McKesson Azure Lab 08: Demo2

Joan Imrich

>>>

Demo2: Azure Queue Storage Services with Visual Studio (C# code) November 30, 2017

>>>

Azure queue storage implemented via Visual Studio Connected Services (ASP.NET)

Overview

Azure queue storage provides cloud messaging between application components. In designing applications for scale, application components are often decoupled, so that they can scale independently. Queue storage delivers **asynchronous messaging** for communication between application components, whether they are running in the cloud, on the desktop, on a mobile device, or on-prem server. Queue storage supports managing asynchronous tasks and building process work flows.

This demo shows how to write ASP.NET code for basic scenarios & common tasks using Azure queue storage entities:

- Create /Delete an Azure queue
- Add, modify, read, and remove messages

Prerequisites

- Microsoft Visual Studio
- Azure storage account

What is Queue Storage?

Azure Queue storage is a service for storing large numbers of messages that can be accessed anywhere via authenticated calls using HTTP or HTTPS. A single queue message can be up to 64 KB in size, and a queue can contain millions of messages, up to the total capacity limit of a storage account.
Common uses of Queue storage include:

- Creating a backlog of work to process asynchronously
- Passing messages from an Azure web role to an Azure worker role

Queue Service Concepts

The Queue service contains the following components:

- **URL format:** Queues are addressable using the following URL format:
  http://<storage account>.queue.core.windows.net/<queue>

The following URL addresses a queue in the diagram: http://myaccount.queue.core.windows.net/images-to-download

- **Storage Account:** All access to Azure Storage is done through a storage account. See Azure Storage Scalability and Performance Targets for details about storage account capacity.
- **Queue:** A queue contains a set of messages. All messages must be in a queue. Note that the queue name must be all lowercase. For information on naming queues, see Naming Queues and Metadata.
- **Message:** A message, in any format, of up to 64 KB. The maximum time that a message can remain in the queue is 7 days.

Create an Azure storage account

The easiest way to create your first Azure storage account is by using the Azure Portal. To learn more, see Create a storage account. You can also create an Azure storage account by using Azure PowerShell, Azure CLI, or the Storage Resource Provider Client Library for .NET.

Obtain Azure storage account management credentials

https://portal.azure.com/

Go to All Resources >> click on Storage Account >> Access Keys … under Settings blade, click the copy button to get the **primary key and connection string** to your clipboard for later use. Paste this value into Notepad or some other temporary location for later use.

---

lab8stor = Storage account Namespace needed for Demo 2
mcarm = resource manager

Primary Key
Secondary Key
Primary Connection String
Secondary Connection String

Primary Key
CsMidVX0x9lv5y+n/dY6m8x8IXyjklwyr2BqiawH+x0X+RZWxCJjbZHarNj+rcdpR4+mitj4IJv/+2/oDipEJQ==

Primary Connection String
DefaultEndpointsProtocol=https;AccountName=lab8stor;AccountKey=CsMidVX0x9lv5y+n/dY6m8x8IXyjklwyr2BqiawH+x0X+RZWxCJjbZHarNj+rcdpR4+mitj4IJv/+2/oDipEJQ==;EndpointSuffix=core.windows.net

Key2
7WwbLtCoiLUg9voSL4+AljvldaKUrtm/HtGshJpAVxoC+QAz3egnSndByjD1BVt1lOSUORxSCuNTW7+OU2qUUQ==
Secondary Connection String
DefaultEndpointsProtocol=https;AccountName=lab8stor;AccountKey=7WwbLtCoiLUg9voSL4+AljvldaKUrtm/HtGshJpAVxoC+QAz3egnSndByjD1BVt1lOSUORxSCuNTW7+OU2qUUQ==;EndpointSuffix=core.windows.net

---

lab8bus = Service Bus Namespace … needed for Demo 3
mckazurerg = resource manager

Primary Key
doNsICQ9tS9NP5+hiCwWwWoqTWboCjB0Ef6P3Cjuiqk=

Primary Connection String
Endpoint=sb://lab8bus.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=doNsICQ9tS9NP5+hiCwWwWoqTWboCjB0Ef6P3Cjuiqk=

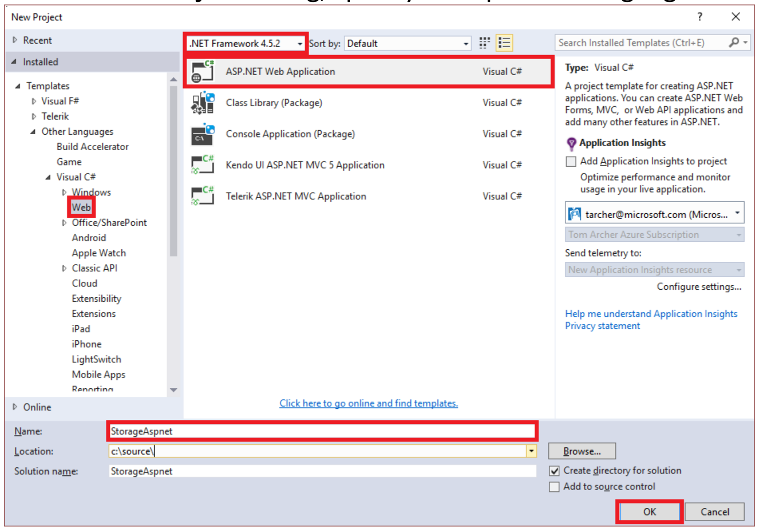
Secondary Key
MmQM8ZU+OyZ6hIzd99Z2rPy2fiDYuhmpZeYIGvCc9ro=
Secondary Connection String
Endpoint=sb://lab8bus.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=MmQM8ZU+OyZ6hIzd99Z2rPy2fiDYuhmpZeYIGvCc9ro=

Set up the development environment

This section walks you setting up your development environment, including creating an ASP.NET MVC app, adding a Connected Services connection, adding a controller, and specifying the required namespace directives.

