* 1. 

Hands-On Lab

Exploring Windows Azure Storage

Lab version: 2.0.0

Last updated: 1/12/2012



Contents

[Overview 3](#_Toc310346223)

[Exercise 1: Working with Tables 5](#_Toc310346224)

[Task 1 – Configuring Storage Account Settings 5](#_Toc310346225)

[Task 2 – Creating Classes to Model the Table Schema 9](#_Toc310346226)

[Task 3 – Creating the Chat User Interface 13](#_Toc310346227)

[Verification 17](#_Toc310346228)

[Exercise 2: Working with Blobs 19](#_Toc310346229)

[Task 1 – Retrieving Blob Data from Storage 19](#_Toc310346230)

[Task 2 – Uploading Blob Data to Storage 28](#_Toc310346231)

[Task 3 – Retrieving Metadata for Blobs in Storage 33](#_Toc310346232)

[Task 4 – Deleting Blobs from Storage 37](#_Toc310346233)

[Task 5 – Copying Blobs 40](#_Toc310346234)

[Task 6 – Taking Blob Snapshots 44](#_Toc310346235)

[Exercise 3: Working with Queues 48](#_Toc310346236)

[Task 1 – Creating the Initial Solution 48](#_Toc310346237)

[Task 2 – Sending Messages to the Queue 52](#_Toc310346238)

[Task 3 – Retrieving Messages from the Queue 56](#_Toc310346239)

[Verification 61](#_Toc310346240)

[Exercise 4: Working with Drives 63](#_Toc310346241)

[Task 1 – Exploring the PhotoAlbum Application 64](#_Toc310346242)

[Task 2 – Using a Windows Azure Drive to Move the Application to the Cloud 66](#_Toc310346243)

[Task 3 – Creating a New Drive in the Cloud 80](#_Toc310346244)

[Task 4 – Creating an NTFS Formatted VHD on Your Local Machine 89](#_Toc310346245)

[Task 5 – Deploying the Application and Uploading the Drive to Windows Azure 100](#_Toc310346246)

[Summary 108](#_Toc310346247)

Overview

* 1. Storage services provide persistent, durable storage in the Windows Azure compute emulator, and include blob and table service and the queue service. In addition, using Windows Azure Drives, your Windows Azure applications running in the cloud can use existing NTFS APIs to access a durable drive backed by blob storage.
  2. In this lab, you will examine the basic process of working with Windows Azure storage on the local compute emulator, and explore some of the features that are available to developers.

# Objectives

* 1. In this hands-on lab, you will learn how to:
  + Use the Table service
  + Use the Blob service
  + Use the Queue service
  + Create and read metadata
  + Use Windows Azure Drives

# Prerequisites

* 1. The following is required to complete this hands-on lab:
  + IIS 7 (with ASP.NET, WCF HTTP Activation, Tracing)
  + [Microsoft .NET Framework 4.0](http://go.microsoft.com/fwlink/?linkid=186916)
  + [Microsoft Visual Studio 2010](http://msdn.microsoft.com/vstudio/products/)
  + [SQL Server 2005 Express Edition (or later)](http://www.microsoft.com/sqlserver)
  + [Windows Azure Tools for Microsoft Visual Studio 1.6](http://www.microsoft.com/windowsazure/sdk/)

# Setup

* 1. In order to execute the exercises in this hands-on lab you need to set up your environment.
  2. Open a Windows Explorer window and browse to the lab’s **Source** folder.
  3. Double-click the **Setup.cmd** file in this folder to launch the setup process that will configure your environment and install the Visual Studio code snippets for this lab.
  4. If the User Account Control dialog is shown, confirm the action to proceed.
  5. **Note:** Make sure you have checked all the dependencies for this lab before running the setup.

# Using the Code Snippets

* 1. Throughout the lab document, you will be instructed to insert code blocks. For your convenience, most of that code is provided as Visual Studio Code Snippets, which you can use from within Visual Studio 2010 to avoid having to add it manually.
  2. If you are not familiar with the Visual Studio Code Snippets, and want to learn how to use them, you can refer to the **Setup.docx** document in the **Assets** folder of the training kit, which contains a section describing how to use them.

# Exercises

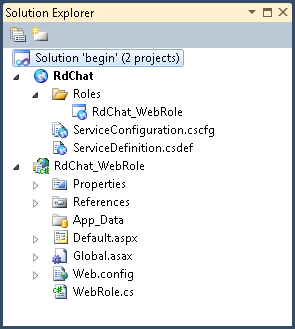
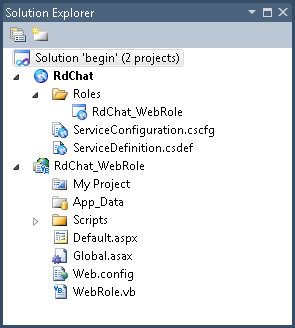
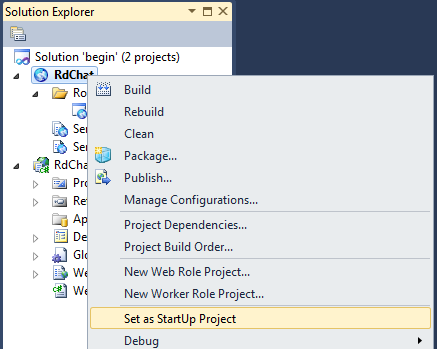
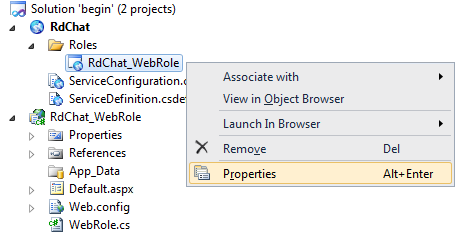
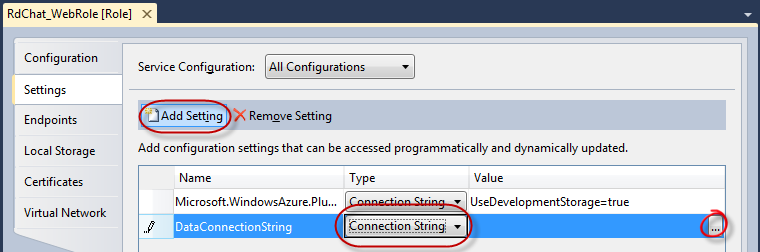
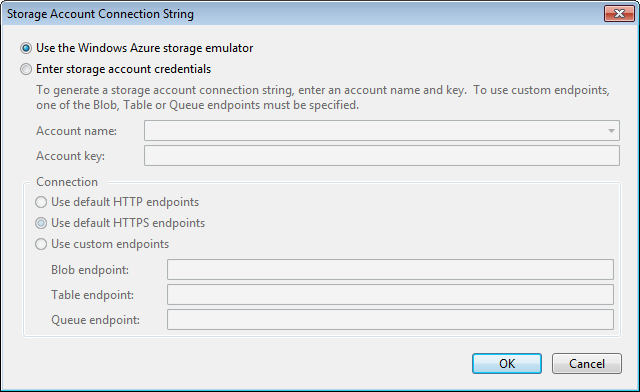
* 1. This hands-on lab includes the following exercises:
  2. Working with Tables
  3. Working with Blobs
  4. Working with Queues
  5. Working with Drives
  6. Estimated time to complete this lab: **90 minutes**.
  7. **Note:** When you first start Visual Studio, you must select one of the predefined settings collections. Every predefined collection is designed to match a particular development style and determines window layouts, editor behavior, IntelliSense code snippets, and dialog box options. The procedures in this lab describe the actions necessary to accomplish a given task in Visual Studio when using the **General Development Settings** collection. If you choose a different settings collection for your development environment, there may be differences in these procedures that you need to take into account.

Exercise 1: Working with Tables

* 1. In this exercise, you use the Windows Azure Table Service API to create a simple application that stores and retrieves data in structured storage. It consists of a simple chat Web application that can save, retrieve and display messages stored in a Windows Azure table.
  2. Windows Azure tables store data as collections of entities. Entities are similar to rows. An entity has a primary key and a set of properties. A property is a name/value pair, similar to a column.
  3. To access Windows Azure Table Service, you use a REST API that is compatible with [WCF Data Services](http://msdn.microsoft.com/en-us/library/cc668792.aspx) (formerly ADO.NET Data Services Framework). This exercise uses the [WCF Data Services Client Library](http://msdn.microsoft.com/en-us/library/cc668772.aspx) (formerly .NET Client Library) to read and write data to table service.

