Get started with Azure Blob storage and Visual Studio connected services (ASP.NET)

12/07/2017 12 minutes to read Contributors all

In this article

ASP.NET

Azure Blob storage is a service that stores unstructured data in the cloud as objects or blobs. Blob storage can store any type of text or binary data, such as a document, media file, or application installer. Blob storage is also referred to as object storage.

This tutorial shows how to write ASP.NET code for some common scenarios that use Blob storage. Scenarios include creating a blob container, and uploading, listing, downloading, and deleting blobs.

> **Tip**
>
> Manage Azure Blob storage resources with Azure Storage Explorer. Azure Storage Explorer is a free, standalone app from Microsoft that enables you to manage Azure Blob storage resources. Using Azure Storage Explorer, you can visually create, read, update, and delete blob containers and blobs, as well as manage access to your blobs containers and blobs.

Prerequisites

- Microsoft Visual Studio

Azure Blob storage is Microsoft's object storage solution for the cloud. Blob storage is optimized for storing massive amounts of unstructured data, such as text or binary data.

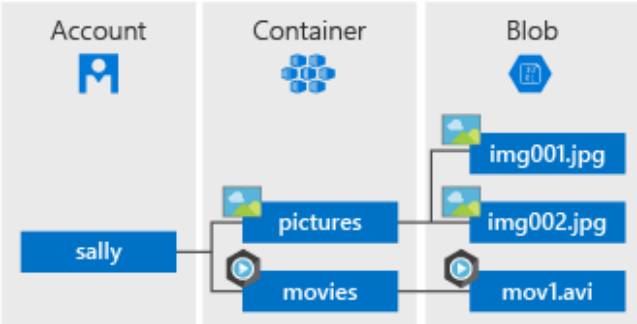
Blob storage is ideal for:

- Serving images or documents directly to a browser.
- Storing files for distributed access.
- Streaming video and audio.
- Writing to log files.
- Storing data for backup and restore, disaster recovery, and archiving.
- Storing data for analysis by an on-premises or Azure-hosted service.

Users or client applications can access Blob storage objects via HTTP or HTTPS—from anywhere in the world—via URLs, the Azure Storage REST API, Azure PowerShell, Azure CLI, or an Azure Storage client library. The storage client libraries are available for various languages, including .NET, Java, Node.js, Python, PHP, and Ruby.

Blob service concepts

Blob storage exposes three resources: your storage account, the containers in the account, and the blobs in a container. The following diagram shows the relationship between these resources.



Storage account

All access to data objects in Azure Storage happens through a storage account. For more information, see Azure storage account overview.

Container

A container organizes a set of blobs, similar to a folder in a file system. All blobs reside within a container. A storage account can include an unlimited number of containers, and a container can store an unlimited number of blobs.

> ⓘ **Note**
>
> The container name must be lowercase.

Blob

Azure Storage offers three types of blobs—block blobs, append blobs, and page blobs (used for VHD files).

- Block blobs store text and binary data, up to about 4.7 TB. Block blobs are made up of blocks of data that can be managed individually.
- Append blobs are made up of blocks like block blobs, but are optimized for append operations. Append blobs are ideal for scenarios such as logging data from virtual machines.
- Page blobs store random access files up to 8 TB in size. Page blobs store the VHD files that back VMs.

All blobs reside within a container. A container is similar to a folder in a file system. You can further organize blobs into virtual directories and navigate them as you would a file system.

There may be times where large datasets and network constraints make uploading data to Blob storage over the wire unrealistic. You can use Azure Data Box Disk to request solid-state disks (SSDs) from Microsoft. You can then copy your data to those disks and ship them back to Microsoft to be uploaded into Blob storage.

If you need to export large amounts of data from your storage account, see Use the Microsoft Azure Import/Export service to transfer data to Blob storage.