Create an ASP.NET MVC app project

Open Visual Studio.
Select **File->New->Project** from the main menu
On the **New Project** dialog, specify the options as highlighted in the following figure:



New Project

.NET Framework 4.5.2 | Sort by: Default

Installed > Templates > Other Languages > Visual C# > Web

ASP.NET Web Application — Visual C#
Class Library (Package) — Visual C#
Console Application (Package) — Visual C#
Kendo UI ASP.NET MVC 5 Application — Visual C#
Telerik ASP.NET MVC Application — Visual C#

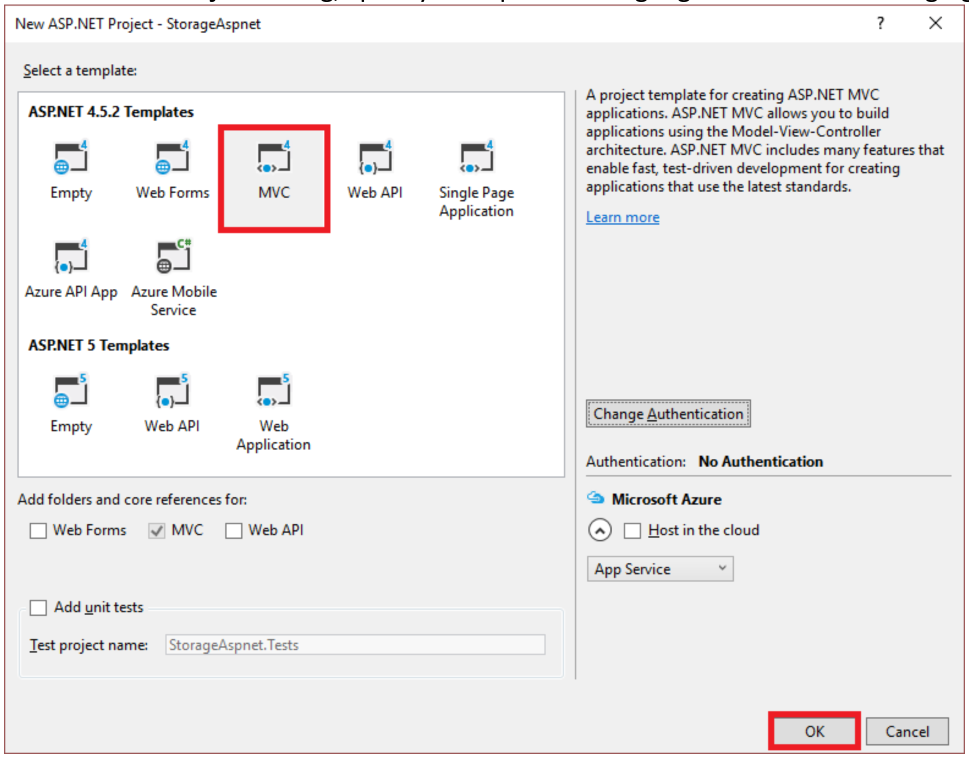
Type: Visual C#
A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights
☐ Add Application Insights to project
Optimize performance and monitor usage in your live application.

Name: StorageAspnet
Location: c:\source\
Solution name: StorageAspnet
☑ Create directory for solution
☐ Add to source control

Select **OK**.

On the **New ASP.NET Project** dialog, specify the options as highlighted in the following figure:



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**

Add folders and core references for:

☐ Web Forms ☑ MVC ☐ Web API

☐ Host in the cloud

☐ Add unit tests

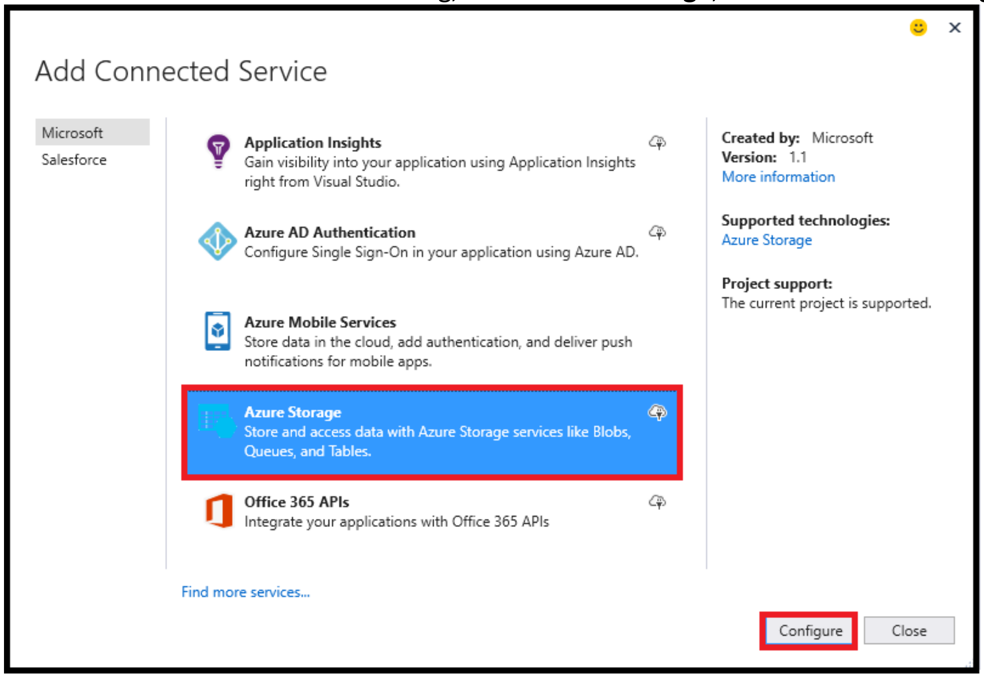
Test project name: StorageAspnet.Tests

Select **OK**.