Task 1 – Configuring Storage Account Settings

In this task, you configure the settings required to make a connection to the Table Service.

* 1. Open Microsoft Visual Studio 2010 elevated **as Administrator** from **Start** | **All Programs** | **Microsoft Visual Studio 2010** by right-clicking **Microsoft Visual Studio 2010** and choosing **Run as administrator**.
  2. In the **File** menu, choose **Open** and then **Project/Solution**. In the **Open Project** dialog, browse to **\Source\Ex01-WorkingWithTables\Begin**, select the folder for your preferred language and open the **Begin.sln** file.
     1. 
     2. Figure 1
     3. Solution Explorer showing the Windows Azure Chat application (C#)
     4. 
     5. Figure 2
     6. Solution Explorer showing the Windows Azure Chat application (VB)
     7. **Note:** The solution contains a Windows Azure WebRole project.
  3. Make sure **RdChat** is the startup project by right-clicking it in **Solution Explorer** and selecting **Set as StartUp Project**.
     1. 
     2. Figure 3
     3. Setting the startup project
  4. Update the service definition to define the configuration settings required to access Windows Azure Table service. To do this, expand the **Roles** folder of the **RdChat** project in **Solution Explorer**, right-click **RdChat\_WebRole**, and then select **Properties**.
     1. 
     2. Figure 4
     3. Launching the service configuration editor
  5. Select the **Settings** tab, click **Add Setting** and create a new configuration setting named *DataConnectionString*. Set its type to **Connection String**, then click the button labeled with an ellipsis and configure the storage connection string to *Use storage emulator*.
     1. 
     2. Figure 5
     3. Creating a storage connection string
     4. 
     5. Figure 6
     6. Configuring a connection string to use Storage emulator
  6. Press **CTRL-S** to save your configuration changes.
     1. **Note:** The StorageClient library uses these settings to access Windows Azure Storage.
     2. **DataConnectionString**: This is the connection string to the Window Azure account, through which we can programmatically access data storage and other functionalities in Windows Azure. This connection string can point to a Windows Azure Account in the cloud as well as the local compute emulator.
     3. **Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString**: This is the connection string to Windows Azure server, same as *DataConnectionString*, however this one is dedicated for logging diagnostics.

Task 2 – Creating Classes to Model the Table Schema

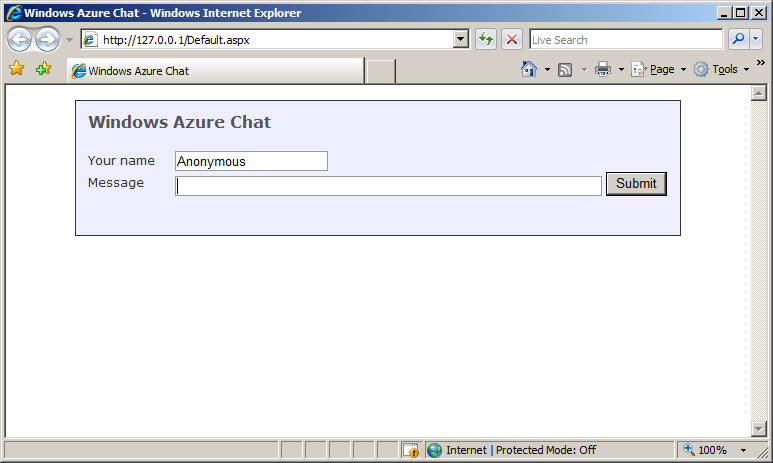
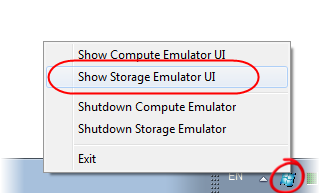
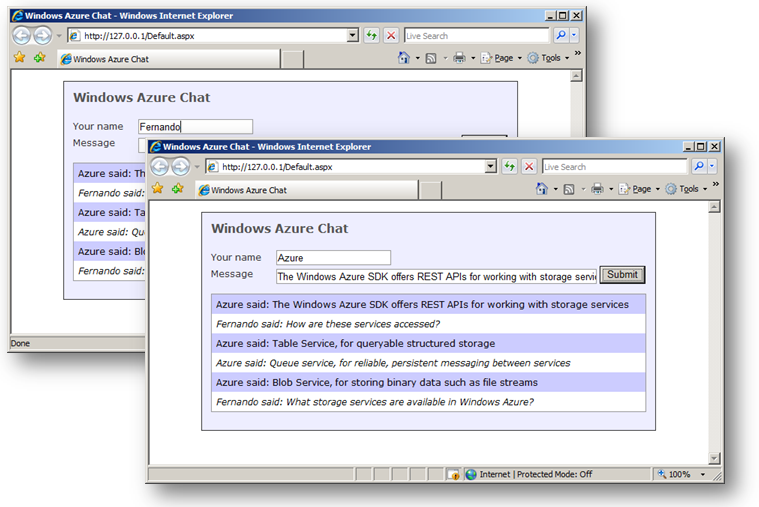
* 1. When working locally against the Storage Emulator service for Table Service, you use WCF Data Services client library.
  2. To use Windows Azure table service in .NET, you construct a class that models the desired schema. In addition to the properties required by your model, the class must include a **Timestamp**, a **PartitionKey** and a **RowKey** property and it must be decorated with a **DataServiceKey**(“*PartitionKey”*, “*RowKey”*) custom attribute. To simplify this, the **Microsoft.WindowsAzure.StorageClient** namespace includes a **TableServiceEntity** class that already defines the mandatory properties and attributes and can easily be derived from in your class.
  3. In this task, you create the model where the data is stored for the Chat application.
  4. Add a reference to the WCF Data Services client library to the Web role project. In **Solution Explorer**, right-click the **RdChat\_WebRole** project node and select **Add Reference**. In the **.NET** tab, select the **System.Data.Services.Client** assembly and click **OK**.
  5. Add a class to the Web role project to model the message table. To do this, in **Solution Explorer**, right-click the **RdChat\_WebRole** project node, point to **Add** and select **Class**. In the **Add New Item** dialog, set the **Name** to **Message.cs** (for Visual C# projects) or **Message.vb** (for Visual Basic projects) and then click **Add**.
  6. Update the declaration of the **Message** class to derive from the **Microsoft.WindowsAzure.StorageClient.TableServiceEntity** class.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-01-MessageClass-CS*)
     2. C#
     3. public class Message
     4. **: Microsoft.WindowsAzure.StorageClient.TableServiceEntity**
     5. {
     6. }
     7. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-01-MessageClass-VB*)
     8. Visual Basic
     9. Public Class Message
     10. **Inherits Microsoft.WindowsAzure.StorageClient.TableServiceEntity**
     11. End Class
     12. **Note:** The **TableServiceEntity** class is included as part of the **Microsoft.WindowsAzure.StorageClient** library. It defines the **PartititionKey**, **RowKey** and **TimeStamp** system properties required by every entity stored in a Windows Azure table.
     13. Together, the **PartitionKey** and **RowKey** define the **DataServiceKey** that uniquely identifies every entity within a table.
  7. Add a default constructor to the **Message** class that initializes its **PartitionKey** and **RowKey** properties.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-02-MessageConstructor-CS*)
     2. C#
     3. **public Message()**
     4. **{**
     5. **PartitionKey = "a";**
     6. **RowKey = string.Format("{0:10}\_{1}", DateTime.MaxValue.Ticks - DateTime.Now.Ticks, Guid.NewGuid());**
     7. **}**
     8. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-02-MessageConstructor-VB*)
     9. Visual Basic
     10. **Public Sub New()**
     11. **PartitionKey = "a"**
     12. **RowKey = String.Format("{0:10}\_{1}", (DateTime.MaxValue.Ticks - DateTime.Now.Ticks), Guid.NewGuid)**
     13. **End Sub**
     14. **Note:** For the purposes of this exercise, you assign a fixed value to the **PartitionKey** property. In a more realistic scenario, you would choose a value that ensures load balancing across storage nodes.
  8. Add two string properties to the **Message** class, **Name** and **Body**, to hold information about the chat message.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-03-TableSchemaProperties-CS*)
     2. C#
     3. **public string Name { get; set; }**
     4. **public string Body { get; set; }**
     5. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-03-TableSchemaProperties-VB*)
     6. Visual Basic
     7. **Public Property Body As String**
     8. **Public Property Name As String**
  9. Save the **Message.cs** / **Message.vb** file.
  10. Next, add a class to the Web role project to define the WCF Data Services **DataServiceContext** required to access the *Messages* table. To do this, in **Solution Explorer**, right-click the **RdChat\_WebRole** project node, point to **Add** and select **Class**. In the **Add New Item** dialog, set the **Name** to **MessageDataServiceContext.cs** (for Visual C# projects) or **MessageDataServiceContext.vb** (for Visual Basic projects) and then click **Add**.
  11. In the new class file, add the following using namespace directives.
      1. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-04-Namespace-CS*)
      2. C#
      3. **using Microsoft.WindowsAzure;**
      4. **using Microsoft.WindowsAzure.StorageClient;**
      5. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-04-Namespace-VB*)
      6. Visual Basic
      7. **Imports Microsoft.WindowsAzure**
      8. **Imports Microsoft.WindowsAzure.StorageClient**
  12. Replace the declaration of the new class to derive from the **TableServiceContext** class and include a default constructor to initialize the base class with the storage account information.
      1. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-05-MessageDataServiceContextClass-CS*)
      2. C#
      3. namespace RdChat\_WebRole
      4. {
      5. **public class MessageDataServiceContext**
      6. **: TableServiceContext**
      7. **{**
      8. **public MessageDataServiceContext(string baseAddress, StorageCredentials credentials)**
      9. **: base(baseAddress, credentials)**
      10. **{**
      11. **}**
      12. **}**
      13. }