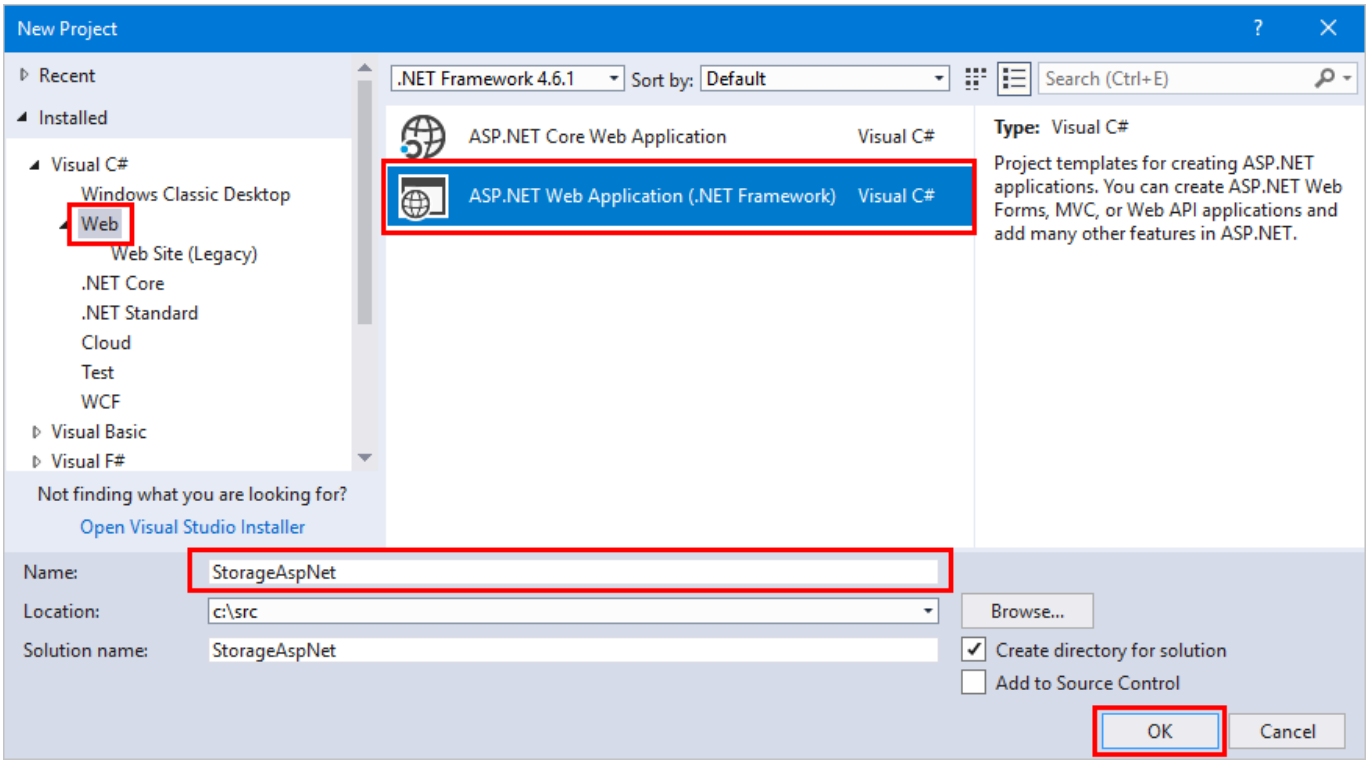
For details about naming containers and blobs, see Naming and referencing containers, blobs, and metadata.

Set up the development environment

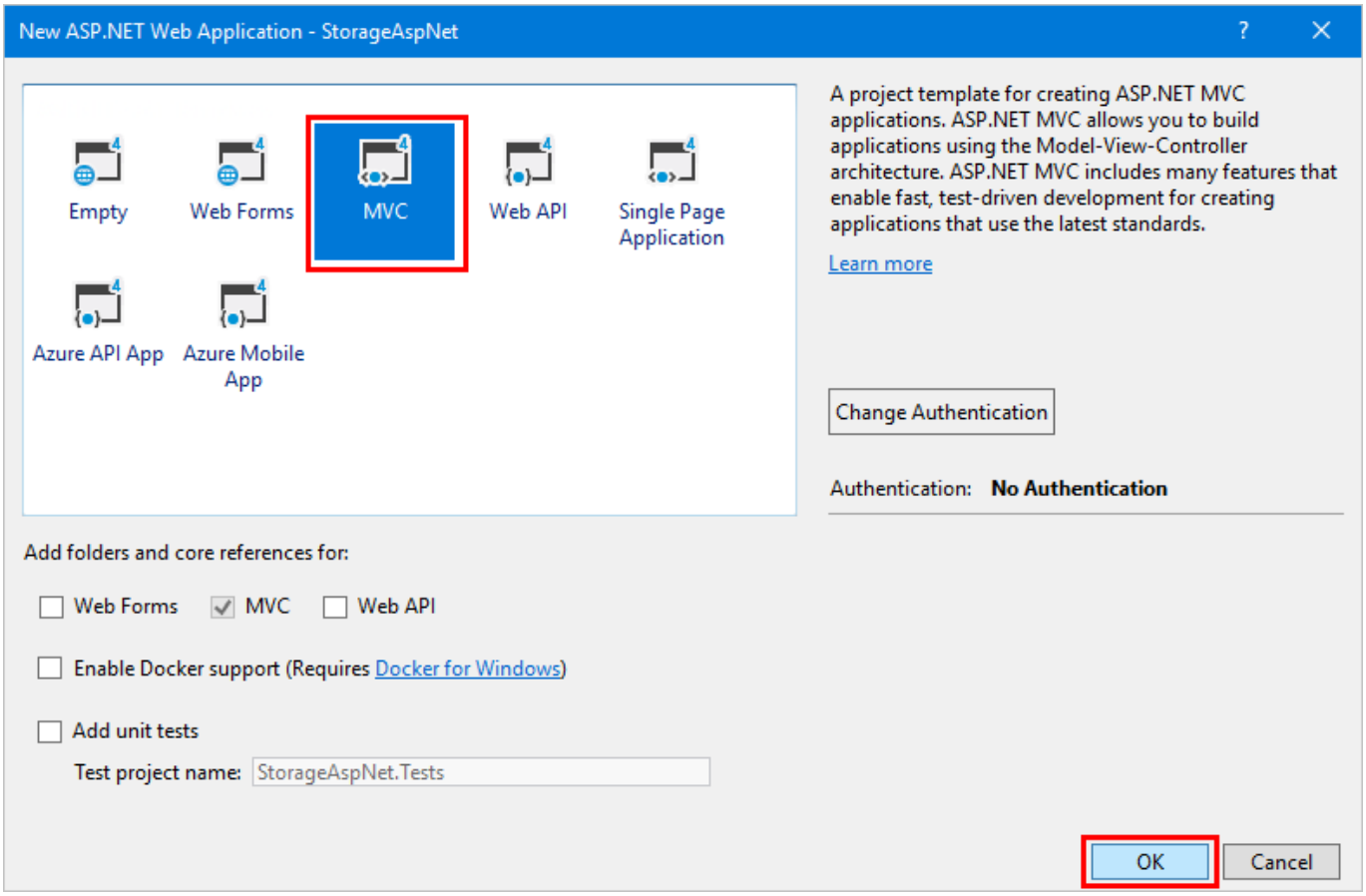
This section walks through setting up the development environment. This includes creating an ASP.NET MVC app, adding a connected services connection, adding a controller, and specifying the required namespace directives.