Use Connected Services to connect to an Azure storage account

In the **Solution Explorer**, right-click the project, and from the context menu, select **Add->Connected Service**.

On the **Add Connected Service** dialog, select **Azure Storage**, and then select **Configure**.



Application Insights
Gain visibility into your application using Application Insights right from Visual Studio.

Azure AD Authentication
Configure Single Sign-On in your application using Azure AD.

Azure Mobile Services
Store data in the cloud, add authentication, and deliver push notifications for mobile apps.

Azure Storage
Store and access data with Azure Storage services like Blobs, Queues, and Tables.

Office 365 APIs
Integrate your applications with Office 365 APIs

Created by: Microsoft
Version: 1.1
More information

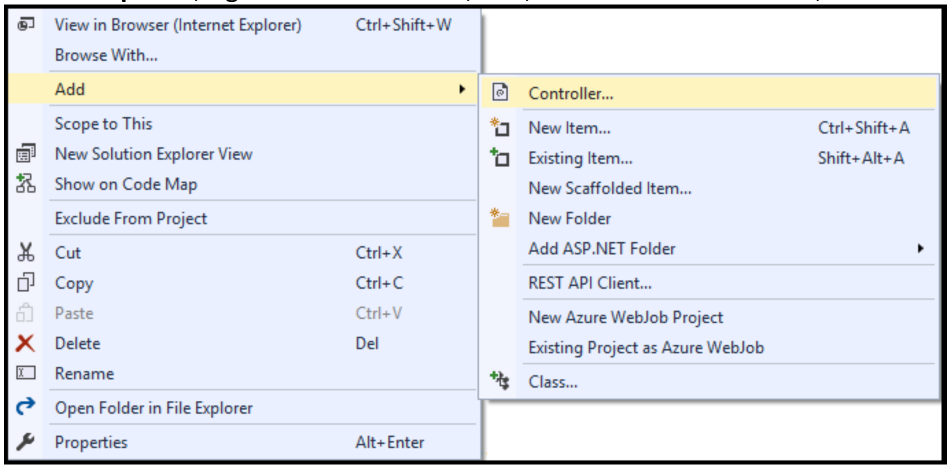
Supported technologies:
Azure Storage

Project support:
The current project is supported.

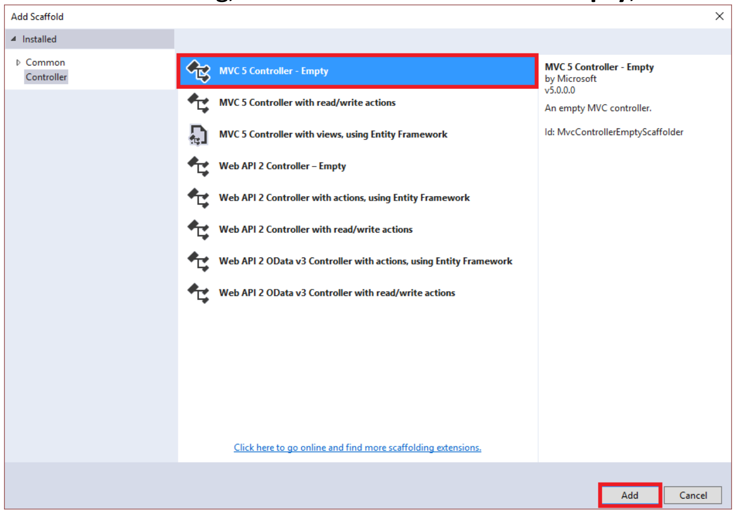
On the **Azure Storage** dialog, select the desired Azure storage account with which you want to work, and select **Add**.

Create MVC

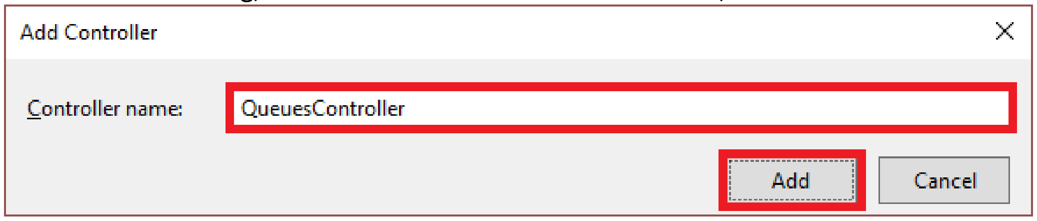
In the **Solution Explorer**, right-click **Controllers**, and, from the context menu, select **Add->Controller**.



On the **Add Scaffold** dialog, select **MVC 5 Controller - Empty**, and select **Add**.


Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions
MVC 5 Controller - Empty
by Microsoft
v5.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder
Click here to go online and find more scaffolding extensions.
Add
Cancel

On the **Add Controller** dialog, name the controller *QueuesController*, and select **Add**.



