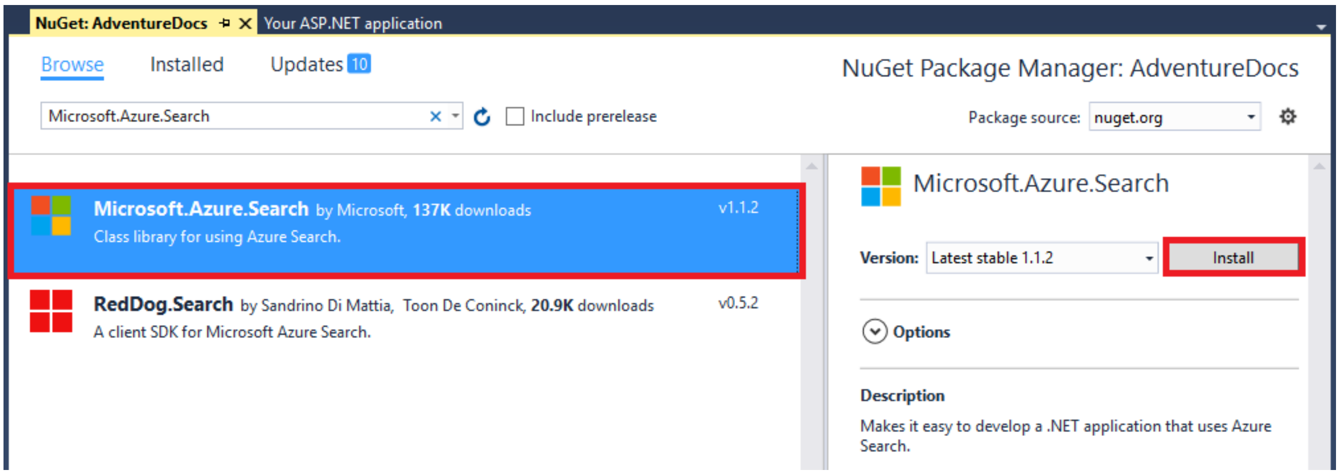

NuGet: AdventureDocs
Your ASP.NET application
Browse
Installed
Updates 10
NuGet Package Manager: AdventureDocs
Microsoft.WindowsAzure.ConfigurationManager
Include prerelease
Package source: nuget.org
Microsoft.WindowsAzure.ConfigurationManager by Microsoft, 5.85M c v3.2.1
Microsoft.WindowsAzure.Configur
Version: Latest stable 3.2.1
Install
Options
Description

Microsoft Azure Configuration Manager provides a unified API to load configuration settings regardless of where the application is hosted - whether on-premises or in a Cl...

Microsoft Azure Configuration Manager provides a unified API to load configuration settings regardless of where the application is hosted - whether on-premises or in a Cloud Service.

Installing Microsoft.WindowsAzure.ConfigurationManager

9. Repeat this process to add the NuGet package named **Microsoft.Azure.Search** to the project. This package contains APIs for accessing Azure Search from .NET applications. Once more, OK any changes and accept any licenses presented to you.


NuGet: AdventureDocs
Your ASP.NET application
Browse
Installed
Updates 10
NuGet Package Manager: AdventureDocs
Microsoft.Azure.Search
Include prerelease
Package source: nuget.org
Microsoft.Azure.Search by Microsoft, 137K downloads v1.1.2
RedDog.Search by Sandrino Di Mattia, Toon De Coninck, 20.9K downloads v0.5.2
Microsoft.Azure.Search
Version: Latest stable 1.1.2
Install
Options
Description

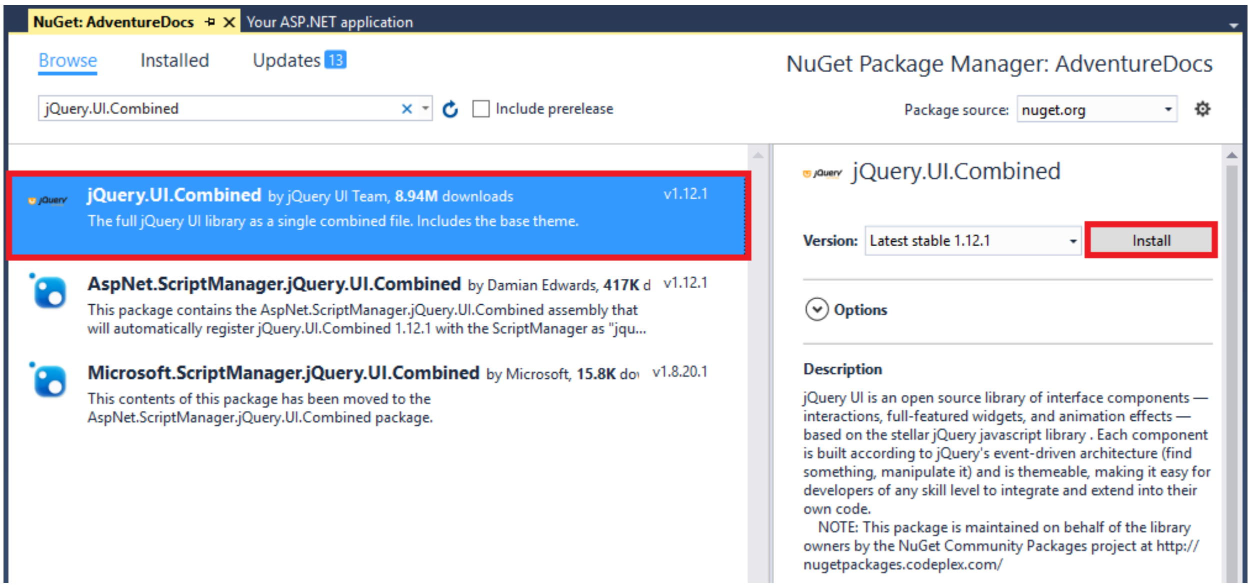
Class library for using Azure Search.

A client SDK for Microsoft Azure Search.

Makes it easy to develop a .NET application that uses Azure Search.

Installing Microsoft.Azure.Search

10. Repeat this process to add the NuGet package named **jQuery.UI.Combined** to the project. This package contains APIs and file elements required by MVC 5 for jQuery user interface elements. Once more, OK any changes and accept any licenses presented to you.



NuGet Package Manager: AdventureDocs

jQuery.UI.Combined

jQuery.UI.Combined by jQuery UI Team, **8.94M** downloads v1.12.1
The full jQuery UI library as a single combined file. Includes the base theme.

AspNet.ScriptManager.jQuery.UI.Combined by Damian Edwards, **417K** d v1.12.1
This package contains the AspNet.ScriptManager.jQuery.UI.Combined assembly that will automatically register jQuery.UI.Combined 1.12.1 with the ScriptManager as "jqu...