(Code Snippet – *ExploringWindowsAzureStorage-Ex01-05-MessageDataServiceContextClass-VB*)

* + 1. Visual Basic
    2. **Public Class MessageDataServiceContext**
    3. **Inherits TableServiceContext**
    4. **Public Sub New(ByVal baseAddress As String, ByVal credentials As StorageCredentials)**
    5. **MyBase.New(baseAddress, credentials)**
    6. **End Sub**
    7. **End Class**
  1. Now, add a property to the **MessageDataServiceContext** class to return a data service query for the *Messages* table.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-06-MessagesProperty-CS*)
     2. C#
     3. public class MessageDataServiceContext
     4. : TableServiceContext
     5. {
     6. ...
     7. **public IQueryable<Message> Messages**
     8. **{**
     9. **get**
     10. **{**
     11. **return this.CreateQuery<Message>("Messages");**
     12. **}**
     13. **}**
     14. }
     15. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-06-MessagesProperty-VB*)
     16. Visual Basic
     17. Public Class MessageDataServiceContext
     18. Inherits TableServiceContext
     19. ...
     20. **Public ReadOnly Property Messages() As IQueryable(Of Message)**
     21. **Get**
     22. **Return MyBase.CreateQuery(Of Message)("Messages")**
     23. **End Get**
     24. **End Property**
     25. End Class
  2. Finally, add a method to the **MessageDataServiceContext** class to insert new messages into the table. You will use this method later when implementing the chat functionality.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-07-AddMessageMethod-CS*)
     2. C#
     3. public class MessageDataServiceContext
     4. : TableServiceContext
     5. {
     6. ...
     7. **public void AddMessage(string name, string body)**
     8. **{**
     9. **this.AddObject("Messages", new Message { Name = name, Body = body });**
     10. **this.SaveChanges();**
     11. **}**
     12. }
     13. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-07-AddMessageMethod-VB*)
     14. Visual Basic
     15. Public Class MessageDataServiceContext
     16. Inherits TableServiceContext
     17. ...
     18. **Public Sub AddMessage(ByVal name As String, ByVal body As String)**
     19. **Dim message As New Message**
     20. **message.Name = name**
     21. **message.Body = body**
     22. **MyBase.AddObject("Messages", message)**
     23. **MyBase.SaveChanges()**
     24. **End Sub**
     25. End Class
  3. In the **Build** menu, select **Build Solution**.