Controller name: QueuesController

Add the following *using* directives to the **QueuesController.cs** file:

```
using Microsoft.Azure;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Auth;
using Microsoft.WindowsAzure.Storage.Queue;
```

COMPLETE SOURCE CODE - File "QueuesController.cs"

```
using Microsoft.Azure;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Auth;
using Microsoft.WindowsAzure.Storage.Queue;
using Microsoft.WindowsAzure;
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace lab8webapp.Controllers
{
    public class QueuesController : Controller
    {
        // GET: Queues
        public ActionResult CreateQueue()
        {
            // The code in this section goes here.
            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
                CloudConfigurationManager.GetSetting("lab8stor_AzureStorageConnectionString"));
            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
            CloudQueue queue = queueClient.GetQueueReference("lab8test-queue");
            ViewBag.Success = queue.CreateIfNotExists();
            ViewBag.QueueName = queue.Name;
            return View();
        }
        public ActionResult AddMessage()
        {
            // The code in this section goes here.
            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
                CloudConfigurationManager.GetSetting("lab8stor_AzureStorageConnectionString"));
            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
            CloudQueue queue = queueClient.GetQueueReference("lab8test-queue");
            CloudQueueMessage message = new CloudQueueMessage("Hello, Azure Lab8 Queue Storage; this is message");
            queue.AddMessage(message);
            ViewBag.QueueName = queue.Name;
            ViewBag.Message = message.AsString;
            return View();
        }
        public ActionResult PeekMessage()
        {
            // The code in this section goes here.
            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
                CloudConfigurationManager.GetSetting("lab8stor_AzureStorageConnectionString"));
            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
            CloudQueue queue = queueClient.GetQueueReference("lab8test-queue");
            CloudQueueMessage message = queue.PeekMessage();
            ViewBag.QueueName = queue.Name;
            ViewBag.Message = (message != null ? message.AsString : "");
            return View();
        }
        public ActionResult ReadMessage()
        {
            // The code in this section goes here.
            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
               CloudConfigurationManager.GetSetting("lab8stor_AzureStorageConnectionString"));
            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
            CloudQueue queue = queueClient.GetQueueReference("lab8test-queue");
            // This message will be invisible to other code for 30 seconds.
            CloudQueueMessage message = queue.GetMessage();
            queue.DeleteMessage(message);
            ViewBag.QueueName = queue.Name;
            ViewBag.Message = message.AsString;
            return View();
        }
        public ActionResult GetQueueLength()
        {
            // The code in this section goes here.
            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
              CloudConfigurationManager.GetSetting("lab8stor_AzureStorageConnectionString"));
            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
            CloudQueue queue = queueClient.GetQueueReference("lab8test-queue");
            queue.FetchAttributes();
            int? nMessages = queue.ApproximateMessageCount;
            ViewBag.QueueName = queue.Name;
            ViewBag.Length = nMessages;
```

```
                return View();
            }
            public ActionResult DeleteQueue()
            {
                // The code in this section goes here.
                CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
                    CloudConfigurationManager.GetSetting("lab8stor_AzureStorageConnectionString"));
                CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
                CloudQueue queue = queueClient.GetQueueReference("lab8test-queue");
                queue.Delete();
                ViewBag.QueueName = queue.Name;
                return View();
            }

        }

    }
```

Create a queue: The following steps illustrate how to create a queue:

1. Open the **"QueuesController.cs"** file.
2. Add a method called **CreateQueue** that returns an **ActionResult**.

```
public ActionResult CreateQueue()
{ // The code in this section goes here.
    return View();
}
```

Within the **CreateQueue** method, get a **CloudStorageAccount** object that represents your storage account information. Use the following code to get the storage connection string and storage account information from the Azure service configuration: (Change *<storage-account-name>* to the name of the Azure storage account you're accessing.)

```
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
   CloudConfigurationManager.GetSetting("<storage-account-
name>_AzureStorageConnectionString"));
```

Get a **CloudQueueClient** object represents a queue service client.

```
CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
```

Get a **CloudQueue** object that represents a reference to the desired queue name. The **CloudQueueClient.GetQueueReference** method does not make a request against queue storage. The reference is returned whether or not the queue exists.

```
CloudQueue queue = queueClient.GetQueueReference("test-queue");
```

Call the **CloudQueue.CreateIfNotExists** method to create the queue if it does not yet exist. The **CloudQueue.CreateIfNotExists** method returns **true** if the queue does not exist, and is successfully created. Otherwise, **false** is returned.

```
ViewBag.Success = queue.CreateIfNotExists();
```

Update the **ViewBag** with the name of the queue.

```
ViewBag.QueueName = queue.Name;
```

In the **Solution Explorer**, expand the **Views** folder, right-click **Queues**, and from the context menu, select **Add->View**.

On the **Add View** dialog, enter **CreateQueue** for the view name, and select **Add**.
Open "**CreateQueue.cshtml**" filr and modify it so that it looks like the following code snippet:

```
@{
    ViewBag.Title = "CreateQueue";
}
<h2>CreateQueue RESULTS</h2>
Creation of @ViewBag.QueueName @(ViewBag.Success == true ? "succeeded" : "failed")
```