Create an ASP.NET MVC app project

1. Open Visual Studio.
2. From the main menu, select **File** > **New** > **Project**.
3. In the **New Project** dialog box, select **Web** > **ASP.NET Web Application (.NET Framework)**. In the **Name** field, specify **StorageAspNet**. Select **OK**.



4. In the **New ASP.NET Web Application** dialog box, select **MVC**, and then select **OK**.

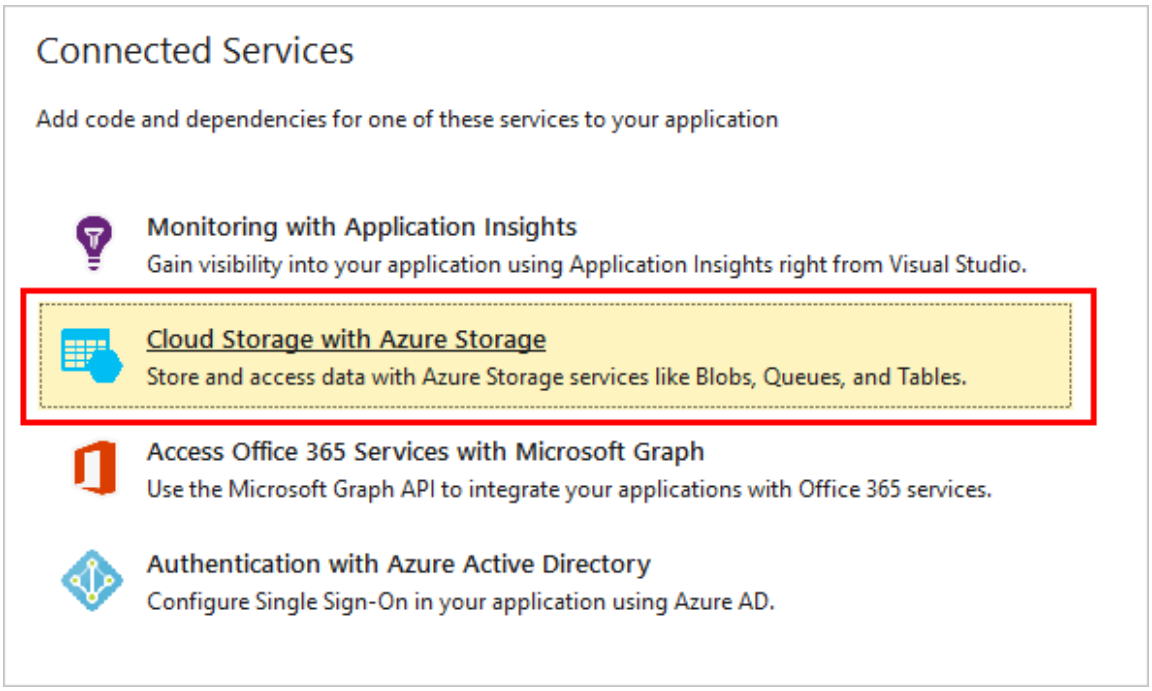


Use connected services to connect to an Azure storage account

1. In **Solution Explorer**, right-click the project.

2. From the context menu, select **Add** > **Connected Service**.

3. In the **Connected Services** dialog box, select **Cloud Storage with Azure Storage**.



4. In the **Azure Storage** dialog box, select the Azure storage account to be used for this tutorial. To create a new Azure storage account, select **Create a New Storage Account**, and complete the form. After selecting either an existing storage account or creating a new one, select **Add**. Visual Studio installs the NuGet package for Azure Storage and a storage connection string to **Web.config**.