Task 3 – Creating the Chat User Interface

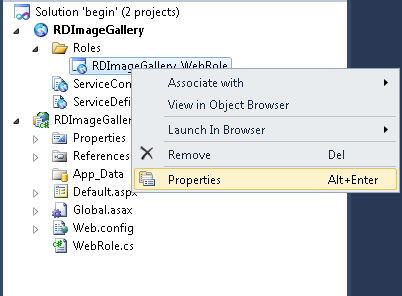
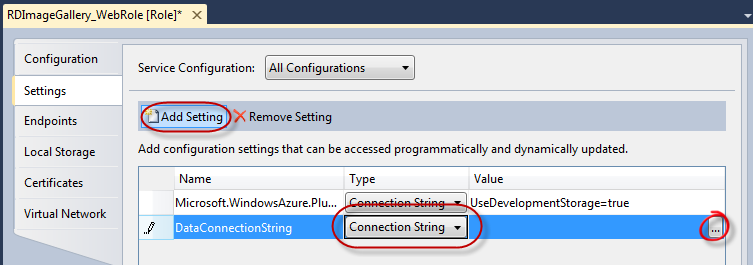
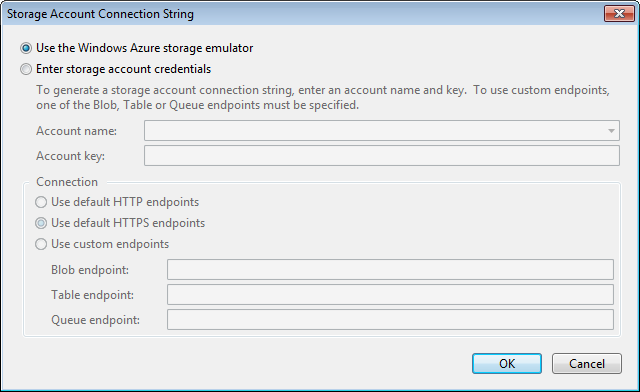
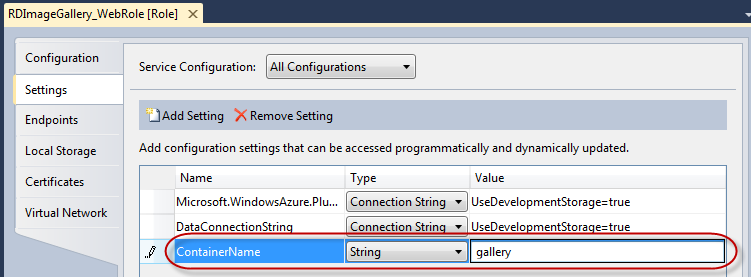
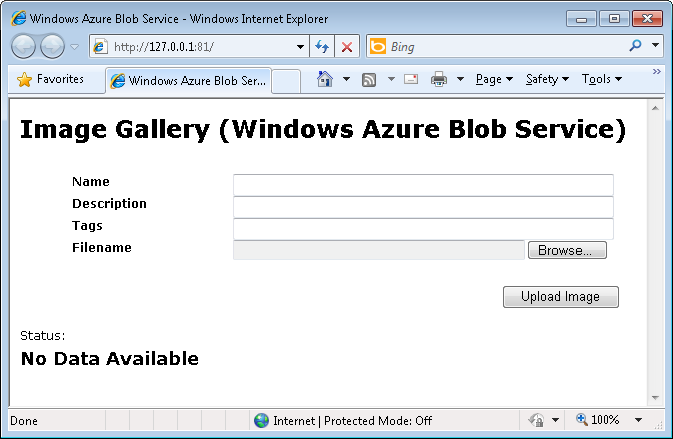
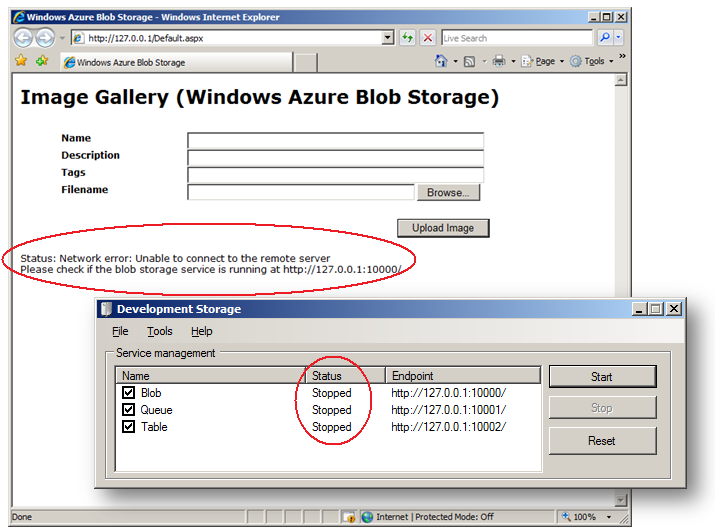
* 1. In this task, you add the code necessary to store messages in a Windows Azure table and display them on the Web page.
  2. Locate the **Application\_Start** method in **Global.asax.cs** / **Global.asax.vb** file and insert the following code (shown in **bold**) into it. This creates storage tables from **MessageDataServiceContext** that we created earlier.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-08-ApplicationStartMethod-CS*)
     2. C#
     3. protected void Application\_Start()
     4. {
     5. ...
     6. **/// Create data table from MessageDataServiceContext**
     7. **/// It is recommended the data tables should be only created once. It is typically done as a**
     8. **/// provisioning step and rarely in application code.**
     9. **var account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");**
     10. **// dynamically create the tables**
     11. **CloudTableClient.CreateTablesFromModel(typeof(MessageDataServiceContext),**
     12. **account.TableEndpoint.AbsoluteUri, account.Credentials);**
     13. }
     14. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-08- ApplicationStartMethod-VB*)
     15. Visual Basic
     16. Sub Application\_Start(ByVal sender As Object, ByVal e As EventArgs)
     17. ...
     18. **CloudStorageAccount.SetConfigurationSettingPublisher (Sub(configName, configSetter) configSetter(RoleEnvironment.GetConfigurationSettingValue(configName)))**
     19. **' Create data table from MessageDataServiceContext**
     20. **' It is a recommended practice to create data tables only once. It is typically done as a**
     21. **' provisioning step and rarely in application code.**
     22. **Dim account As CloudStorageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString")**
     23. **' dynamically create the tables**
     24. **CloudTableClient.CreateTablesFromModel(GetType(MessageDataServiceContext), account.TableEndpoint.AbsoluteUri, account.Credentials)**
     25. End Sub
     26. **Note:** The code shown above creates the required tables from the model defined by the **MessageDataServiceContext** class created earlier.
     27. Note that the recommendation is that data tables should only be created once. Typically, you would do this during a provisioning step and rarely in application code. The **Application\_Start** method in the **Global** class is a recommended place for this initialization logic.
     28. To retrieve and display messages, the method creates an instance of the **MessageDataServiceContext** class and initializes it from account information available in the service configuration file(**ServiceConfiguration.cscfg**). It binds the **Messages** property, which returns a data service query for th*e Messages* table, to a **ListView** control on the page for display.
     29. Objects of type **CloudStorageAccount** represent a storage account, which contains the settings required to make a connection to the Storage Service. Associated with a storage account are the account name, the URI of the account and a shared key, which the **CloudTableClient** helper class uses for its initialization. These settings are obtained from **ServiceConfiguration.cscfg**.
  3. Make sure the following namespace directives exist at the top of the **Global.asax.cs** / **Global.asax.vb** code file. They are for the utility storage classes and for the **ServiceRuntime** classes.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-09-GlobalNamespace-CS*)
     2. C#
     3. **using Microsoft.WindowsAzure;**
     4. **using Microsoft.WindowsAzure.ServiceRuntime;**
     5. **using Microsoft.WindowsAzure.StorageClient;**
     6. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-09- GlobalNamespace -VB*)
     7. Visual Basic
     8. **Imports Microsoft.WindowsAzure**
     9. **Imports Microsoft.WindowsAzure.ServiceRuntime**
     10. **Imports Microsoft.WindowsAzure.StorageClient**
  4. Expand the **RDChat\_WebRole** node in **Solution Explorer**, then right-click **Default.aspx** and select **View Code** to open the code-behind file for the Web page that contains the UI for the chat application.
     1. Make sure the following namespace directives are included in the **Default.aspx.cs** / **Default.aspx.vb** code-behind file.
     2. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-10-Namespace-CS*)
     3. C#
     4. **using System.Data.Services.Client;**
     5. **using Microsoft.WindowsAzure;**
     6. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-10-Namespace-VB*)
     7. Visual Basic
     8. **Imports System.Data.Services.Client**
     9. **Imports Microsoft.WindowsAzure**
  5. Locate the **SubmitButton\_Click** event handler in **Default.aspx.cs** / **Default.aspx.vb** and insert the following code (shown in **bold**) into the method body to save messages entered by the user to the Table Service, then data bind messages from Table Service to the page. The method uses the **AddMessage** method, which you created earlier in the lab, to insert a new **Message** entity into the table.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-11-SubmitButtonClick-CS*)
     2. C#
     3. protected void SubmitButton\_Click(object sender, EventArgs e)
     4. {
     5. **var statusMessage = String.Empty;**
     6. **try**
     7. **{**
     8. **var account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");**
     9. **var context = new MessageDataServiceContext(account.TableEndpoint.ToString(), account.Credentials);**
     10. **context.AddMessage(HttpUtility.HtmlEncode(this.nameBox.Text), HttpUtility.HtmlEncode(this.messageBox.Text));**
     11. **this.messageList.DataSource = context.Messages;**
     12. **this.messageList.DataBind();**
     13. **}**
     14. **catch (DataServiceRequestException ex)**
     15. **{**
     16. **statusMessage = "Unable to connect to the table storage server. Please check that the service is running.<br>"**
     17. **+ ex.Message;**
     18. **}**
     19. **status.Text = statusMessage;**
     20. }
     21. (Code Snippet – *ExploringWindowsAzureStorage-Ex01-11-SubmitButtonClick-VB*)
     22. Visual Basic
     23. Protected Sub SubmitButton\_Click(ByVal sender As Object, ByVal e As EventArgs)
     24. **Dim statusMessage As String = String.Empty**
     25. **Try**
     26. **Dim account As CloudStorageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString")**
     27. **Dim context As New MessageDataServiceContext(account.TableEndpoint.ToString, account.Credentials)**
     28. **context.AddMessage(HttpUtility.HtmlEncode(Me.nameBox.Text), HttpUtility.HtmlEncode(Me.messageBox.Text))**
     29. **Me.messageList.DataSource = context.Messages**
     30. **Me.messageList.DataBind()**
     31. **Catch ex As DataServiceRequestException**
     32. **statusMessage = ("Unable to connect to the table storage server. Please check that the service is running.<br>" & ex.Message)**
     33. **End Try**
     34. **Me.status.Text = statusMessage**
     35. End Sub
  6. Save the all files and select **Build Solution** from the **Build** menu.

1. Verification
   1. To test your service running in the Compute Emulator:
   2. In Visual Studio, press **F5** to build and run the application. The compute emulator starts and a new deployment containing the **RdChat** Web Role initializes. A browser window opens to display the Windows Azure Chat Web page.
      1. 
      2. Figure 7
      3. Application after connecting successfully to the storage emulator table server
      4. When you start the program in the debugger, Visual Studio automatically starts Storage Emulator. If the Chat application is unable to access the table service server, you will see an error as shown on the page. To examine the status of the service, right-click the icon in the system tray (it looks like a server) and select **Show Storage Emulator UI**.
      5. 
      6. Figure 8
      7. Viewing the status of storage emulator
   3. Now, test the Chat application by entering a few messages. Type your name and the text of your message and click **Submit**.
   4. In Internet Explorer, press **Ctrl + N** to open a second browser window. Enter a different name and type a few more messages. Notice how the chat box updates the conversation after sending a new message.
      1. 
      2. Figure 9
      3. Using the Windows Azure chat application