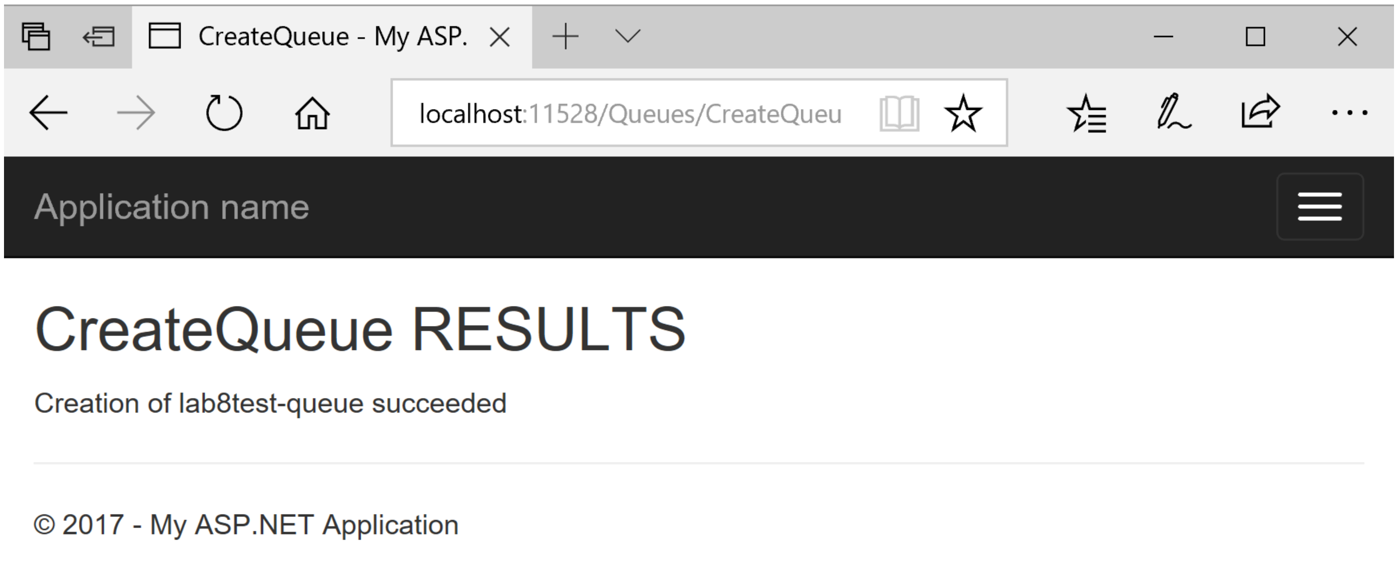
In the **Solution Explorer**, expand the **Views->Shared** folder, and open **_Layout.cshtml** file
After the last **Html.ActionLink**, add the following **Html.ActionLink**:

```
<li>@Html.ActionLink("Create queue", "CreateQueue", "Queues")</li>
```

The **ENTIRE CODE SNIPPET** for "**_Layout.cshtml**" file is the following:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</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("Application name", "Index", "Home", new { area = "" }, new
{ @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                    <li>@Html.ActionLink("Create queue", "CreateQueue", "Queues")</li>
                    <li>@Html.ActionLink("Delete queue", "DeleteQueue", "Queues")</li>
                    <li>@Html.ActionLink("Add message", "AddMessage", "Queues")</li>
                    <li>@Html.ActionLink("Peek message", "PeekMessage", "Queues")</li>
                    <li>@Html.ActionLink("Read/Delete message", "ReadMessage", "Queues")</li>
                    <li>@Html.ActionLink("Get queue length", "GetQueueLength", "Queues")</li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - McKesson Deep Azure Lab8 </p>
        </footer>
    </div>

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

Run the application, and select **Create queue** to see results similar to the following screen shot:



CreateQueue RESULTS

Creation of lab8test-queue succeeded

© 2017 - My ASP.NET Application

As mentioned previously, the **CloudQueue.CreateIfNotExists** method returns **true** only when the queue doesn't exist and is created. Therefore, if you run the app when the queue exists, the method returns **false**. To run the app multiple times, you must delete the queue before running the app again. Deleting the queue can be done via the **CloudQueue.Delete** method. You can also delete the queue using the Azure portal or the Microsoft Azure Storage Explorer.

Add a message to a queue

Once you've created a queue, you can add messages to that queue. This section walks you through adding a message to a queue *test-queue*.

1. Open the QueuesController.cs file.
2. Add a method called **AddMessage** that returns an **ActionResult**.

Within the **AddMessage** method, get a **CloudStorageAccount** object that represents your storage account information. Use the following code to get the storage connection string and storage account information from the Azure service configuration: (Change *<storage-account-name>* to the name of the Azure storage account you're accessing.)

Create the **CloudQueueMessage** object representing the message you want to add to the queue. A **CloudQueueMessage** object can be created from either a string (in UTF-8 format) or a byte array.

```
  public ActionResult AddMessage()
        { // The code in this section goes here.
            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
                CloudConfigurationManager.GetSetting("lab8stor_AzureStorageConnectionString"));
            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
            CloudQueue queue = queueClient.GetQueueReference("lab8test-queue");
            CloudQueueMessage message = new CloudQueueMessage("Hello, Azure Lab8 Queue Storage; this is
message");
            queue.AddMessage(message);
            ViewBag.QueueName = queue.Name;
            ViewBag.Message = message.AsString;
            return View();
        }
```

In the **Solution Explorer**, expand the **Views** folder, right-click **Queues**, and from the context menu, select **Add->View**.

On the **Add View** dialog, enter **AddMessage** for the view name, and select **Add**.

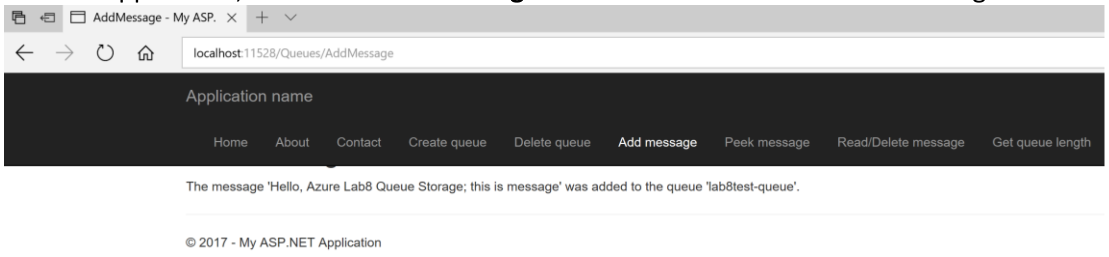
Open **AddMessage.cshtml,** and modify it so that it looks like the following code snippet:

```
@{
    ViewBag.Title = "AddMessage";
}
<h2>Add Message RESULTS</h2>
The message '@ViewBag.Message' was added to the queue '@ViewBag.QueueName'.
```

In the **Solution Explorer**, expand the **Views->Shared** folder, and open _Layout.cshtml.
After the last **Html.ActionLink**, add the following **Html.ActionLink**:
html

```
<li>@Html.ActionLink("Add message", "AddMessage", "Queues")</li>
```

Run the application, and select **Add message** to see results similar to the following screen shot:



The message 'Hello, Azure Lab8 Queue Storage; this is message' was added to the queue 'lab8test-queue'.