Microsoft.ScriptManager.jQuery.UI.Combined by Microsoft, **15.8K** do v1.8.20.1
This contents of this package has been moved to the AspNet.ScriptManager.jQuery.UI.Combined package.

jQuery.UI.Combined

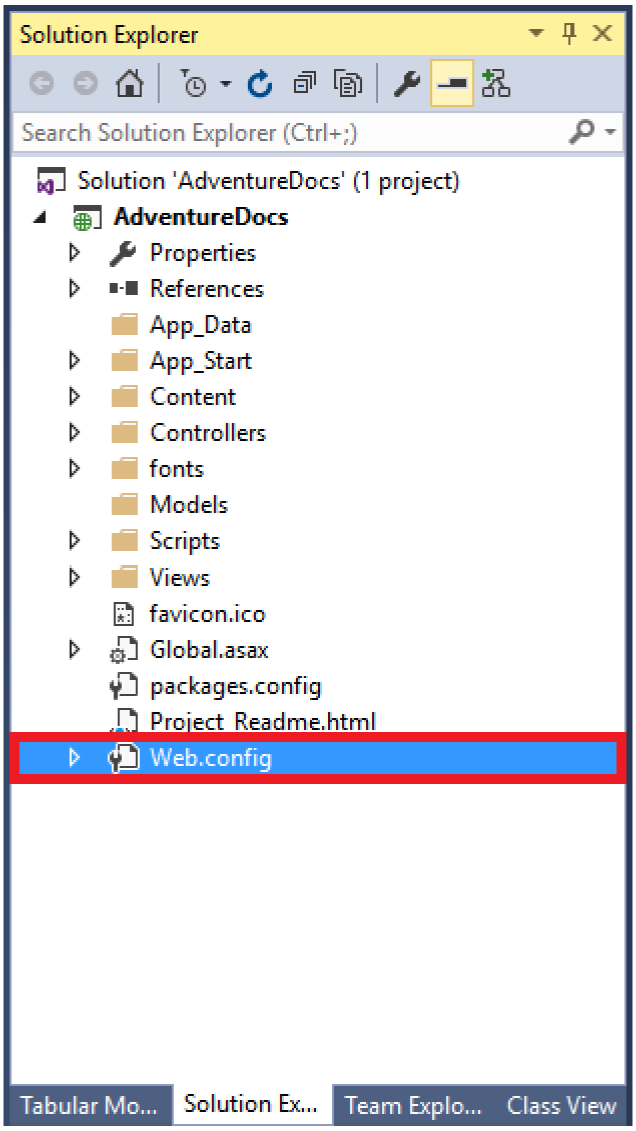
Version: Latest stable 1.12.1

Description

jQuery UI is an open source library of interface components — interactions, full-featured widgets, and animation effects — based on the stellar jQuery javascript library . Each component is built according to jQuery's event-driven architecture (find something, manipulate it) and is themeable, making it easy for developers of any skill level to integrate and extend into their own code.
NOTE: This package is maintained on behalf of the library owners by the NuGet Community Packages project at http://nugetpackages.codeplex.com/

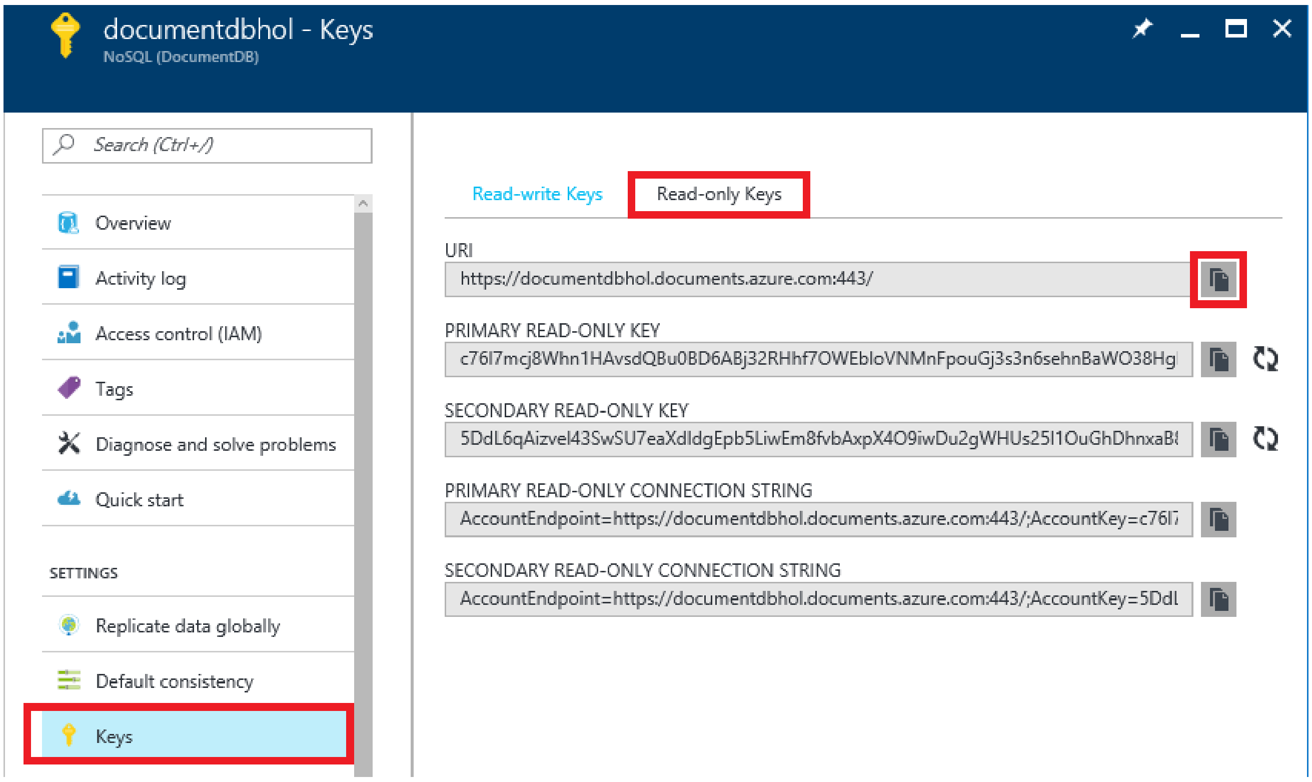
Installing jQuery.UI.Combined

11. In the Solution Explorer window, double-click **Web.config** to open it for editing.



Opening Web.config

12. Return to the Azure Portal and open the blade for the DocumentDB account that you created in Exercise 1. Click **Keys**. Then click **Read-only keys**, and click the **Copy** button to the right of the **URI** box to copy the DocumentDB URI to the clipboard.



URI
https://documentdbhol.documents.azure.com:443/

PRIMARY READ-ONLY KEY
c76I7mcj8Whn1HAvsdQBu0BD6ABj32RHhf7OWEbloVNMnFpouGj3s3n6sehnBaWO38Hg

SECONDARY READ-ONLY KEY
5DdL6qAizvel43SwSU7eaXdldgEpb5LiwEm8fvbAxpX4O9iwDu2gWHUs25I1OuGhDhnxaB

PRIMARY READ-ONLY CONNECTION STRING
AccountEndpoint=https://documentdbhol.documents.azure.com:443/;AccountKey=c76I7