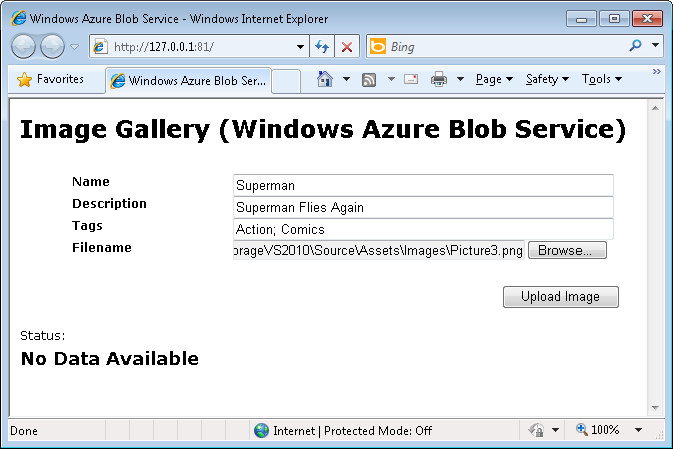
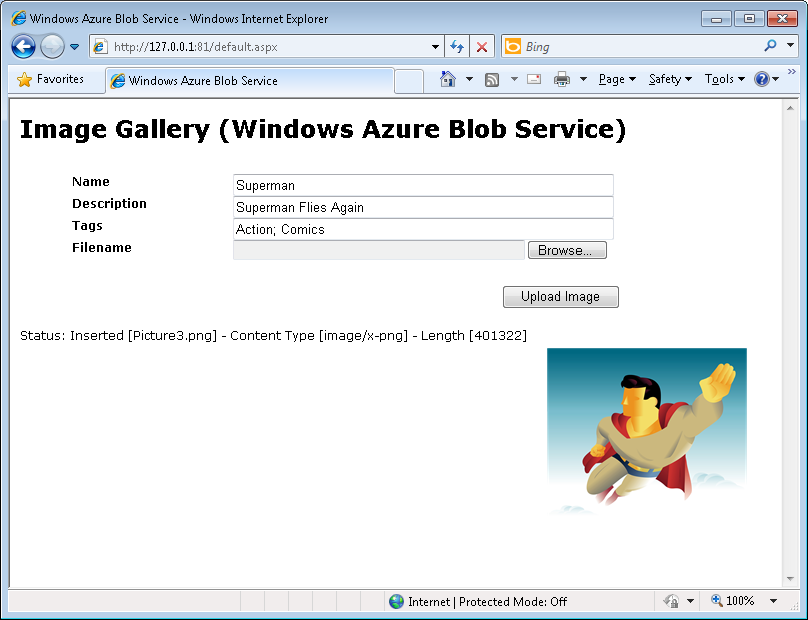
Exercise 2: Working with Blobs

* 1. In this exercise, you will use the Windows Azure Blob Service API to create an application that saves and retrieves image data stored as blobs in Windows Azure storage. It consists of a simple image gallery Web site that can display, upload and remove images in Windows Azure storage, and allows you to enter and display related metadata. The application uses a single container to store its image content as blobs.
  2. When you create a blob in Windows Azure, you associate a content type that specifies the format in which the API returns it and allows you to retrieve an image directly from the URL of the corresponding blob.

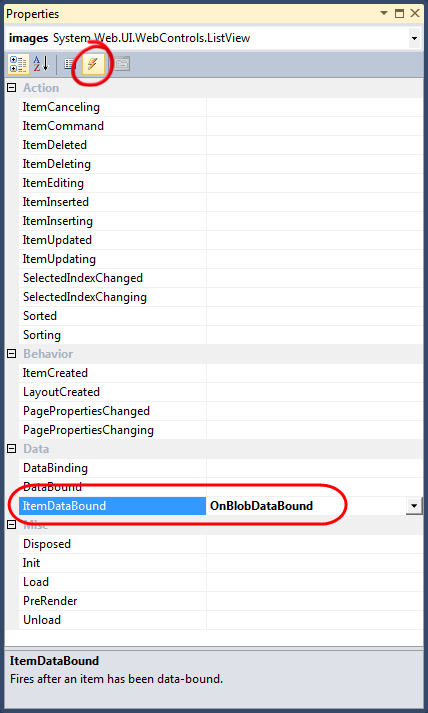
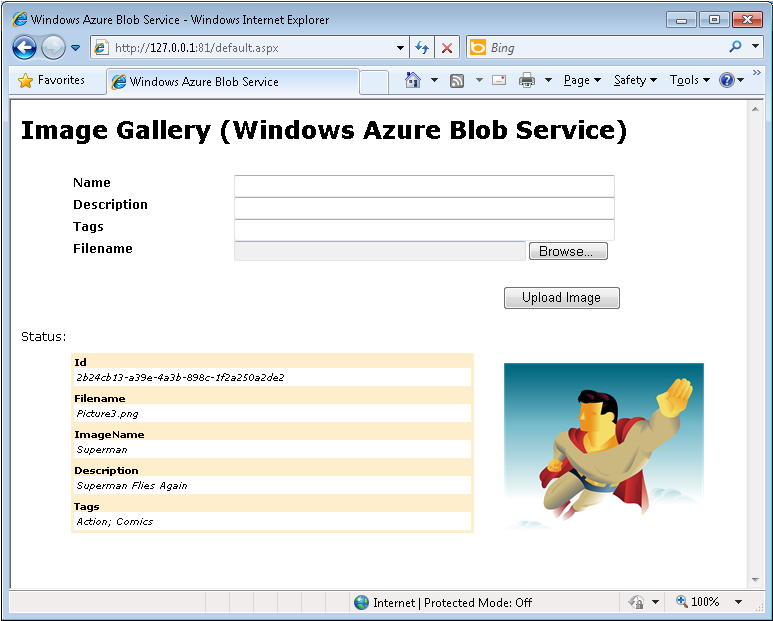
Task 1 – Retrieving Blob Data from Storage