> **Tip**

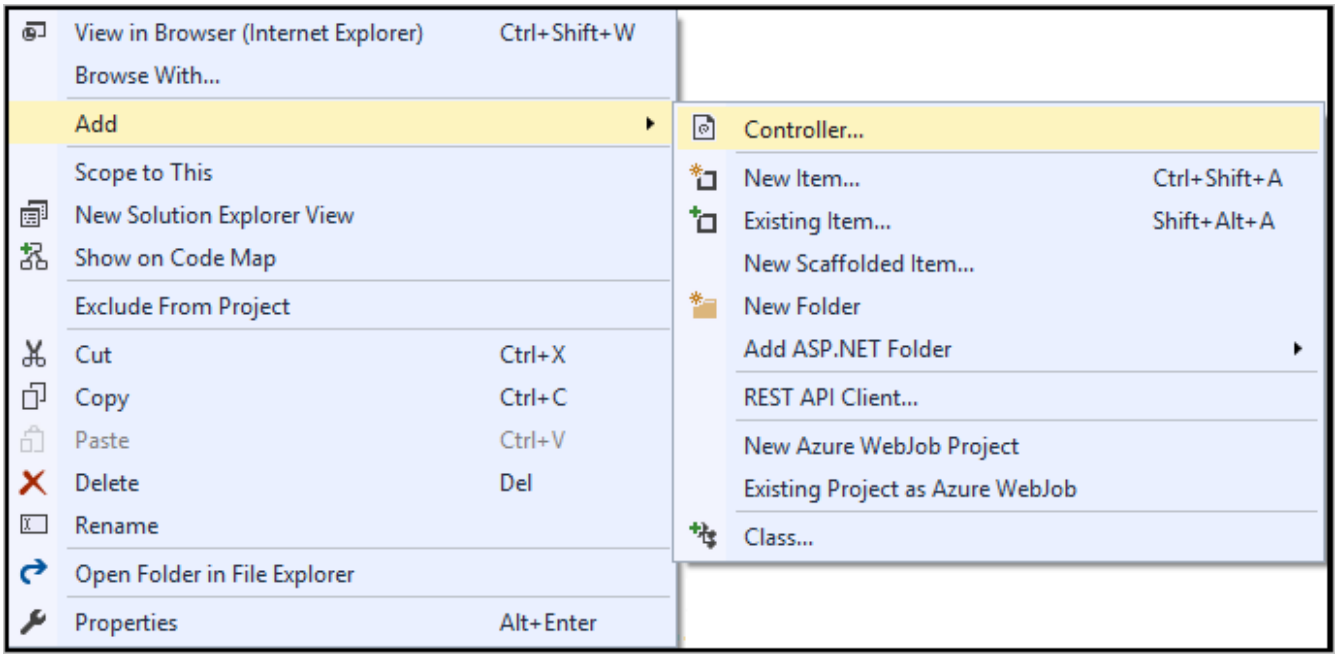
To learn how to create a storage account with the Azure portal, see Create a storage account.

You can also create a storage account by using Azure PowerShell, Azure CLI, or Azure Cloud Shell.

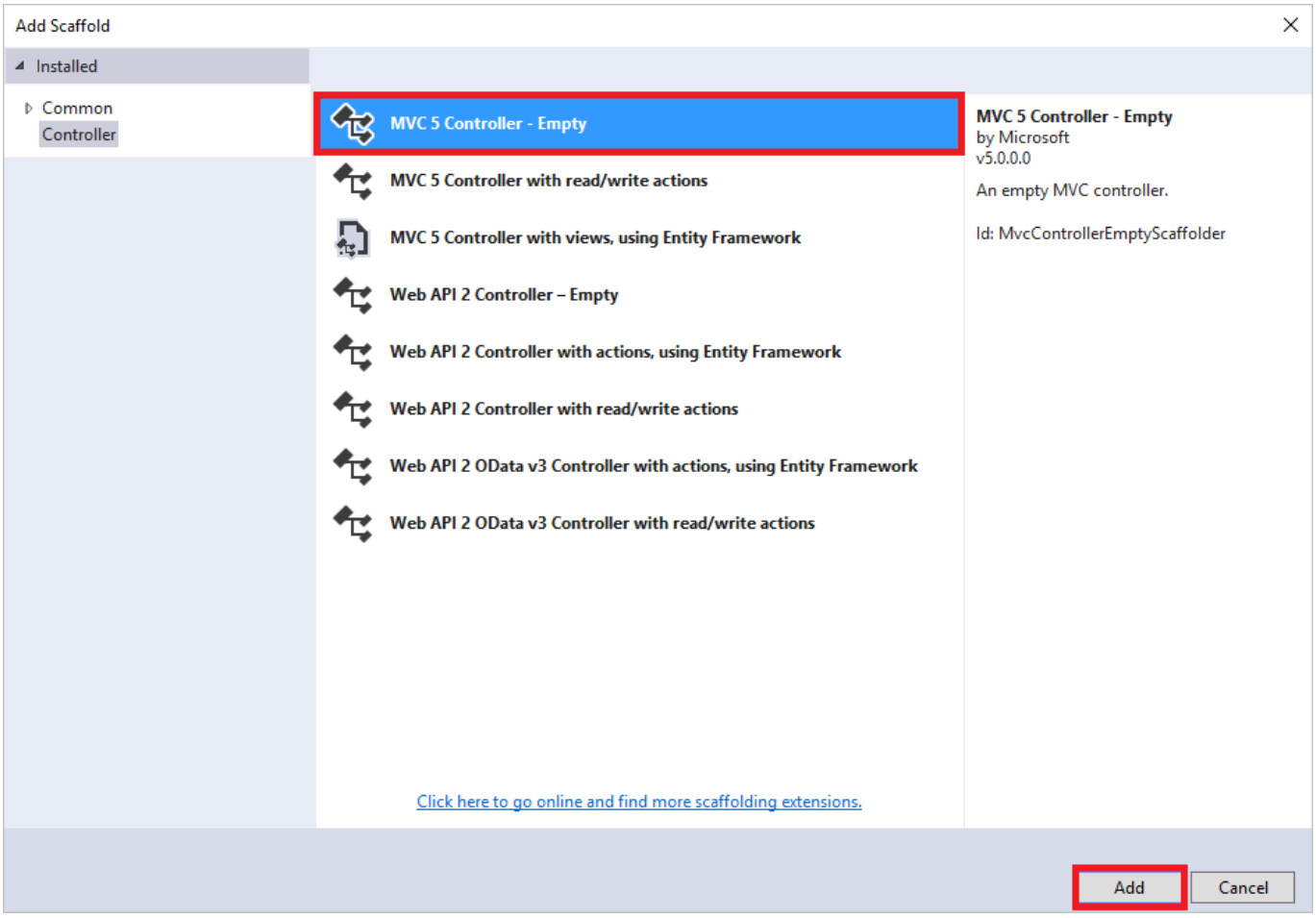
Create an MVC controller

1. In **Solution Explorer**, right-click **Controllers**.

2. From the context menu, select **Add** > **Controller**.



3. In the **Add Scaffold** dialog box, select **MVC 5 Controller - Empty**, and select **Add**.



4. In the **Add Controller** dialog box, name the controller *BlobsController*, and select **Add**.

5. Add the following `using` directives to the `BlobsController.cs` file:

```
C#
```

Copy

```
using Microsoft.Azure;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Blob;
```

Connect to a storage account and get a container reference

A blob container is a nested hierarchy of blobs and folders. The rest of the steps in this document require a reference to a blob container, so that code should be placed in its own method for reusability.