© 2017 - My ASP.NET Application

The two sections - Read a message from a queue without removing it and Read and remove a message from a queue - illustrate how to read messages from a queue.

Read a message from a queue without removing it
This section illustrates how to peek at a queued message (read the first message without removing it).

1. Open the "**QueuesController.cs**" file.
2. Add a method called **PeekMessage** that returns an **ActionResult**.
   Within the **PeekMessage** method, get a **CloudStorageAccount** object that represents your storage account information. Use the following code to get the storage connection string and storage account information from the Azure service configuration: (Change *<storage-account-name>* to the name of the Azure storage account you're accessing.)

   Get a **CloudQueueClient** object represents a queue service client.
   Get a **CloudQueueContainer** object that represents a reference to the queue.
   Call the **CloudQueue.PeekMessage** method to read the first message in the queue without removing it from the queue.
   Update the **ViewBag** with two values: the queue name and the message that was read. The **CloudQueueMessage** object exposes two properties for getting the object's value: **CloudQueueMessage.AsBytes** and **CloudQueueMessage.AsString**. **AsString** (used in this example) returns a string, while **AsBytes** returns a byte array.

```
public ActionResult PeekMessage()
{
    // The code in this section goes here.
    CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
        CloudConfigurationManager.GetSetting("lab8stor_AzureStorageConnectionString"));
    CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
    CloudQueue queue = queueClient.GetQueueReference("lab8test-queue");
    CloudQueueMessage message = queue.PeekMessage();
    ViewBag.QueueName = queue.Name;
    ViewBag.Message = (message != null ? message.AsString : "");
    return View();
}
```

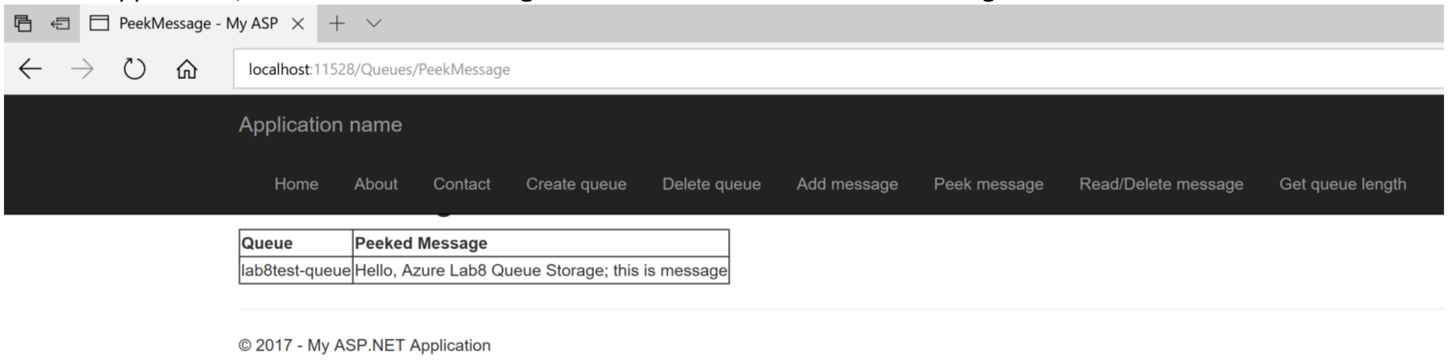
In the **Solution Explorer**, expand the **Views** folder, right-click **Queues**, and from the context menu, select **Add->View**.
On the **Add View** dialog, enter **PeekMessage** for the view name, and select **Add**.
Open **PeekMessage.cshtml**, and modify it so that it looks like the following code snippet:

```
@{
    ViewBag.Title = "PeekMessage";
}

<h2>Peek Message RESULTS</h2>

<table border="1">
    <tr><th>Queue</th><th>Peeked Message</th></tr>
    <tr><td>@ViewBag.QueueName</td><td>@ViewBag.Message</td></tr>
</table>
```

In the **Solution Explorer**, expand the **Views->Shared** folder, and open _Layout.cshtml.
After the last **Html.ActionLink**, add the following **Html.ActionLink**:

```
<li>@Html.ActionLink("Peek message", "PeekMessage", "Queues")</li>
```

Run the application, and select **Peek message** to see results similar to the following screen shot:



Queue	Peeked Message
lab8test-queue	Hello, Azure Lab8 Queue Storage; this is message

© 2017 - My ASP.NET Application

Read and remove a message from a queue

In this section, you learn how to read and remove a message from a queue.

1. Open the "**QueuesController.cs**" file.
2. Add a method called **ReadMessage** that returns an **ActionResult**.

Within the **ReadMessage** method, get a **CloudStorageAccount** object that represents your storage account information. Use the following code to get the storage connection string and storage account information from the Azure service configuration: (Change *<storage-account-name>* to the name of the Azure storage account you're accessing.)
Get a **CloudQueueClient** object represents a queue service client.
Get a **CloudQueueContainer** object that represents a reference to the queue.
Call the **CloudQueue.GetMessage** method to read the first message in the queue. The **CloudQueue.GetMessage** method makes the message invisible for 30 seconds (by default) to any other code reading messages so that no other code can modify or delete the message while your processing it. To change the amount of time the message is invisible, modify the **visibilityTimeout** parameter being passed to the **CloudQueue.GetMessage** method.
Call the **CloudQueueMessage.Delete** method to delete the message from the queue.
Update the **ViewBag** with the message deleted, and the name of the queue.

```
public ActionResult ReadMessage()
{
    // The code in this section goes here.
    CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
       CloudConfigurationManager.GetSetting("lab8stor_AzureStorageConnectionString"));
```

```
            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
            CloudQueue queue = queueClient.GetQueueReference("lab8test-queue");
            // This message will be invisible to other code for 30 seconds.
            CloudQueueMessage message = queue.GetMessage();
            queue.DeleteMessage(message);
            ViewBag.QueueName = queue.Name;
            ViewBag.Message = message.AsString;
            return View();
        }
```