* 1. In this task, you will create an image gallery web page to display images retrieved from Windows Azure storage. The provided solution consists of a web site project with a single page that contains the elements required to display images and enter metadata. You will add the necessary functionality by editing the code-behind file.
  2. Open Microsoft Visual Studio 2010 elevated **as Administrator** from **Start** | **All Programs** | **Microsoft Visual Studio 2010** by right-clicking **Microsoft Visual Studio 2010** and choosing **Run as administrator**.
  3. In the **File** menu, choose **Open** and then **Project/Solution**. In the **Open Project** dialog, browse to **\Source\Ex02-WorkingWithBlobs\begin**, select the folder for your preferred language and open the **begin.sln** file.
  4. Update the service definition to define the configuration settings required to access Windows Azure Table service. To do this, expand the **Roles** folder of the **RDImageGallery** project in **Solution Explorer**, right-click **RDImageGallery\_WebRole**, and then select **Properties**.
     1. 
     2. Figure 10
     3. Setting role configuration settings
  5. In the **Settings** tab, click **Add Setting** and create a **ConnectionString** type named *DataConnectionString*. Click the button labeled with an ellipsis and set the connection string to **Use storage emulator**.
     1. 
     2. Figure 11
     3. Configuring a storage connection string
     4. 
     5. Figure 12
     6. Storage connection string dialog
  6. Add another setting named *ContainerName* and set its value to ‘*gallery’*.
     1. 
     2. Figure 13
     3. Creating a setting for the container name
     4. **Note:**  The container name must be a valid Domain Name System (DNS) name, conforming to the following naming rules:
     5. - Must start with a letter or number, and can contain only letters, numbers, and dash(-) characters.
     6. - All letters must be lowercase.
     7. - Must be from 3 to 63 characters long.
     8. - A name cannot contain a dash next to a period.
  7. Expand the **RDImageGallery\_WebRole** node in **Solution Explorer**, then right-click **Default.aspx** and select **View Code** to open the code-behind file for the user interface of the image gallery. In the next steps, you will modify this file to add some of the required functionality.
  8. Make sure the following namespace directives exist at the top of the code file. They are for the utility storage classes and for the **ServiceRuntime** classes.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-01-Namespace-CS*)
     2. C#
     3. **using Microsoft.WindowsAzure;**
     4. **using Microsoft.WindowsAzure.ServiceRuntime;**
     5. **using Microsoft.WindowsAzure.StorageClient;**
     6. **using Microsoft.WindowsAzure.StorageClient.Protocol;**
     7. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-01-Namespace-VB*)
     8. Visual Basic
     9. **Imports Microsoft.WindowsAzure**
     10. **Imports Microsoft.WindowsAzure.ServiceRuntime**
     11. **Imports Microsoft.WindowsAzure.StorageClient**
     12. **Imports Microsoft.WindowsAzure.StorageClient.Protocol**
  9. For this lab, you need to store the blobs in a public container, so they are visible on the web to normal, anonymous users. In this step, you will ensure that the container specified in **ServiceConfiguration.cscfg** exists. To do this, add the following method at the bottom of the **\_Default** class.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-02-EnsureContainerExistsMethod-CS*)
     2. C#
     3. public partial class \_Default : System.Web.UI.Page
     4. {
     5. ...
     6. **private void EnsureContainerExists()**
     7. **{**
     8. **var container = GetContainer();**
     9. **container.CreateIfNotExist();**
     10. **var permissions = container.GetPermissions();**
     11. **permissions.PublicAccess = BlobContainerPublicAccessType.Container;**
     12. **container.SetPermissions(permissions);**
     13. **}**
     14. }
     15. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-02-EnsureContainerExistsMethod-VB*)
     16. Visual Basic
     17. Partial Public Class \_Default
     18. Inherits System.Web.UI.Page
     19. ...
     20. **Private Sub EnsureContainerExists()**
     21. **Dim container = GetContainer()**
     22. **container.CreateIfNotExist()**
     23. **Dim permissions = container.GetPermissions()**
     24. **permissions.PublicAccess = BlobContainerPublicAccessType.Container**
     25. **container.SetPermissions(permissions)**
     26. **End Sub**
     27. End Class
  10. Next, you create a utility method to retrieve a reference to the container created by the code in the previous step. This method will be called in almost all operations since the container is involved with all blob operations. Add a method to create the container at the bottom of the **\_Default** class. This method uses the configuration settings you entered in earlier steps.
      1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-03-GetContainerMethod-CS*)
      2. C#
      3. public partial class \_Default : System.Web.UI.Page
      4. {
      5. ...
      6. **private CloudBlobContainer GetContainer()**
      7. **{**
      8. **// Get a handle on account, create a blob service client and get container proxy**
      9. **var account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");**
      10. **var client = account.CreateCloudBlobClient();**
      11. **return client.GetContainerReference(RoleEnvironment.GetConfigurationSettingValue("ContainerName"));**
      12. **}**
      13. }
      14. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-03-GetContainerMethod-VB*)
      15. Visual Basic
      16. Partial Public Class \_Default
      17. Inherits System.Web.UI.Page
      18. ...
      19. **Private Function GetContainer() As CloudBlobContainer**
      20. **' Get a handle on account, create a blob service client and get container proxy**
      21. **Dim account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString")**
      22. **Dim client = account.CreateCloudBlobClient()**
      23. **Return client.GetContainerReference(RoleEnvironment.GetConfigurationSettingValue("ContainerName"))**
      24. **End Function**
      25. End Class
  11. Insert the following code (shown in **bold**) in the **Page\_Load** method to initialize the container and refresh the **asp:ListView** control on the page that displays the images retrieved from storage.
      1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-04-PageLoadMethod-CS*)
      2. C#
      3. public partial class \_Default : System.Web.UI.Page
      4. {
      5. ...
      6. protected void Page\_Load(object sender, EventArgs e)
      7. {
      8. **try**
      9. **{**
      10. **if (!IsPostBack)**
      11. **{**
      12. **this.EnsureContainerExists();**
      13. **}**
      14. **this.RefreshGallery();**
      15. **}**
      16. **catch (System.Net.WebException we)**
      17. **{**
      18. **status.Text = "Network error: " + we.Message;**
      19. **if (we.Status == System.Net.WebExceptionStatus.ConnectFailure)**
      20. **{**
      21. **status.Text += "<br />Please check if the blob service is running at " +**
      22. **ConfigurationManager.AppSettings["storageEndpoint"];**
      23. **}**
      24. **}**
      25. **catch (StorageException se)**
      26. **{**
      27. **Console.WriteLine("Storage service error: " + se.Message);**
      28. **}**
      29. }
      30. ...
      31. }
      32. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-04-PageLoadMethod-VB*)
      33. Visual Basic
      34. Partial Public Class \_Default
      35. Inherits System.Web.UI.Page
      36. ...
      37. Protected Sub Page\_Load(ByVal sender As Object, ByVal e As EventArgs)
      38. **Try**
      39. **If Not IsPostBack Then**
      40. **Me.EnsureContainerExists()**
      41. **End If**
      42. **Me.RefreshGallery()**
      43. **Catch we As System.Net.WebException**
      44. **status.Text = "Network error: " & we.Message**
      45. **If we.Status = System.Net.WebExceptionStatus.ConnectFailure Then**
      46. **status.Text += "<br />Please check if the blob service is running at " & ConfigurationManager.AppSettings("storageEndpoint")**
      47. **End If**
      48. **Catch se As StorageException**
      49. **Console.WriteLine("Storage service error: " & se.Message)**
      50. **End Try**
      51. End Sub
      52. ...
      53. End Class
  12. Add the following method at the bottom of the **\_Default** class to bind the *images* control to the list of blobs available in the image gallery container. The code uses the **ListBlobs** method in the **CloudBlobContainer** object to retrieve a collection of **IListBlobItem** objects that contain information about each of the blobs. The *images* **asp:ListView** control in the page binds to these objects to display their value.
      1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-05-RefreshGalleryMethod-CS*)
      2. C#
      3. public partial class \_Default : System.Web.UI.Page
      4. {
      5. ...
      6. **private void RefreshGallery()**
      7. **{**
      8. **images.DataSource =**
      9. **this.GetContainer().ListBlobs(new BlobRequestOptions()**
      10. **{**
      11. **UseFlatBlobListing = true,**
      12. **BlobListingDetails = BlobListingDetails.All**
      13. **});**
      14. **images.DataBind();**
      15. **}**
      16. }
      17. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-05-RefreshGalleryMethod-VB*)
      18. Visual Basic
      19. Partial Public Class \_Default
      20. Inherits System.Web.UI.Page
      21. ...
      22. **Private Sub RefreshGallery()**
      23. **Dim options As BlobRequestOptions = New BlobRequestOptions() With**
      24. **{.BlobListingDetails = BlobListingDetails.All,**
      25. **.UseFlatBlobListing = True}**
      26. **images.DataSource = Me.GetContainer().ListBlobs(options)**
      27. **images.DataBind()**
      28. **End Sub**
      29. End Class
  13. Press **F5** to build and run the application. A browser window launches and displays the contents of the image gallery. Note that at this point, the container is empty and the list view displays a “No Data Available” message. In the next task, you will implement the functionality required to store images as blobs in Windows Azure storage.
      1. 
      2. Figure 14
      3. The image gallery application displaying an empty container
      4. **Note:** If you have not configured the storage settings correctly or if the storage service is not running, an error similar to the one shown below is displayed.
      5. 
      6. Figure 15
      7. An error caused by an invalid Windows Azure storage configuration or a service problem
  14. Press **Shift+F5** in Visual Studio to stop debugging and delete the deployment from the compute emulator.