The following steps create a method to connect to the storage account by using the connection string in **Web.config**. The steps also create a reference to a container. The connection string setting in **Web.config** is named with the format `<storageaccountname>_AzureStorageConnectionString`.

1. Open the `BlobsController.cs` file.
2. Add a method called **GetCloudBlobContainer** that returns a **CloudBlobContainer**. Be sure to replace `<storageaccountname>_AzureStorageConnectionString` with the actual name of the key in **Web.config**.

```
C#
```

Copy

```
private CloudBlobContainer GetCloudBlobContainer()
{
    CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
            CloudConfigurationManager.GetSetting("
<storageaccountname>_AzureStorageConnectionString"));
    CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
    CloudBlobContainer container = blobClient.GetContainerReference("test-blob-
container");
    return container;
}
```

Note

Even though *test-blob-container* doesn't exist yet, this code creates a reference to it. This is so the container can be created with the `CreateIfNotExists` method shown in the next step.

Create a blob container

The following steps illustrate how to create a blob container:

1. Add a method called `CreateBlobContainer` that returns an `ActionResult`.

```
C#
```



```
public ActionResult CreateBlobContainer()
{
    // The code in this section goes here.

    return View();
}
```

2. Get a `CloudBlobContainer` object that represents a reference to the desired blob container name.

```
C#
```



```
CloudBlobContainer container = GetCloudBlobContainer();
```

3. Call the `CloudBlobContainer.CreateIfNotExists` method to create the container, if it does not yet exist. The `CloudBlobContainer.CreateIfNotExists` method returns **true** if the container does not exist, and is successfully created. Otherwise, the method returns **false**.

```
C#
```



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

4. Update `ViewBag` with the name of the blob container.

```
C#
```



```
ViewBag.BlobContainerName = container.Name;
```

The following shows the completed `CreateBlobContainer` method:

```
C#
```



```
public ActionResult CreateBlobContainer()
{
    CloudBlobContainer container = GetCloudBlobContainer();
    ViewBag.Success = container.CreateIfNotExists();
    ViewBag.BlobContainerName = container.Name;

    return View();
}
```

5. In **Solution Explorer**, right-click the **Views** folder.

6. If there isn't a **Blobs** folder, create one. From the context menu, select **Add** > **New Folder**. Name the new folder *Blobs*.

7. In **Solution Explorer**, expand the **Views** folder, and right-click **Blobs**.

8. From the context menu, select **Add** > **View**.

9. In the **Add View** dialog box, enter **CreateBlobContainer** for the view name, and select **Add**.

10. Open `CreateBlobContainer.cshtml`, and modify it so that it looks like the following code snippet:

```
C#
```

```
@{
    ViewBag.Title = "Create Blob Container";
}

<h2>Create Blob Container results</h2>

Creation of @ViewBag.BlobContainerName @(ViewBag.Success == true ? "succeeded" :
"failed")
```

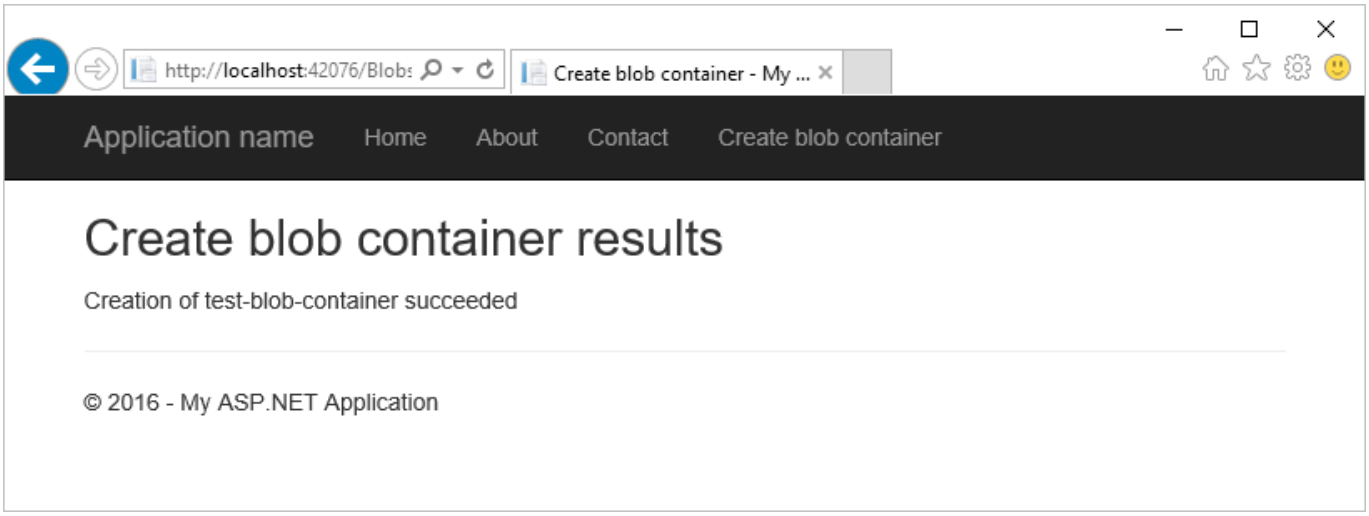
11. In **Solution Explorer**, expand the **Views** > **Shared** folder, and open `_Layout.cshtml`.