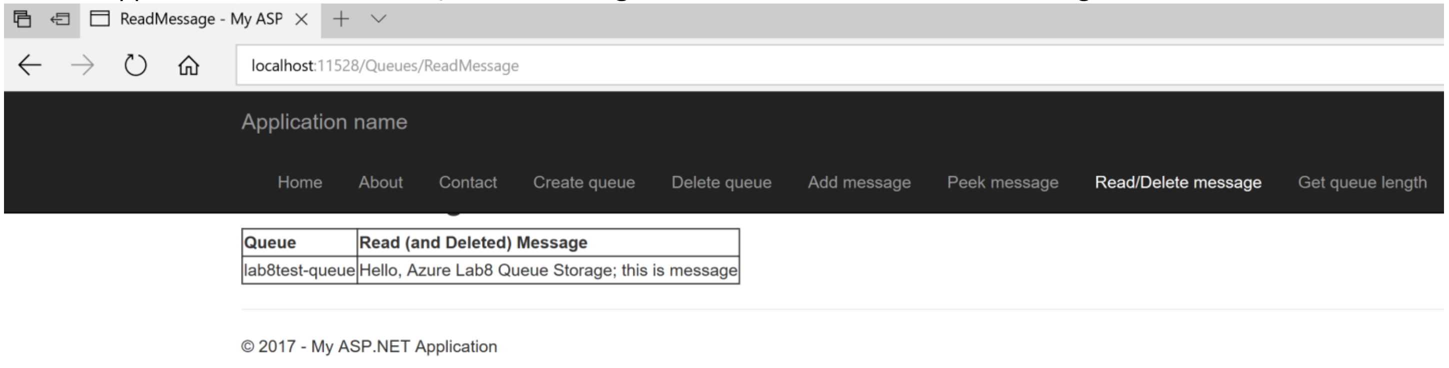
In the **Solution Explorer**, expand the **Views** folder, right-click **Queues**, and from the context menu, select **Add->View**.
On the **Add View** dialog, enter **ReadMessage** for the view name, and select **Add**.
Open **ReadMessage.cshtml,** and modify it so that it looks like the following code snippet:

```
@{
    ViewBag.Title = "ReadMessage";
}
<h2>Read Message RESULTS</h2>
<table border="1">
    <tr><th>Queue</th><th>Read (and Deleted) Message</th></tr>
    <tr><td>@ViewBag.QueueName</td><td>@ViewBag.Message</td></tr>
</table>
```

In the **Solution Explorer**, expand the **Views->Shared** folder, and open **_Layout.cshtml.**
After the last **Html.ActionLink**, add the following **Html.ActionLink**:

```
<li>@Html.ActionLink("Read/Delete message", "ReadMessage", "Queues")</li>
```

Run the application, and select **Read/Delete message** to see results similar to the following screen shot:



Queue	Read (and Deleted) Message
lab8test-queue	Hello, Azure Lab8 Queue Storage; this is message

© 2017 - My ASP.NET Application

Get the queue length

This section illustrates how to get the queue length (number of messages).

1. Open the "**QueuesController.cs**" file.
2. Add a method called **GetQueueLength** that returns an **ActionResult**.

Within the **ReadMessage** method, get a **CloudStorageAccount** object that represents your storage account information. Use the following code to get the storage connection string and storage account information from the Azure service configuration: (Change *<storage-account-name>* to the name of the Azure storage account you're accessing.)
Get a **CloudQueueClient** object represents a queue service client.
Get a **CloudQueueContainer** object that represents a reference to the queue.
Call the **CloudQueue.FetchAttributes** method to retrieve the queue's attributes (including its length).
Access the **CloudQueue.ApproximateMessageCount** property to get the queue's length.
Update the **ViewBag** with the name of the queue, and its length.

```
        public ActionResult GetQueueLength()
        {
            // The code in this section goes here.
            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
              CloudConfigurationManager.GetSetting("lab8stor_AzureStorageConnectionString"));
            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
            CloudQueue queue = queueClient.GetQueueReference("lab8test-queue");
            queue.FetchAttributes();
            int? nMessages = queue.ApproximateMessageCount;
            ViewBag.QueueName = queue.Name;
            ViewBag.Length = nMessages;
            return View();
        }
```