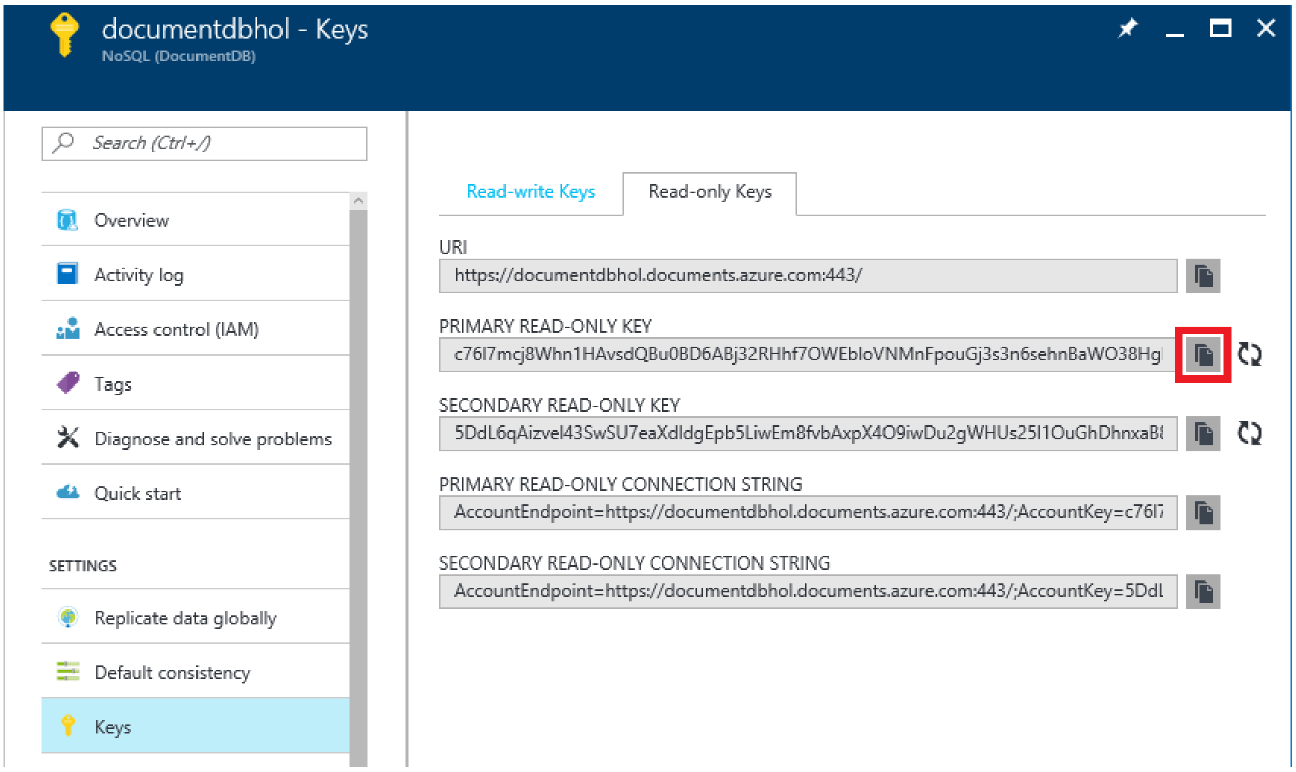
SECONDARY READ-ONLY CONNECTION STRING
AccountEndpoint=https://documentdbhol.documents.azure.com:443/;AccountKey=5DdL

Copying the DocumentDB URI

13. Return to Visual Studio. In **Web.config**, add the following statement to the <appSettings> section, replacing *documentdb_endpoint* with the URI on the clipboard:

```
<add key="DocumentDBEndpointUrl" value="documentdb_endpoint" />
```

14. Return to the Azure Portal and click the **Copy** button to the right of **PRIMARY READ-ONLY KEY** to copy the access key the clipboard.



URI
https://documentdbhol.documents.azure.com:443/

PRIMARY READ-ONLY KEY
c76l7mcj8Whn1HAvsdQBu0BD6ABj32RHhf7OWEbloVNMnFpouGj3s3n6sehnBaWO38Hg

SECONDARY READ-ONLY KEY
5DdL6qAizvel43SwSU7eaXdldgEpb5LiwEm8fvbAxpX4O9iwDu2gWHUs25l1OuGhDhnxaB

PRIMARY READ-ONLY CONNECTION STRING
AccountEndpoint=https://documentdbhol.documents.azure.com:443/;AccountKey=c76l7

SECONDARY READ-ONLY CONNECTION STRING
AccountEndpoint=https://documentdbhol.documents.azure.com:443/;AccountKey=5DdL

Copying the DocumentDB access key

15. Return to Visual Studio. In **Web.config**, add the following statement to the <appSettings> section, replacing *documentdb_key* with the access key on the clipboard:

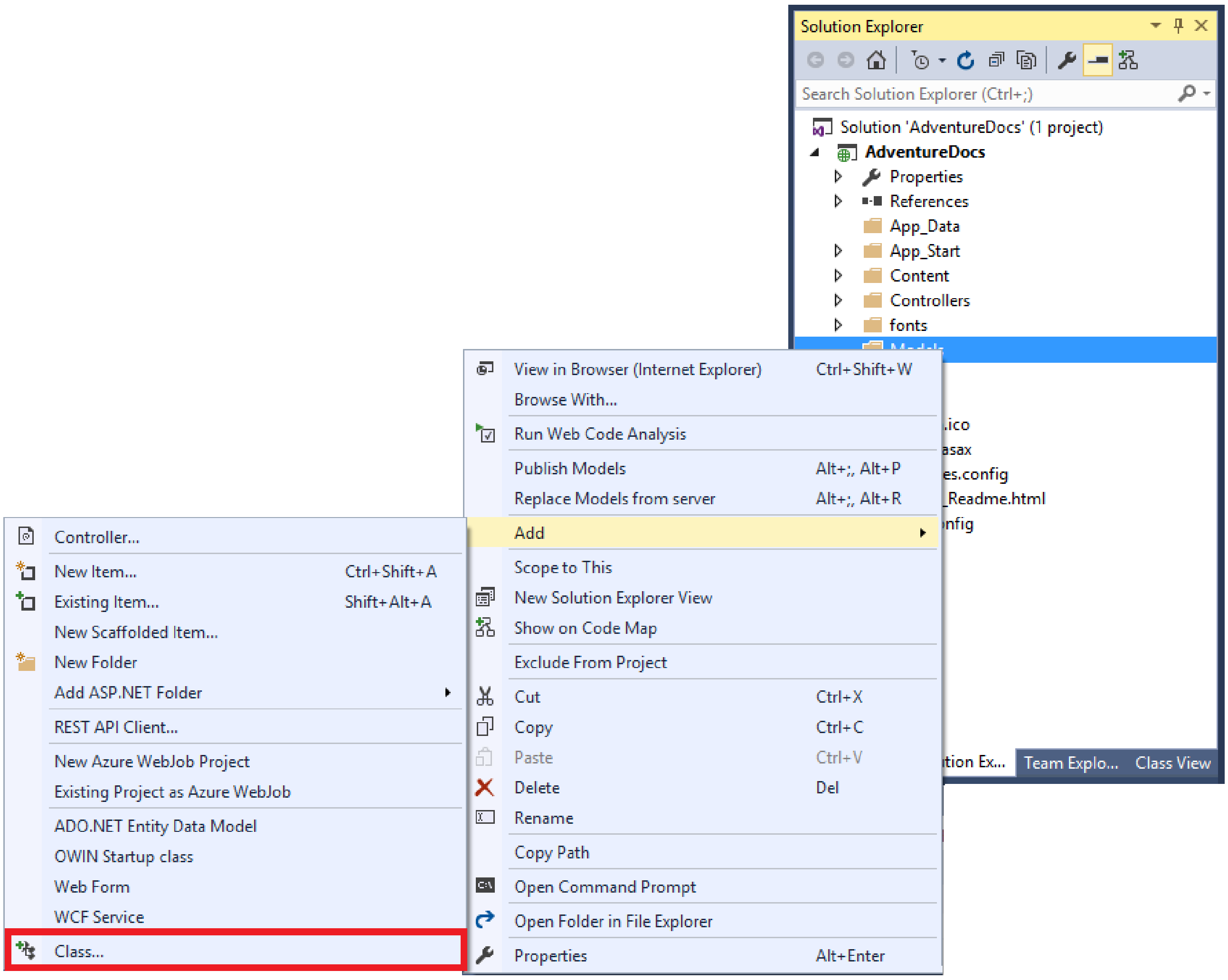
```
<add key="DocumentDBKey" value="documentdb_key" />
```

16. In the Solution Explorer window, find the file named **_Layout.cshtml** in the "Views/Shared" folder. Double-click it to open it. Then replace its contents with the following statements:

```
<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
    @Styles.Render("~/Content/css")
    @Styles.Render("~/Content/themes/base/css")
    @Scripts.Render("~/bundles/modernizr")
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @Scripts.Render("~/bundles/jqueryui")
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
```

```
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse"
data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("AdventureDB", "Index", "Home", new { area = "" },
new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Document Search", "Index", "Home")</li>
                    <li>@Html.ActionLink("Customer Lookup", "Lookup", "Home")</li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p class="text-muted">All rights reserved. Copyright
&copy;@DateTime.Now.Year AdventureDB.</p>
        </footer>
    </div>

    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>
```

17. In Solution Explorer, right-click the "Models" folder and select **Add -> Class...**. Then type "OrderInformation.cs" (without quotation marks) into the **Name** box and click **OK** to add the class to project.