12. After the last **Html.ActionLink**, add the following **Html.ActionLink**:

```
HTML
```

```
<li>@Html.ActionLink("Create blob container", "CreateBlobContainer", "Blobs")</li>
```

13. Run the application, and select **Create Blob Container** to see results similar to the following screenshot:



As mentioned previously, the `CloudBlobContainer.CreateIfNotExists` method returns **true** only when the container doesn't exist and is created. Therefore, if the app is run when the container exists, the method returns **false**.

Upload a blob into a blob container

When the blob container is created, upload files into that container. This section walks through uploading a local file to a blob container. The steps assume there is a blob container named *test-blob-container*.

1. Open the `BlobsController.cs` file.

2. Add a method called `UploadBlob` that returns a string.

```
C#
```

```
public string UploadBlob()
{
    // The code in this section goes here.

    return "success!";
}
```

3. Within the `UploadBlob` method, get a `CloudBlobContainer` object that represents a reference to the desired blob container name.

```
C#
```

```
CloudBlobContainer container = GetCloudBlobContainer();
```

4. Azure storage supports different blob types. This tutorial uses block blobs. To retrieve a reference to a block blob, call the `CloudBlobContainer.GetBlockBlobReference` method.

```
C#
```

```
CloudBlockBlob blob = container.GetBlockBlobReference("myBlob");
```

> **Note**
>
> The blob name is part of the URL used to retrieve a blob, and can be any string, including the name of the file.

5. After there is a blob reference, you can upload any data stream to it by calling the blob reference object's `UploadFromStream` method. The `UploadFromStream` method creates the blob if it doesn't exist, or overwrites it if it does exist. (Change *<file-to-upload>* to a fully qualified path to a file to be uploaded.)

```
C#
```

```
using (var fileStream = System.IO.File.OpenRead(@"<file-to-upload>"))
{
    blob.UploadFromStream(fileStream);
}
```

The following shows the completed `UploadBlob` method (with a fully qualified path for the file to be uploaded):

C# Copy

```
public string UploadBlob()
{
    CloudBlobContainer container = GetCloudBlobContainer();
    CloudBlockBlob blob = container.GetBlockBlobReference("myBlob");
    using (var fileStream = System.IO.File.OpenRead(@"c:\src\sample.txt"))
    {
        blob.UploadFromStream(fileStream);
    }
    return "success!";
}
```

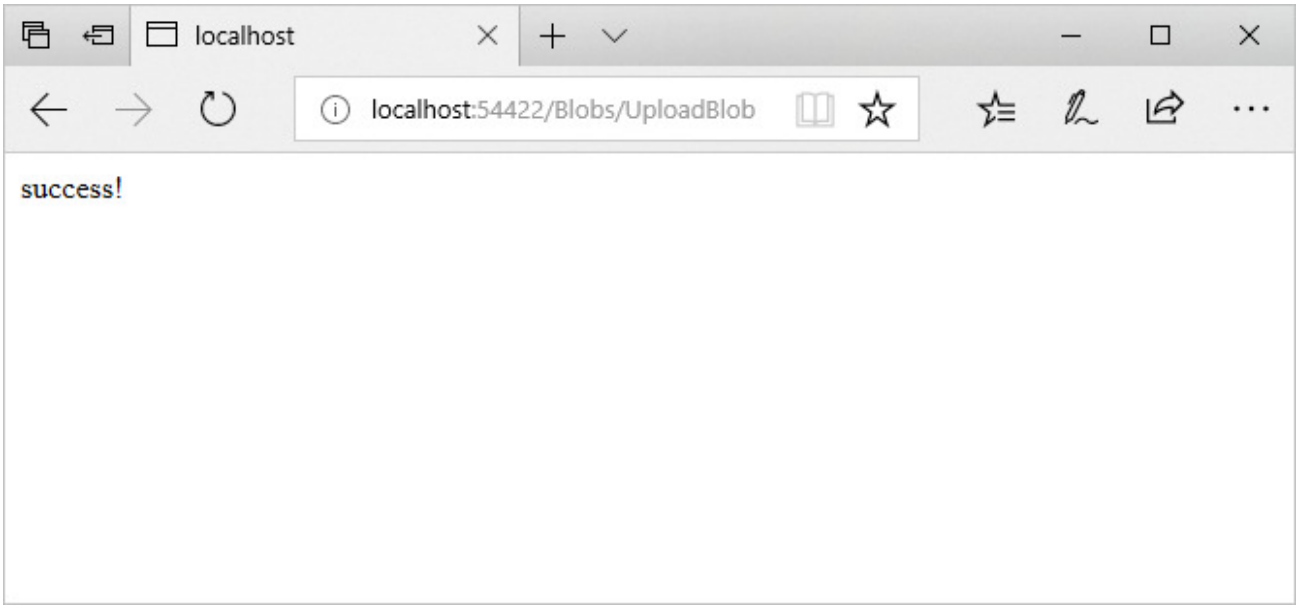
6. In **Solution Explorer**, expand the **Views** > **Shared** folder, and open `_Layout.cshtml`.