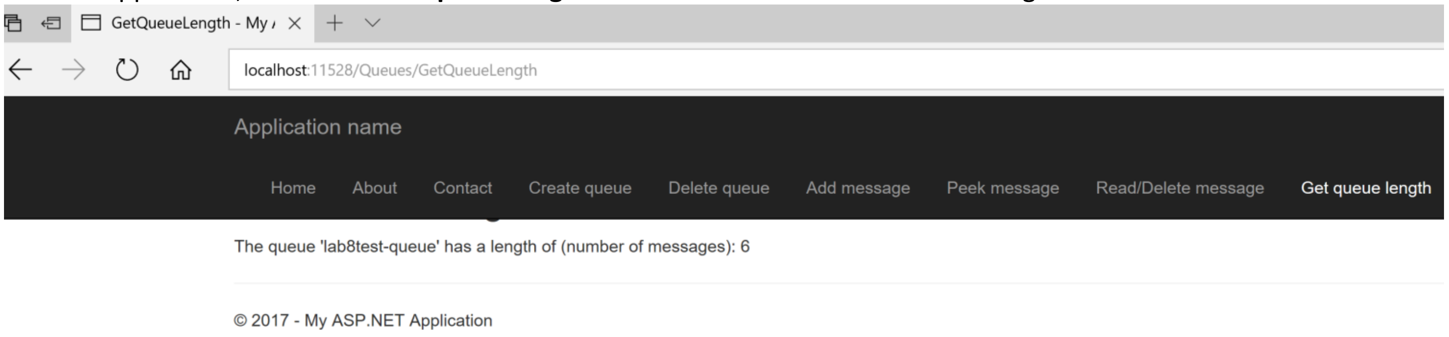
In the **Solution Explorer**, expand the **Views** folder, right-click **Queues**, and from the context menu, select **Add->View**.
On the **Add View** dialog, enter **GetQueueLength** for the view name, and select **Add**.
Open **GetQueueLengthMessage.cshtml,** and modify it so that it looks like the following code snippet:

```
@{
    ViewBag.Title = "GetQueueLength";
}
<h2>Get Queue Length RESULTS</h2>
The queue '@ViewBag.QueueName' has a length of (number of messages): @ViewBag.Length
```

In the **Solution Explorer**, expand the **Views->Shared** folder, and open **_Layout.cshtml**.
 After the last **Html.ActionLink**, add the following **Html.ActionLink**:

```
   <li>@Html.ActionLink("Get queue length", "GetQueueLength", "Queues")</li>
```

Run the application, and select **Get queue length** to see results similar to the following screen shot:



The queue 'lab8test-queue' has a length of (number of messages): 6

© 2017 - My ASP.NET Application

Delete a queue
This section illustrates how to delete a queue.

1. Open the "**QueuesController.cs**" file.
2. Add a method called **DeleteQueue** that returns an **ActionResult**.

Within the **DeleteQueue** method, get a **CloudStorageAccount** object that represents your storage account information. Use the following code to get the storage connection string and storage account information from the Azure service configuration: (Change *<storage-account-name>* to the name of the Azure storage account you're accessing.)
Get a **CloudQueueClient** object represents a queue service client.
Get a **CloudQueueContainer** object that represents a reference to the queue.
Call the **CloudQueue.Delete** method to delete the queue represented by the **CloudQueue** object.
Update the **ViewBag** with the name of the queue, and its length.

```
        public ActionResult DeleteQueue()
        {
            // The code in this section goes here.
            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
                CloudConfigurationManager.GetSetting("lab8stor_AzureStorageConnectionString"));
            CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
            CloudQueue queue = queueClient.GetQueueReference("lab8test-queue");
            queue.Delete();
            ViewBag.QueueName = queue.Name;
            return View();
        }
```

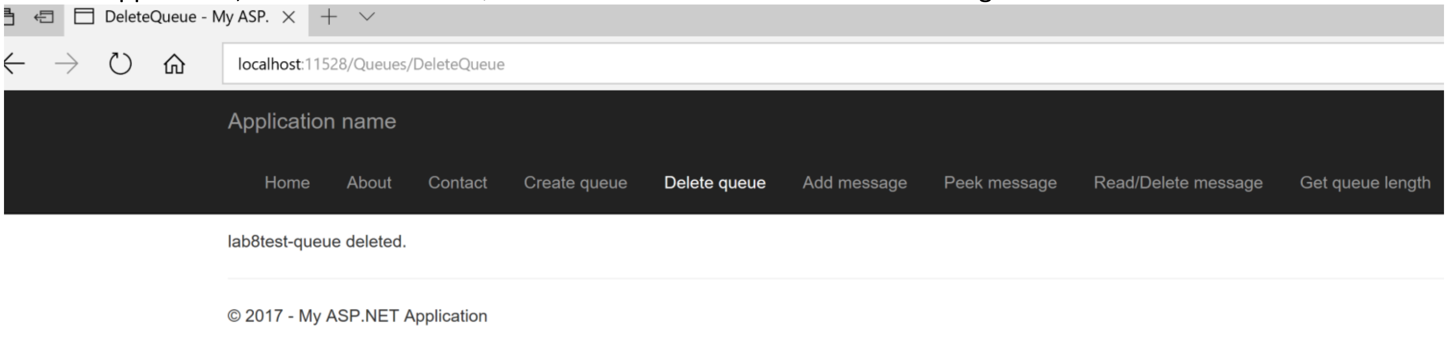
In the **Solution Explorer**, expand the **Views** folder, right-click **Queues**, and from the context menu, select **Add->View**.
On the **Add View** dialog, enter **DeleteQueue** for the view name, and select **Add**.
Open **DeleteQueue.cshtml,** and modify it so that it looks like the following code snippet:

```
@{
    ViewBag.Title = "DeleteQueue";
}

<h2>Delete Queue RESULTS</h2>

@ViewBag.QueueName deleted.
```

In the **Solution Explorer**, expand the **Views->Shared** folder, and open **_Layout.cshtml.**
After the last **Html.ActionLink**, add the following **Html.ActionLink**:

```
<li>@Html.ActionLink("Delete queue", "DeleteQueue", "Queues")</li>
```

Run the application, and select **Delete Queue** to see results similar to the following screen shot:



lab8test-queue deleted.

© 2017 - My ASP.NET Application

References
Try the Microsoft Azure Storage Explorer
Microsoft Azure Storage Explorer is a free, standalone app from Microsoft that enables you to work visually with Azure Storage data on Windows, macOS, and Linux.

https://docs.microsoft.com/en-us/azure/visual-studio/vs-storage-aspnet-getting-started-queues