Adding a class to the "Models" folder

18. Replace the empty *OrderInformation* class with the following class definitions, and note that you are making the classes public rather than private, as well as marking the *OrderInformation* class "Serializable:"

```
[Serializable]

public class OrderInformation
{
    public string CustomerID { get; set; }
    public string CompanyName { get; set; }
    public Customer Customer { get; set; }
}

public class Customer
{
    public string CustomerID { get; set; }
    public string CompanyName { get; set; }
    public string ContactName { get; set; }
    public string ContactTitle { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public object Region { get; set; }
    public string PostalCode { get; set; }
    public int Latitude { get; set; }
    public int Longitude { get; set; }
    public string Country { get; set; }
    public string Phone { get; set; }
    public string Fax { get; set; }
    public Orders Orders { get; set; }
}

public class Orders
{
    public int OrderID { get; set; }
    public string CustomerID { get; set; }
    public int EmployeeID { get; set; }
    public DateTime OrderDate { get; set; }
    public DateTime RequiredDate { get; set; }
    public DateTime ShippedDate { get; set; }
    public int ShipVia { get; set; }
    public float Freight { get; set; }
    public string ShipName { get; set; }
    public string ShipAddress { get; set; }
    public string ShipCity { get; set; }
    public object ShipRegion { get; set; }
    public string ShipPostalCode { get; set; }
    public string ShipCountry { get; set; }
    public Details Details { get; set; }
}

public class Details
{
    public int OrderID { get; set; }
    public int ProductID { get; set; }
    public int Quantity { get; set; }
    public float Discount { get; set; }
    public Product Product { get; set; }
}

public class Product
{
    public int ProductID { get; set; }
    public string ProductName { get; set; }
    public int SupplierID { get; set; }
    public int CategoryID { get; set; }
    public string QuantityPerUnit { get; set; }
    public int UnitPrice { get; set; }
    public int UnitsInStock { get; set; }
    public int UnitsOnOrder { get; set; }
    public int ReorderLevel { get; set; }
    public bool Discontinued { get; set; }
}
```

19. Right-click the "Models" folder again and use the **Add -> Class...** command to add a file named **SearchResultInformation.cs** to the folder. Replace the empty *SearchResultInformation* class with the following class definition:

```
public class SearchResultInformation
{
    public string Title { get; set; }
    public string Description { get; set; }
    public string DocumentContent { get; set; }
}
```

20. Repeat this process to add an *OrderViewModel* class to the "Models" folder, and replace the empty class with the following class definition:

```
public class OrderViewModel
{
    public string SearchQuery { get; set; }
    public List<SearchResultInformation> SearchResults { get; set; }
    public List<string> Collections { get; set; }
    public string SelectedCollectionName { get; set; }
    public string SearchResultTitle { get; set; }
    public string SearchResultDescription { get; set; }
}
```

21. In the Solution Explorer window, right-click the **AdventureDocs** project and use the **Add -> New Folder** command to add a folder named "Helpers" to the project.
22. Right-click the "Helpers" folder and use the **Add -> Class...** command to add a file named **DocumentHelper.cs** to the folder. Replace the contents of the file with the following statements:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Azure.Documents;
using Microsoft.Azure.Documents.Client;
using Newtonsoft.Json;
using System.Threading.Tasks;
using AdventureDocs.Models;

namespace AdventureDocs.Helpers
{
    public class DocumentHelper
    {
        public static DocumentClient GetDocumentClient()
        {
            string                          endpointUrl                          =
Microsoft.Azure.CloudConfigurationManager.GetSetting("DocumentDBEndpointUrl");
            string                          primaryKey                           =
Microsoft.Azure.CloudConfigurationManager.GetSetting("DocumentDBKey");

            DocumentClient  client  =  new  DocumentClient(new  Uri(endpointUrl),
primaryKey);

            return client;
        }

        public             static             async             Task<List<string>>
GetAvailableCollectionNamesAsync(DocumentClient client)
        {
            List<string> collections = new List<string>();

            try
            {
                var dbFeed = await client.ReadDatabaseFeedAsync();
                var defaultDb = dbFeed.FirstOrDefault();

                if (defaultDb != null)
                {
                    FeedResponse<DocumentCollection>     collFeed     =     await
client.ReadDocumentCollectionFeedAsync(defaultDb.SelfLink);

                    collections = (from feed in collFeed select feed.Id).ToList();
                }
            }
            catch (Exception ex)
            {

            }

            return collections;

        }
public static List<SearchResultInformation> GetOrdersByOrder(DocumentClient client,
string countryName)
        {
            FeedOptions queryOptions = new FeedOptions { MaxItemCount = -1 };

            IQueryable<OrderInformation>               orderQuery                =
client.CreateDocumentQuery<OrderInformation>(
                    UriFactory.CreateDocumentCollectionUri("CustomerOrders",
"Orders"),                   queryOptions)Where(f                              =>
f.Customer.Orders.ShipCountry.ToLower().StartsWith(countryName.ToLower()));
```

```
            var orderItems = orderQuery.ToList();

            var results = (from item in orderItems
                           select new SearchResultInformation()
                           {
                               Title = item.Customer.CompanyName,
                               Description = item.Customer.Orders.ShipCountry,
                               DocumentContent =
JsonConvert.SerializeObject(item),

                           }).ToList();


            return   results.Select(r   =>   r.Title).Distinct().Select(title   =>
results.First(r => r.Title == title)).ToList();
        }

        public                 static                 List<SearchResultInformation>
GetOrdersByCustomer(DocumentClient client, string companyName)
        {
            FeedOptions queryOptions = new FeedOptions { MaxItemCount = -1 };

            IQueryable<OrderInformation>               orderQuery                 =
client.CreateDocumentQuery<OrderInformation>(
                    UriFactory.CreateDocumentCollectionUri("CustomerOrders",
"Orders"), queryOptions)
                     .Where(f                                                    =>
f.Customer.CompanyName.ToLower().StartsWith(companyName.ToLower()));


            var orderItems = orderQuery.ToList();

            List<SearchResultInformation> results = (from item in orderItems
                                                    select                     new
SearchResultInformation()
 {
      Title = item.Customer.CompanyName,
      Description = item.Customer.Country,
      DocumentContent = JsonConvert.SerializeObject(item),
  }).ToList();
return results.Select(r => r.Title).Distinct().Select(title => results.First(r =>
r.Title == title)).ToList();
 }

        public                 static                 List<SearchResultInformation>
GetOrdersByProduct(DocumentClient client, string productName)
        {
            FeedOptions queryOptions = new FeedOptions { MaxItemCount = -1 };
            IQueryable<OrderInformation>               orderQuery               =
client.CreateDocumentQuery<OrderInformation>(
                    UriFactory.CreateDocumentCollectionUri("CustomerOrders",
"Orders"), queryOptions)
                    .Where(f                                                   =>
f.Customer.Orders.Details.Product.ProductName.ToLower().StartsWith(productName.ToL
ower()));

            var orderItems = orderQuery.ToList();

            var results = (from item in orderItems
                           select new SearchResultInformation()
                           {
                               Title                                           =
item.Customer.Orders.Details.Product.ProductName,
                               Description                                     =
item.Customer.Orders.Details.Product.QuantityPerUnit,
                               DocumentContent                                 =
JsonConvert.SerializeObject(item),

                           }).ToList();



            return   results.Select(r   =>   r.Title).Distinct().Select(title   =>
results.First(r => r.Title == title)).ToList();
        }
    }
}
```

23. Open **HomeController.cs** in the project's "Controllers" folder. Add the following using statements to the top of the file:

```
using System.Threading.Tasks;
using AdventureDocs.Models;
```

24. Replace the *Index* method in **HomeController.cs** with the following implementation:

```
public async Task<ActionResult> Index()
{
    var    model    =    new    OrderViewModel()    {    SearchResults    =    new
List<SearchResultInformation>() };

    var documentClient = Helpers.DocumentHelper.GetDocumentClient();


    var                availableCollections                =                await
Helpers.DocumentHelper.GetAvailableCollectionNamesAsync(documentClient);

    var searchResults = (List<SearchResultInformation>)TempData["SearchResults"];
    var searchQuery = (string)Request["SearchQuery"];
    if (searchResults != null)
    {
        model.SearchQuery = (string)TempData["SearchQuery"];
        model.SearchResults                                                      =
(List<SearchResultInformation>)TempData["SearchResults"];

        model.SelectedCollectionName = (string)TempData["SelectedCollectionName"];
        model.SearchResultTitle = $"{model.SelectedCollectionName}";
        model.SearchResultDescription  =  $"The  following  results  were  found  in
{model.SelectedCollectionName} for '{model.SearchQuery.ToUpper()}':";
    }
    else if (!string.IsNullOrEmpty(searchQuery))
    {
        model.SearchQuery = searchQuery;

        searchResults = Helpers.DocumentHelper.GetOrdersByCustomer(documentClient,
searchQuery);

        model.SearchResults = searchResults;

        model.SelectedCollectionName = "Customers";
        model.SearchResultTitle = $"{model.SelectedCollectionName}";
        model.SearchResultDescription  =  $"The  following  results  were  found  in
{model.SelectedCollectionName} for '{model.SearchQuery.ToUpper()}':";
    }
    else
    {
        model.SearchQuery = "";
        model.SelectedCollectionName = "Customers";
        model.SearchResultTitle = "";
        model.SearchResultDescription = "";
    }

    model.Collections = availableCollections;
    return View(model);
}
```

25. Add the following methods to the *HomeController* class in **HomeController.cs**:

```
public ActionResult Lookup()
{
    ViewBag.Message = "Your application description page.";
    return View();
}

[HttpGet]
public ActionResult ViewSource(string[] content)
{
    return new JsonResult
    {
        JsonRequestBehavior = JsonRequestBehavior.AllowGet,
        Data = content[0]
    };
}

[HttpPost]
public ActionResult Search(OrderViewModel model)
{
    ViewBag.Message = "Your application description page.";

    string searchQuery = model.SearchQuery + "";

    var documentClient = Helpers.DocumentHelper.GetDocumentClient();

    List<SearchResultInformation>           searchResults          =           new
List<SearchResultInformation>();

    switch (model.SelectedCollectionName)
    {
        case "Customers":
            searchResults                                                        =
Helpers.DocumentHelper.GetOrdersByCustomer(documentClient, searchQuery);
            break;
        case "Products":
            searchResults                                                        =
Helpers.DocumentHelper.GetOrdersByProduct(documentClient, searchQuery);
            break;
        case "Orders":
            searchResults                                                        =
Helpers.DocumentHelper.GetOrdersByOrder(documentClient, searchQuery);
            break;
        default:
            break;
    }

    TempData["SearchQuery"] = searchQuery;
    TempData["SearchResults"] = searchResults;
    TempData["SelectedCollectionName"] = model.SelectedCollectionName;

    return RedirectToAction("Index");
}
```

26. Open **Index.cshmtl** in the "Views/Home" folder and replace its contents with the following statements:

```
@{
    ViewBag.Title = "AdventureDocs";

}



<div class="row">
    @model AdventureDocs.Models.OrderViewModel
    <div>
        <h2>Document Search</h2>
        <p>
            To search documents in your Azure DocumentDB database, enter a value,
select a DocumentDB collection, and click Search.
        </p>

        @using (Html.BeginForm("Search", "Home", FormMethod.Post))
        {
            <div>Search for:</div>
            @Html.TextBoxFor(o => Model.SearchQuery)
            <p></p>
            <div>Select a collection:</div>
            @Html.DropDownListFor(x      =>      x.SelectedCollectionName,      new
SelectList(Model.Collections))
            <input type="submit" value="Search">
        }

        <div>
            <h4>@Html.DisplayFor(o => Model.SearchResultTitle)</h4>
            <div>@Html.DisplayFor(o => Model.SearchResultDescription)</div>
            <table style="margin:10px" border="0" cellpadding="3">
                @foreach (var item in Model.SearchResults)
                {
                    <tr>
                        <td>
                            <strong>@Html.DisplayFor(modelItem                  =>
item.Title)</strong>
                        </td>
                        <td>
                            <em>@Html.DisplayFor(modelItem                      =>
item.Description)</em>
                        </td>

                        <td>
                            @Html.ActionLink(
                            linkText: "[view document]",
                            actionName: "ViewSource",
                            controllerName: "Home",
                            routeValues: new { content = item.DocumentContent },
                            htmlAttributes: null)
                        </td>
                    </tr>
                }
            </table>
        </div>
    </div>
</div>
```

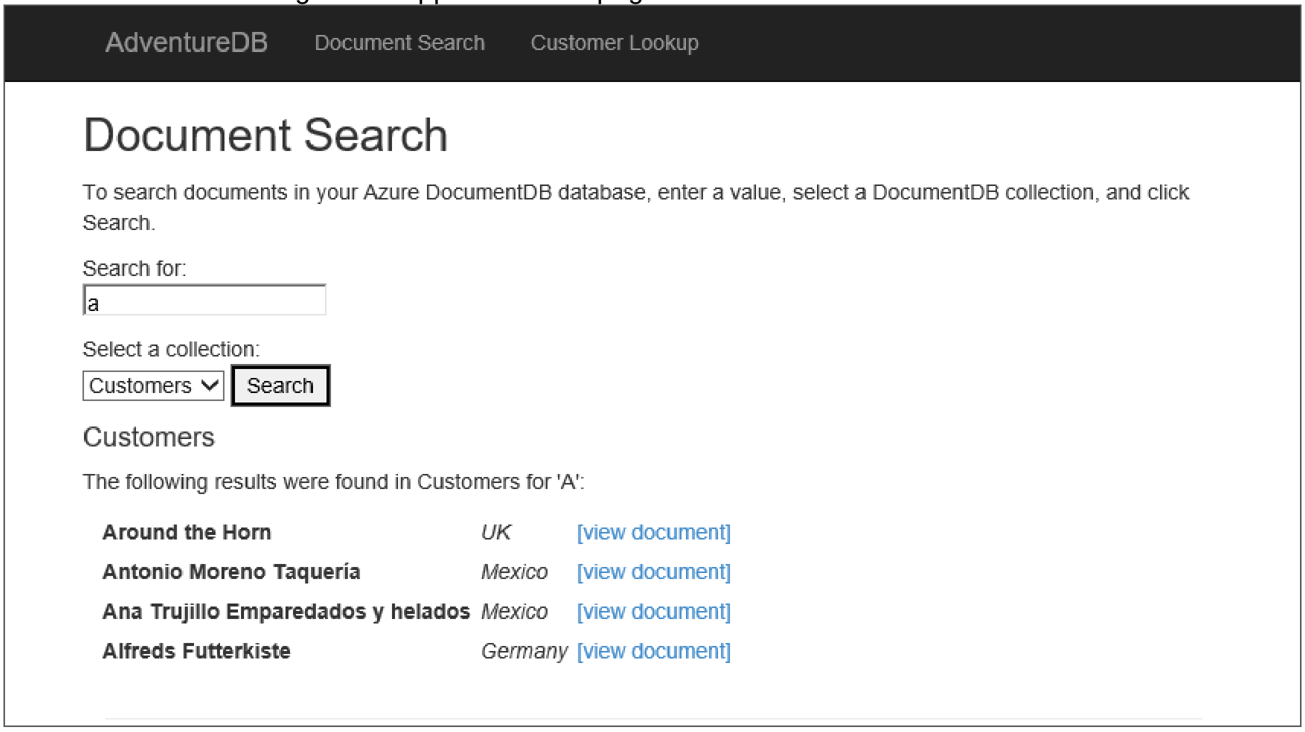
27. Find **About.cshmtl** in the "Views/Home" folder. Right-click the file and use the **Rename** command to change its name to **Lookup.cshtml**. This is the view that will serve as the document lookup page.
28. Replace the contents of **Lookup.cshtml** with the following statements:

```
@Scripts.Render("~/bundles/jqueryui")



<script type="text/javascript">
    $(document).ready(function () {

        $('#customers').autocomplete({
            source: '@Url.Action("Suggest")',
            autoFocus: true,
            select: function (event, ui) {

                if (ui.item) {
                    $("#SearchQuery").val(ui.item.value);
                    $("form").submit();
                }
            }
        });
    })
</script>
<div class="row">
    <div class="col-md-4">
        <h2>Customer Lookup</h2>
        <p>
            To search documents in your Azure DocumentDB database, enter a value
and select an autosuggested customer.
        </p>
        <div>Search for:</div>
        <input id="customers" name="customers">
        <form action="/" method="post">
            <input     hidden="hidden"     id="SearchQuery"     name="SearchQuery"
type="text"/>
        </form>
    </div>
    <div style="height:400px" class="col-md-4"></div>
</div>
```

29. Open **BundleConfig.cs** in the project's "App_Start" folder. Add the following code at the end of the **RegisterBundles** method:

```
bundles.Add(new   ScriptBundle("~/bundles/jqueryui").Include("~/Scripts/jquery-ui-
{version}.js"));
bundles.Add(new StyleBundle("~/Content/themes/base/css").Include(
            "~/Content/themes/base/jquery.ui.core.css",
            "~/Content/themes/base/jquery.ui.autocomplete.css",
"~/Content/themes/base/jquery.ui.theme.css"));
```

30. Use Visual Studio's **Debug -> Start Without Debugging** command (or press **Ctrl+F5**) to launch the application in your browser.
31. **Type** the letter "a" in the **Search for** box and click the **Search** button. Confirm that a list of customer names starting with A appears on the page:



Document Search

To search documents in your Azure DocumentDB database, enter a value, select a DocumentDB collection, and click Search.

Search for: a

Select a collection: Customers

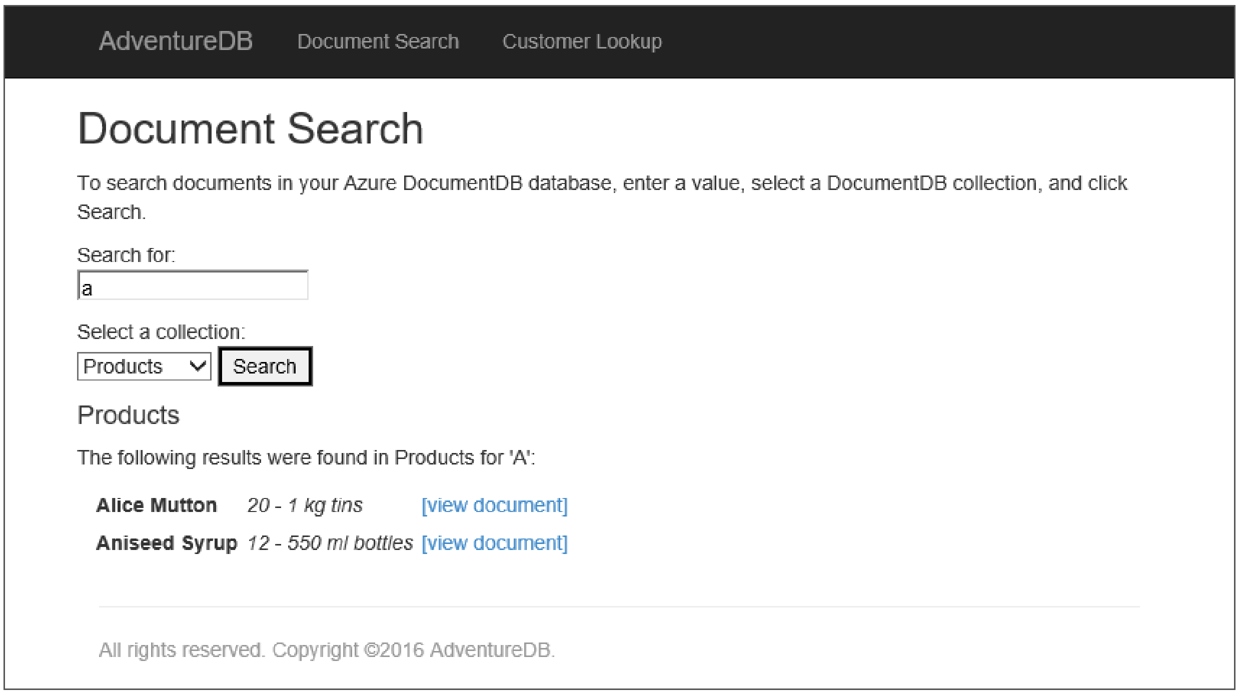
Customers

The following results were found in Customers for 'A':

Around the Horn	*UK*	[view document]
Antonio Moreno Taquería	*Mexico*	[view document]
Ana Trujillo Emparedados y helados	*Mexico*	[view document]
Alfreds Futterkiste	*Germany*	[view document]

Searching for customer names that begin with A

32. Select **Products** from the list of collections and click the **Search** button. Confirm that a list of product names starting with A appears on the page:



AdventureDB Document Search Customer Lookup

Document Search

To search documents in your Azure DocumentDB database, enter a value, select a DocumentDB collection, and click Search.

Search for: a

Select a collection: Products

Search

Products

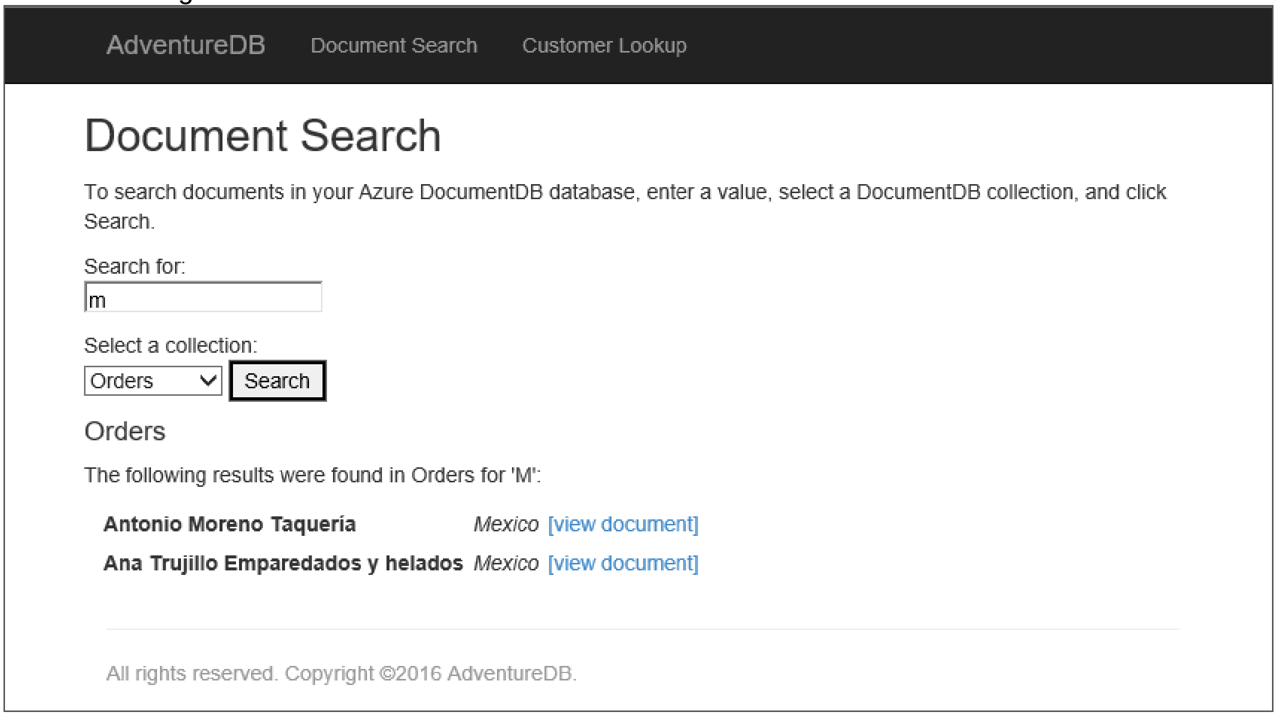
The following results were found in Products for 'A':

Alice Mutton	*20 - 1 kg tins*	[view document]
Aniseed Syrup	*12 - 550 ml bottles*	[view document]

All rights reserved. Copyright ©2016 AdventureDB.

Searching for product names that begin with A

33. Replace the letter "a" in the **Search for** box with the letter "m." Then select **Orders** and click the **Search** button. Confirm that a list of orders appears, with the countries they were shipped to listed on the right:



AdventureDB Document Search Customer Lookup

Document Search

To search documents in your Azure DocumentDB database, enter a value, select a DocumentDB collection, and click Search.

Search for: m

Select a collection: Orders

Search

Orders

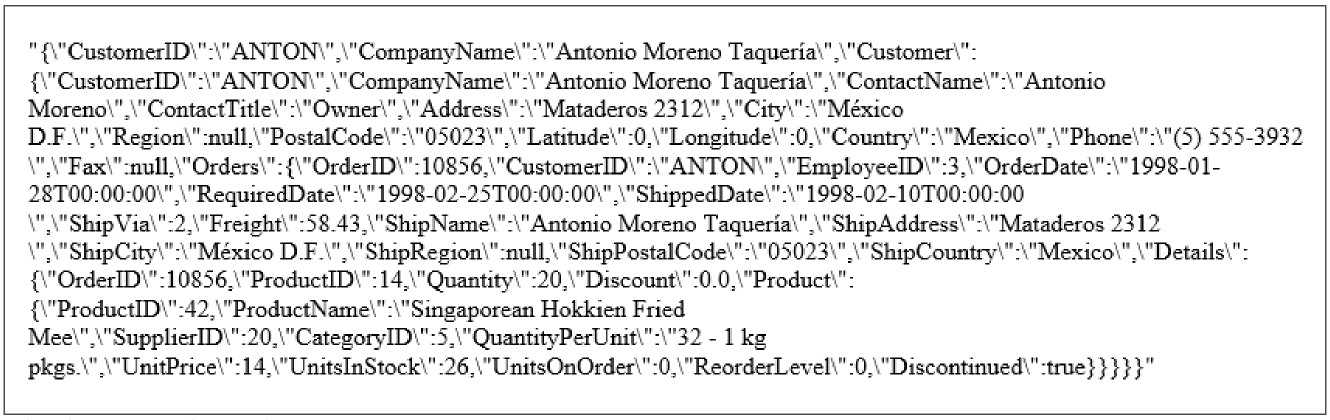
The following results were found in Orders for 'M':

Antonio Moreno Taquería	*Mexico*	[view document]
Ana Trujillo Emparedados y helados	*Mexico*	[view document]

All rights reserved. Copyright ©2016 AdventureDB.

Searching for orders shipped to countries that begin with M

34. The Orders listing displays the CompanyName and ShipRegion values from the orders returned in the search results. To view the entire order, click **[view document]** to the right of an order. The result is the JSON defining the order:



```
"{\"CustomerID\":\"ANTON\",\"CompanyName\":\"Antonio Moreno Taquería\",\"Customer\":
{\"CustomerID\":\"ANTON\",\"CompanyName\":\"Antonio Moreno Taquería\",\"ContactName\":\"Antonio
Moreno\",\"ContactTitle\":\"Owner\",\"Address\":\"Mataderos 2312\",\"City\":\"México
D.F.\",\"Region\":null,\"PostalCode\":\"05023\",\"Latitude\":0,\"Longitude\":0,\"Country\":\"Mexico\",\"Phone\":\"(5) 555-3932
\",\"Fax\":null,\"Orders\":{\"OrderID\":10856,\"CustomerID\":\"ANTON\",\"EmployeeID\":3,\"OrderDate\":\"1998-01-
28T00:00:00\",\"RequiredDate\":\"1998-02-25T00:00:00\",\"ShippedDate\":\"1998-02-10T00:00:00
\",\"ShipVia\":2,\"Freight\":58.43,\"ShipName\":\"Antonio Moreno Taquería\",\"ShipAddress\":\"Mataderos 2312
\",\"ShipCity\":\"México D.F.\",\"ShipRegion\":null,\"ShipPostalCode\":\"05023\",\"ShipCountry\":\"Mexico\",\"Details\":
{\"OrderID\":10856,\"ProductID\":14,\"Quantity\":20,\"Discount\":0.0,\"Product\":
{\"ProductID\":42,\"ProductName\":\"Singaporean Hokkien Fried
Mee\",\"SupplierID\":20,\"CategoryID\":5,\"QuantityPerUnit\":\"32 - 1 kg
pkgs.\",\"UnitPrice\":14,\"UnitsInStock\":26,\"UnitsOnOrder\":0,\"ReorderLevel\":0,\"Discontinued\":true}}}}}"
```

Viewing an entire order

This is a great start, and it demonstrates how an ASP.NET MVC Web app can access data stored in the cloud in a DocumentDB database. But right now, the search UI is somewhat clumsy; you have to enter letters blindly, without any feedback to tell you whether there are any matching documents. Let's enhance the UI by adding auto-suggest.

Exercise 6: Add auto-suggest

Azure Search enables super-fast retrieval of indexed values in the data stores is is connected to. In this exercise, you will leverage that speed to add an auto-suggest list to customer search to provide feedback to the user as he or she types.

1. Return to Visual Studio. Right-click the "Helpers" folder and use the **Add -> Class...** command to add a file named **SearchHelper.cs** to the folder. Replace the contents of the file with the following statements:

```
using Microsoft.Azure.Search;
using Microsoft.Azure.Search.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