7. After the last **Html.ActionLink**, add the following **Html.ActionLink**:

HTML Copy

```
<li>@Html.ActionLink("Upload blob", "UploadBlob", "Blobs")</li>
```

8. Run the application, and select **Upload blob**. The word *success!* should appear.



List the blobs in a blob container

This section illustrates how to list the blobs in a blob container. The sample code references the *test-blob-container* created in the section, Create a blob container.

1. Open the `BlobsController.cs` file.
2. Add a method called `ListBlobs` that returns an `ActionResult`.

```
C#
```

```
public ActionResult ListBlobs()
{
    // The code in this section goes here.

}
```

3. Within the `ListBlobs` method, get a `CloudBlobContainer` object that represents a reference to the blob container.

```
C#
```

```
CloudBlobContainer container = GetCloudBlobContainer();
```

4. To list the blobs in a blob container, use the `CloudBlobContainer.ListBlobs` method. The `CloudBlobContainer.ListBlobs` method returns an `IListBlobItem` object that can be cast to a `CloudBlockBlob`, `CloudPageBlob`, or `CloudBlobDirectory` object. The following code snippet enumerates all the blobs in a blob container. Each blob is cast to the appropriate object, based on its type. Its name (or URI in the case of a **CloudBlobDirectory**) is added to a list.

```
C#
```

```
List<string> blobs = new List<string>();

foreach (IListBlobItem item in container.ListBlobs())
{
    if (item.GetType() == typeof(CloudBlockBlob))
    {
        CloudBlockBlob blob = (CloudBlockBlob)item;
        blobs.Add(blob.Name);
    }
    else if (item.GetType() == typeof(CloudPageBlob))
    {
        CloudPageBlob blob = (CloudPageBlob)item;
        blobs.Add(blob.Name);
    }
    else if (item.GetType() == typeof(CloudBlobDirectory))
    {
        CloudBlobDirectory dir = (CloudBlobDirectory)item;
        blobs.Add(dir.Uri.ToString());
    }
}
```

```
return View(blobs);
```

In addition to blobs, blob containers can contain directories. Suppose there is a blob container called *test-blob-container*, with the following hierarchy:

Copy

```
foo.png
dir1/bar.png
dir2/baz.png
```

Using the preceding code example, the **blobs** string list contains values similar to the following:

Copy

```
foo.png
<storage-account-url>/test-blob-container/dir1
<storage-account-url>/test-blob-container/dir2
```

As shown, the list includes only the top-level entities, not the nested ones (*bar.png* and *baz.png*). To list all the entities within a blob container, change the code so that the **CloudBlobContainer.ListBlobs** method is passed **true** for the **useFlatBlobListing** parameter.

C# Copy

```
//...
foreach (IListBlobItem item in container.ListBlobs(useFlatBlobListing:true))
//...
```

Setting the **useFlatBlobListing** parameter to **true** returns a flat listing of all entities in the blob container. This yields the following results:

Copy

```
foo.png
dir1/bar.png
dir2/baz.png
```

The following shows the completed **ListBlobs** method:

C# Copy

```
public ActionResult ListBlobs()
{
    CloudBlobContainer container = GetCloudBlobContainer();
    List<string> blobs = new List<string>();
    foreach (IListBlobItem item in container.ListBlobs(useFlatBlobListing: true))
```

```
    {
        if (item.GetType() == typeof(CloudBlockBlob))
        {
            CloudBlockBlob blob = (CloudBlockBlob)item;
            blobs.Add(blob.Name);
        }
        else if (item.GetType() == typeof(CloudPageBlob))
        {
            CloudPageBlob blob = (CloudPageBlob)item;
            blobs.Add(blob.Name);
        }
        else if (item.GetType() == typeof(CloudBlobDirectory))
        {
            CloudBlobDirectory dir = (CloudBlobDirectory)item;
            blobs.Add(dir.Uri.ToString());
        }
    }

    return View(blobs);
}
```

5. In **Solution Explorer**, expand the **Views** folder, and right-click **Blobs**.
6. From the context menu, select **Add** > **View**.
7. In the **Add View** dialog box, enter `ListBlobs` for the view name, and select **Add**.
8. Open `ListBlobs.cshtml`, and replace the contents with the following code:

HTML Copy

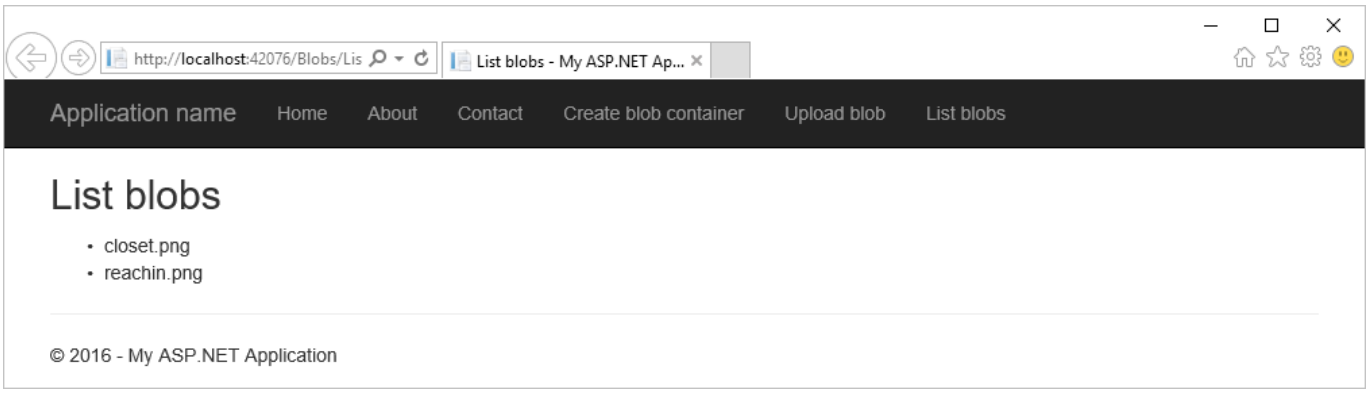
```
@model List<string>
@{
    ViewBag.Title = "List blobs";
}

<h2>List blobs</h2>

<ul>
    @foreach (var item in Model)
    {
    <li>@item</li>
    }
</ul>
```

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

HTML Copy

```
<li>@Html.ActionLink("List blobs", "ListBlobs", "Blobs")</li>
```

11. Run the application, and select **List blobs** to see results similar to the following screenshot:



Download blobs

This section illustrates how to download a blob. You can either persist it to local storage or read the contents into a string. The sample code references the *test-blob-container* created in the section, Create a blob container.

1. Open the `BlobsController.cs` file.
2. Add a method called `DownloadBlob` that returns a string.

```
C#
```

```
public string DownloadBlob()
{
    // The code in this section goes here.

    return "success!";
}
```

3. Within the `DownloadBlob` method, get a `CloudBlobContainer` object that represents a reference to the blob container.

```
C#
```

```
CloudBlobContainer container = GetCloudBlobContainer();
```

4. Get a blob reference object by calling the `CloudBlobContainer.GetBlockBlobReference` method.

```
C#
```

```
CloudBlockBlob blob = container.GetBlockBlobReference("myBlob");
```

5. To download a blob, use the `CloudBlockBlob.DownloadToStream` method. The following code transfers a blob's contents to a stream object. That object is then persisted to a local file. (Change

<local-file-name> to the fully qualified file name representing where the blob is to be downloaded.)

```
C#
```

```
using (var fileStream = System.IO.File.OpenWrite(<local-file-name>))
{
    blob.DownloadToStream(fileStream);
}
```

The following shows the completed `ListBlobs` method (with a fully qualified path for the local file being created):

```
C#
```

```
public string DownloadBlob()
{
    CloudBlobContainer container = GetCloudBlobContainer();
    CloudBlockBlob blob = container.GetBlockBlobReference("myBlob");
    using (var fileStream = System.IO.File.OpenWrite(@"c:\src\downloadedBlob.txt"))
    {
        blob.DownloadToStream(fileStream);
    }
    return "success!";
}
```

6. In **Solution Explorer**, expand the **Views** > **Shared** folder, and open `_Layout.cshtml`.

7. After the last **Html.ActionLink**, add the following **Html.ActionLink**:

```
HTML
```

```
<li>@Html.ActionLink("Download blob", "DownloadBlob", "Blobs")</li>
```

8. Run the application, and select **Download blob** to download the blob. The blob specified in the `CloudBlobContainer.GetBlockBlobReference` method call downloads to the location specified in the `File.OpenWrite` method call. The text *success!* should appear in the browser.

Delete blobs

The following steps illustrate how to delete a blob:

1. Open the `BlobsController.cs` file.

2. Add a method called `DeleteBlob` that returns a string.

```
C#
```

```
public string DeleteBlob()
{
    // The code in this section goes here.

    return "success!";
}
```

3. Within the `DeleteBlob` method, get a `CloudBlobContainer` object that represents a reference to the blob container.

C# Copy

```
CloudBlobContainer container = GetCloudBlobContainer();
```

4. Get a blob reference object by calling the `CloudBlobContainer.GetBlockBlobReference` method.

C# Copy

```
CloudBlockBlob blob = container.GetBlockBlobReference("myBlob");
```

5. To delete a blob, use the `Delete` method.

C# Copy

```
blob.Delete();
```

The completed `DeleteBlob` method should appear as follows:

C# Copy

```
public string DeleteBlob()
{
    CloudBlobContainer container = GetCloudBlobContainer();
    CloudBlockBlob blob = container.GetBlockBlobReference("myBlob");
    blob.Delete();
    return "success!";
}
```

6. In **Solution Explorer**, expand the **Views** > **Shared** folder, and open `_Layout.cshtml`.

7. After the last **Html.ActionLink**, add the following **Html.ActionLink**:

HTML Copy

```
<li>@Html.ActionLink("Delete blob", "DeleteBlob", "Blobs")</li>
```

8. Run the application, and select **Delete blob** to delete the blob specified in the `CloudBlobContainer.GetBlockBlobReference` method call. The text *success!* should appear in the browser. Select the browser's **Back** button, and then select **List blobs** to verify that the blob is no longer in the container.

Next steps

In this tutorial, you learned how to store, list, and retrieve blobs in Azure Storage by using ASP.NET. View more feature guides to learn about additional options for storing data in Azure.

- Get started with Azure Table storage and Visual Studio connected services (ASP.NET)
- Get started with Azure Queue storage and Visual Studio connected services (ASP.NET)