Task 2 – Uploading Blob Data to Storage

* 1. In this task, you add functionality to the image gallery Web page to enter metadata and upload image files to Windows Azure storage. The page contains text controls that you can use to enter descriptive metadata for the selected image. An **asp:FileUpload** control on the page retrieves images from disk and posts them to the page, where they are stored in blob storage.
  2. Open the ***Default.aspx.cs*** */* ***Default.aspx.vb*** file in the Visual Studio text editor. To do this, right-click the Default.aspx file in Solution Explorer and select **View Code**.
  3. Add a method at the bottom of the page to save images and their metadata as blobs in Windows Azure storage. The method uses the **GetBlobReference** method in the **CloudBlobContainer** object to create a blob from the image data array and the metadata properties.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-06-SaveImageMethod-CS*)
     2. C#
     3. public partial class \_Default : System.Web.UI.Page
     4. {
     5. ...
     6. **private void SaveImage(string id, string name, string description, string tags, string fileName, string contentType, byte[] data)**
     7. **{**
     8. **// Create a blob in container and upload image bytes to it**
     9. **var blob = this.GetContainer().GetBlobReference(name);**
     10. **blob.Properties.ContentType = contentType;**
     11. **// Create some metadata for this image**
     12. **var metadata = new NameValueCollection();**
     13. **metadata["Id"] = id;**
     14. **metadata["Filename"] = fileName;**
     15. **metadata["ImageName"] = String.IsNullOrEmpty(name) ? "unknown" : name;**
     16. **metadata["Description"] = String.IsNullOrEmpty(description) ? "unknown" : description;**
     17. **metadata["Tags"] = String.IsNullOrEmpty(tags) ? "unknown" : tags;**
     18. **// Add and commit metadata to blob**
     19. **blob.Metadata.Add(metadata);**
     20. **blob.UploadByteArray(data);**
     21. **}**
     22. }
     23. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-06-SaveImageMethod-VB*)
     24. Visual Basic
     25. Partial Public Class \_Default
     26. Inherits System.Web.UI.Page
     27. ...
     28. **Private Sub SaveImage(ByVal id As String, ByVal name As String, ByVal description As String, ByVal tags As String, ByVal fileName As String, ByVal contentType As String, \_**
     29. **ByVal data As Byte())**
     30. **' Create a blob in container and upload image bytes to it**
     31. **Dim blob = Me.GetContainer().GetBlobReference(name)**
     32. **blob.Properties.ContentType = contentType**
     33. **' Create some metadata for this image**
     34. **Dim metadata = New NameValueCollection()**
     35. **metadata("Id") = id**
     36. **metadata("Filename") = fileName**
     37. **metadata("ImageName") = If([String].IsNullOrEmpty(name), "unknown", name)**
     38. **metadata("Description") = If([String].IsNullOrEmpty(description), "unknown", description)**
     39. **metadata("Tags") = If([String].IsNullOrEmpty(tags), "unknown", tags)**
     40. **' Add and commit metadata to blob**
     41. **blob.Metadata.Add(metadata)**
     42. **blob.UploadByteArray(data)**
     43. **End Sub**
     44. End Class
  4. Complete the code in the event handler for the **Upload Image** button by inserting the code (shown in **bold** below) to **upload\_Click** method.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-07-UploadClickMethod-CS*)
     2. C#
     3. public partial class \_Default : System.Web.UI.Page
     4. {
     5. ...
     6. protected void upload\_Click(object sender, EventArgs e)
     7. {
     8. **if (imageFile.HasFile)**
     9. **{**
     10. **status.Text = "Inserted [" + imageFile.FileName + "] - Content Type [" + imageFile.PostedFile.ContentType + "] - Length [" + imageFile.PostedFile.ContentLength + "]";**
     11. **this.SaveImage(**
     12. **Guid.NewGuid().ToString(),**
     13. **imageName.Text,**
     14. **imageDescription.Text,**
     15. **imageTags.Text,**
     16. **imageFile.FileName,**
     17. **imageFile.PostedFile.ContentType,**
     18. **imageFile.FileBytes**
     19. **);**
     20. **RefreshGallery();**
     21. **}**
     22. **else**
     23. **status.Text = "No image file";**
     24. }
     25. ...
     26. }
     27. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-07-UploadClickMethod-VB*)
     28. Visual Basic
     29. Protected Sub upload\_Click(ByVal sender As Object, ByVal e As EventArgs)
     30. **If imageFile.HasFile Then**
     31. **status.Text = (("Inserted [" & imageFile.FileName & "] - Content Type [") + imageFile.PostedFile.ContentType & "] - Length [") + imageFile.PostedFile.ContentLength.ToString() + "]"**
     32. **Me.SaveImage(Guid.NewGuid().ToString(), imageName.Text, imageDescription.Text, imageTags.Text, imageFile.FileName, imageFile.PostedFile.ContentType, \_**
     33. **imageFile.FileBytes)**
     34. **RefreshGallery()**
     35. **Else**
     36. **status.Text = "No image file"**
     37. **End If**
     38. End Sub
     39. The code retrieves metadata from the text controls and from properties in the **asp:FileUpload** control on the page, which include the content type of the posted file, its file name, and the array of bytes containing the image data. It then calls the **SaveImage** method to store the image and its metadata to Windows Azure storage.
  5. Press **F5** to build and run the application and open the image gallery page in a browser window.
  6. Enter metadata in the **Name**, **Description** and **Tags** text boxes. To select the image file, click **Browse**, navigate to **\Source\Assets\Images**, and select one of the available images.
     1. 
     2. Figure 16
     3. Entering metadata to store with the image in blob storage
  7. Click **Upload Image** to post the image to the Web application. The page refreshes and the newly added image displays in the list view. A status message shows the file name, content type and size of the uploaded file. Note that at this point, no metadata is displayed for the image. In the next task, you will implement the functionality required to retrieve and display metadata for blobs stored in Windows Azure.
     1. 
     2. Figure 17
     3. The image gallery showing the uploaded image
  8. In Visual Studio, press **Shift+F5** to stop debugging and delete the deployment from the Compute Emulator.

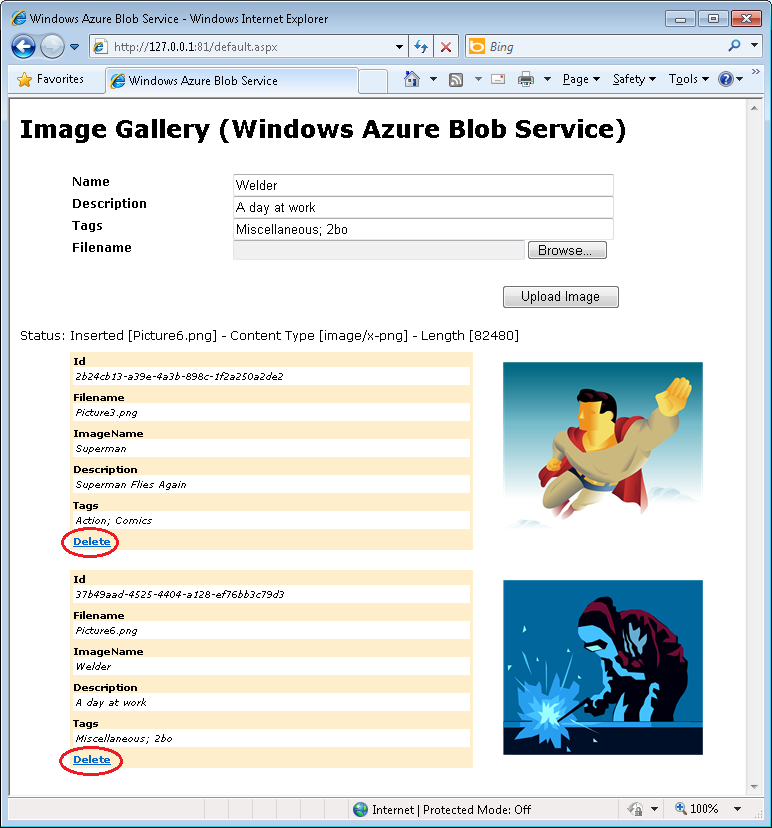
Task 3 – Retrieving Metadata for Blobs in Storage

* 1. Blobs can have metadata attached to them. Metadata headers can be set on a request that creates a new container or blob resource, or on a request that explicitly creates a property on an existing resource. In this task, you will add functionality to the image gallery page to retrieve and display metadata associated with images stored in a Windows Azure container.
  2. Add an event handler to retrieve metadata for each blob displayed in the list view control that displays images. To do this, go to **Default.aspx**, right-click **View Designer**, select the ***images*** **ListView** control, and in the **Properties** Window (you may need to make it visible by right-clicking the control and choosing **Properties**) click the **Events** button. Locate the **ItemDataBound** event in the *Data* category, type **OnBlobDataBound** and press <**Enter**>. Alternatively, you may edit the ASP.NET markup directly to insert the required event handler.
     1. 
     2. Figure 18
     3. Configuring the event handler to display metadata
  3. In the code-behind file, locate the **OnBlobDataBound** method and insert the following code (shown in **bold**) that retrieves the properties for each blob bound to the list view and creates a collection that contains name / value pairs for each metadata item found. The collection is then used as a data source for an **asp:Repeater** control that displays metadata for each image.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-08-OnBlobDataBoundMethod-CS*)
     2. C#
     3. protected void OnBlobDataBound(object sender, ListViewItemEventArgs e)
     4. {
     5. **if (e.Item.ItemType == ListViewItemType.DataItem)**
     6. **{**
     7. **var metadataRepeater = e.Item.FindControl("blobMetadata") as Repeater;**
     8. **var blob = ((ListViewDataItem)(e.Item)).DataItem as CloudBlob;**
     10. **// If this blob is a snapshot, rename button to "Delete Snapshot"**
     11. **if (blob != null)**
     12. **{**
     13. **if (blob.SnapshotTime.HasValue)**
     14. **{**
     15. **var delBtn = e.Item.FindControl("deleteBlob") as LinkButton;**
     17. **if (delBtn != null)**
     18. **{**
     19. **delBtn.Text = "Delete Snapshot";**
     20. **var snapshotRequest = BlobRequest.Get(new Uri(delBtn.CommandArgument), 0, blob.SnapshotTime.Value, null);**
     21. **delBtn.CommandArgument = snapshotRequest.RequestUri.AbsoluteUri;**
     22. **}**
     24. **var snapshotBtn = e.Item.FindControl("SnapshotBlob") as LinkButton;**
     25. **if (snapshotBtn != null) snapshotBtn.Visible = false;**
     26. **}**
     28. **if (metadataRepeater != null)**
     29. **{**
     30. **//bind to metadata**
     31. **metadataRepeater.DataSource = from key in blob.Metadata.AllKeys**
     32. **select new**
     33. **{**
     34. **Name = key,**
     35. **Value = blob.Metadata[key]**
     36. **};**
     37. **metadataRepeater.DataBind();**
     38. **}**
     39. **}**
     40. **}**
     41. }
     42. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-08-OnBlobDataBoundMethod-VB*)
     43. Visual Basic
     44. Protected Sub OnBlobDataBound(ByVal sender As Object, ByVal e As ListViewItemEventArgs) Handles images.ItemDataBound
     45. **If e.Item.ItemType = ListViewItemType.DataItem Then**
     46. **Dim metadataRepeater = TryCast(e.Item.FindControl("blobMetadata"), Repeater)**
     47. **Dim blob = TryCast(DirectCast((e.Item), ListViewDataItem).DataItem, CloudBlob)**
     48. **' If this blob is a snapshot, rename button to "Delete Snapshot"**
     49. **If blob IsNot Nothing Then**
     50. **If blob.SnapshotTime.HasValue Then**
     51. **Dim delBtn = TryCast(e.Item.FindControl("deleteBlob"), LinkButton)**
     52. **If delBtn IsNot Nothing Then**
     53. **delBtn.Text = "Delete Snapshot"**
     54. **Dim snapshotRequest = BlobRequest.Get(New Uri(delBtn.CommandArgument), 0, blob.SnapshotTime.Value, Nothing)**
     55. **delBtn.CommandArgument = snapshotRequest.RequestUri.AbsoluteUri**
     56. **End If**
     57. **Dim snapshotBtn = TryCast(e.Item.FindControl("Snapshot"), LinkButton)**
     58. **If snapshotBtn IsNot Nothing Then**
     59. **snapshotBtn.Visible = False**
     60. **End If**
     61. **End If**
     62. **If metadataRepeater IsNot Nothing Then**
     63. **'bind to metadata**
     64. **metadataRepeater.DataSource = From key In blob.Metadata.AllKeys \_**
     65. **Select New With {.Name = key, .Value = blob.Metadata.Get(key)}**
     66. **metadataRepeater.DataBind()**
     67. **End If**
     68. **End If**
     69. **End If**
     70. End Sub
  4. Press **F5** to build and run the application. Note that the list view now displays the metadata for the image that was uploaded in the previous exercise.
     1. 
     2. Figure 19
     3. The image gallery showing metadata retrieved from blob storage
  5. Press **Shift+F5** to stop debugging and delete the deployment from the compute emulator.