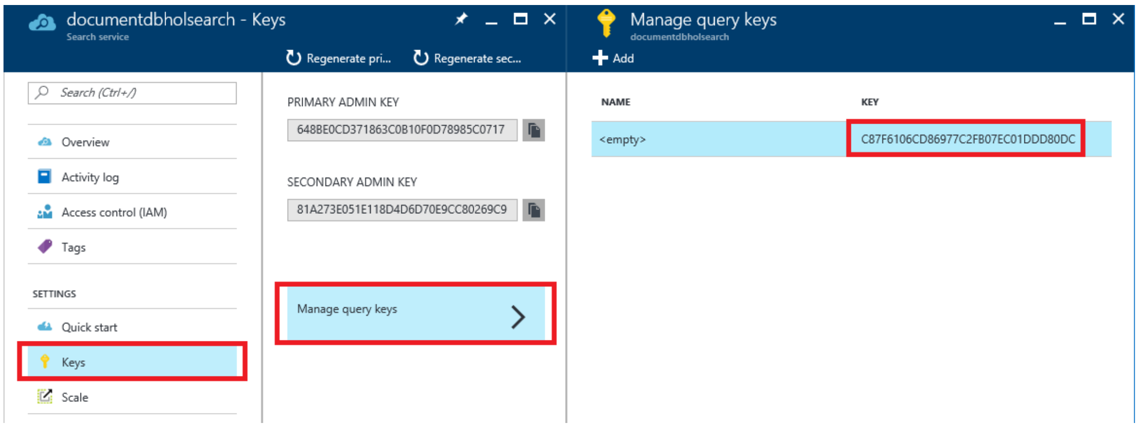
```
using System.Web;
using AdventureDocs.Models;
namespace AdventureDocs.Helpers
{
    public static class SearchHelper
    {
        public static List<string> GetSuggestions(string query)
        {
            List<string> suggestions = new List<string>();

            string                                               searchServiceName
Microsoft.Azure.CloudConfigurationManager.GetSetting("SearchServiceName");
            string                        searchServiceKey                        =
Microsoft.Azure.CloudConfigurationManager.GetSetting("SearchServiceKey");
```

```
            SearchServiceClient           serviceClient           =           new
SearchServiceClient(searchServiceName, new SearchCredentials(searchServiceKey));

            ISearchIndexClient                    indexClient                    =
serviceClient.Indexes.GetClient("customerindex");

            DocumentSearchResult<Customer>                response                =
indexClient.Documents.Search<Customer>($"{query.Trim()}*");

            suggestions = (from result in response.Results
                           select result.Document.CompanyName).ToList();

            return suggestions;
        }
    }
}
```

2. Open **Web.config** and add the following statement to the <appSettings> section, replacing *search_service_name* with the name you assigned to the Azure Search service in Exercise 4, Step 2:

```
<add key="SearchServiceName" value="search_service_name" />
```

3. Return to the Azure Portal and open the blade for the Azure Search service. Click **Keys**, followed by **Manage query keys**. Then copy the query key to the clipboard.
The purpose of query keys is to allow applications to query the Search service and to do so securely.



PRIMARY ADMIN KEY
648BE0CD371863C0B10F0D78985C0717

SECONDARY ADMIN KEY
81A273E051E118D4D6D70E9CC80269C9

NAME	KEY
<empty>	C87F6106CD86977C2FB07EC01DDD80DC

Copying the query key to the clipboard

4. Return to Visual Studio. Add the following statement to the <appSettings> section of **Web.config**, replacing *search_service_key* with the query key on the clipboard.

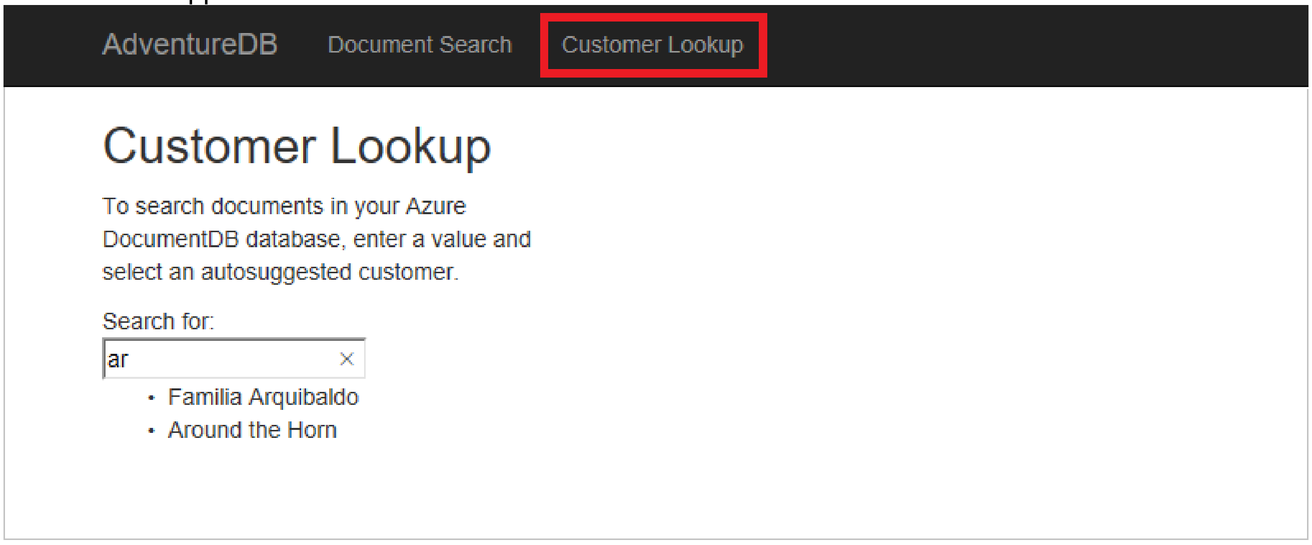
```
<add key="SearchServiceKey" value="search_service_key" />
```

5. Add the following methods to the *HomeController* class in **HomeController.cs**:

```
[HttpPost]

public ActionResult AutoSearch(string item)
{
    ViewBag.Message = "Your application description page.";
```

```
        string searchQuery = item + "";

        TempData["SearchQuery"] = searchQuery;
        TempData["SelectedCollectionName"] = "Customers";

        return RedirectToAction("Index");
    }

    [HttpGet]
    public ActionResult Suggest(string term)
    {
        List<string> suggestions = new List<string>();

        suggestions = Helpers.SearchHelper.GetSuggestions(term);

        return new JsonResult
        {
            JsonRequestBehavior = JsonRequestBehavior.AllowGet,
            Data = suggestions
        };
    }
```

6. Use Visual Studio's **Debug -> Start Without Debugging** command (or press **Ctrl+F5**) to launch the application in your browser.
7. Click **Customer Lookup**. Then type "ar" into the **Search for** box and wait for a list of suggested customers to appear.



Customer Lookup

To search documents in your Azure DocumentDB database, enter a value and select an autosuggested customer.

Search for:

ar

- Familia Arquibaldo
- Around the Horn

Auto-suggest in action

8. Select **Around the Horn** and press **Enter** to search for customers named "Around the Horn."

Auto-suggest vastly improves the search experience and is relatively easy to add thanks to some of the classes you imported in NuGet packages in Exercise 5.

When you created the project for the Web app in Visual Studio, you checked the **Host in the cloud** box so the app could be deployed to Azure, and you created an Azure App Service to host it. Up to now, the app has run locally. If you would like to deploy it to Azure so it can be opened from anywhere, simply right-click the project in Solution Explorer, select **Publish** from the context menu, and click the **Publish** button in the ensuing dialog. Once the app has been published, it will open in a browser window.

When you're finished using the app, it is recommended that you delete the resource group containing it. Since you placed the Azure Web App in the same resource group as the DocumentDB account and the Search service, deleting the resource group deletes **all** of these resources, removes all traces of this lab from your account, and prevents any further charges from being incurred for it. To delete the resource group, simply open the resource-group blade in the portal and click **Delete** at the top of the blade. You will be asked to type the resource group's name to confirm that you want to delete it, because once deleted, a resource group can't be recovered.

Summary

In this hands-on lab you learned how to:

- Create an Azure DocumentDB account
- Create DocumentDB collections and populate them with documents
- Create an Azure Search service and and use it to index DocumentDB data
- Access Azure DocumentDB collections from your apps
- Query the Azure Search service connected to a DocumentDB database

Not surprisingly, there is much more you can do to leverage the power of Azure DocumentDB. Experiment with other DocumentDB features, especially triggers, stored procedures, and user-defined functions, and identify other ways you can enhance your data and search strategies by integrating Azure DocumentDB into your application ecosystems.

Reach me with the below details:

Abdul Rasheed Feroz Khan,
feroz@cwtechnologies.in
@TechFero