Task 4 – Deleting Blobs from Storage

* 1. In this task, you will add functionality to the image gallery Web page to delete blobs containing image data from Windows Azure storage.
  2. Update the image list view to add an **asp:LinkButton** control that is used to delete images from the gallery container. To do this, right-click **Default.aspx**, and select **View Markup** and locate the **ItemTemplate** for the images **asp:ListView** control. Uncomment the ASP.NET markup located immediately following the **blobMetadata** repeater control (shown **bolded** below).
     1. HTML
     2. ...
     3. <div class="item">
     4. <ul style="width:40em;float:left;clear:left" >
     5. <asp:Repeater ID="blobMetadata" runat="server">
     6. <ItemTemplate>
     7. <li><%# Eval("Name") %><span><%# Eval("Value") %></span></li>
     8. </ItemTemplate>
     9. </asp:Repeater>
     10. **<li>**
     11. **<asp:LinkButton ID="deleteBlob"**
     12. **OnClientClick="return confirm('Delete image?');"**
     13. **CommandName="Delete"**
     14. **CommandArgument='<%# Eval("Uri")%>'**
     15. **runat="server" Text="Delete" oncommand="OnDeleteImage" />**
     16. ...
     17. **</li>**
     18. </ul>
     19. <img src="<%# Eval("Uri") %>" alt="<%# Eval("Uri") %>" style="float:left"/>
     20. </div>
     21. ...
  3. Add code (shown in **bold**) to **Default.aspx.cs** / **Default.aspx.vb** to implement the command handler for the *deleteBlob* **asp:LinkButton** control. The code verifies if a blob exists in storage and deletes it.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-09-OnDeleteImageMethod-CS*)
     2. C#
     3. public partial class \_Default : System.Web.UI.Page
     4. {
     5. ...
     6. protected void OnDeleteImage(object sender, CommandEventArgs e)
     7. {
     8. **try**
     9. **{**
     10. **if (e.CommandName == "Delete")**
     11. **{**
     12. **var blobUri = (string)e.CommandArgument;**
     13. **var blob = this.GetContainer().GetBlobReference(blobUri);**
     14. **blob.DeleteIfExists();**
     15. **}**
     16. **}**
     17. **catch (StorageClientException se)**
     18. **{**
     19. **status.Text = "Storage client error: " + se.Message;**
     20. **}**
     21. **catch (Exception) { }**
     22. **RefreshGallery();**
     23. }

...

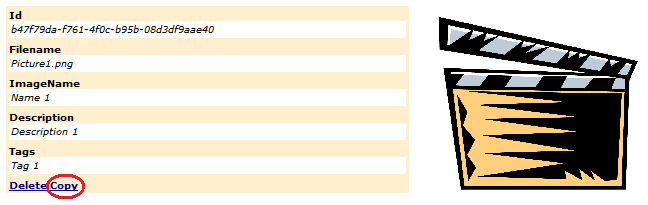
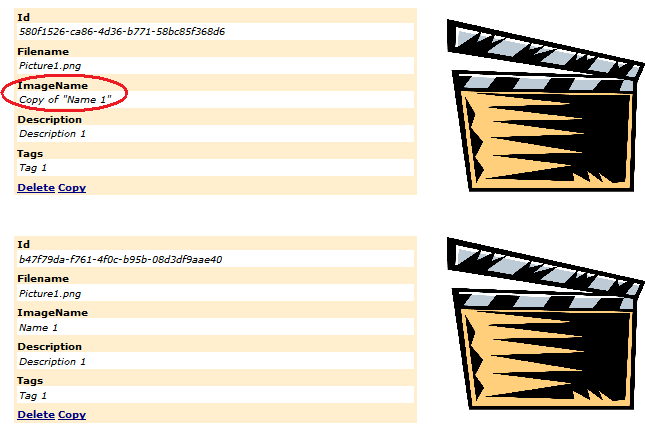
* + 1. }
    2. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-09-OnDeleteImageMethod-VB*)
    3. Visual Basic
    4. Partial Public Class \_Default
    5. Inherits System.Web.UI.Page
    6. ...
    7. Protected Sub OnDeleteImage(ByVal sender As Object, ByVal e As CommandEventArgs)
    8. **Try**
    9. **If e.CommandName = "Delete" Then**
    10. **Dim blobUri = DirectCast(e.CommandArgument, String)**
    11. **Dim blob = Me.GetContainer().GetBlobReference(blobUri)**
    12. **blob.DeleteIfExists()**
    13. **End If**
    14. **Catch se As StorageClientException**
    15. **status.Text = "Storage client error: " & se.Message**
    16. **Catch generatedExceptionName As Exception**
    17. **End Try**
    18. **RefreshGallery()**
    19. End Sub
    20. ...
    21. End Class
  1. Press **F5** to build and run the application.
  2. Upload a few more images from **Assets\Images** in the **Source** folder of the Lab and click **Delete** on any of the images displayed to remove the corresponding blob from storage.
     1. 
     2. Figure 20
     3. Adding and deleting image blobs from storage
  3. Press **Shift+F5** to stop debugging and delete the deployment from the Compute Emulator.

Task 5 – Copying Blobs

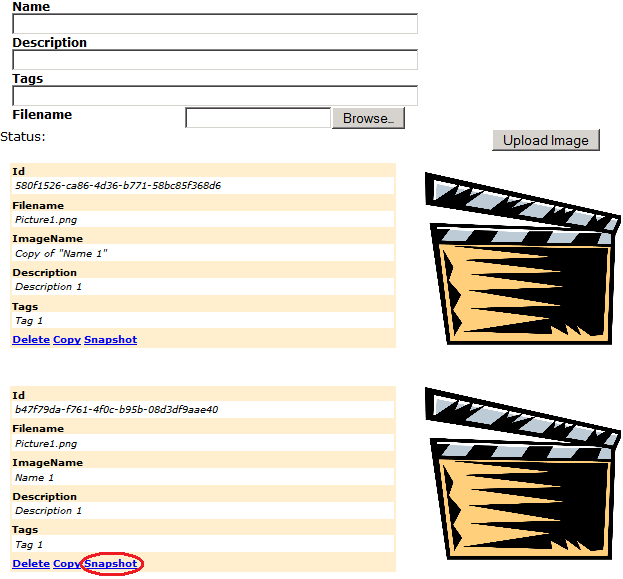
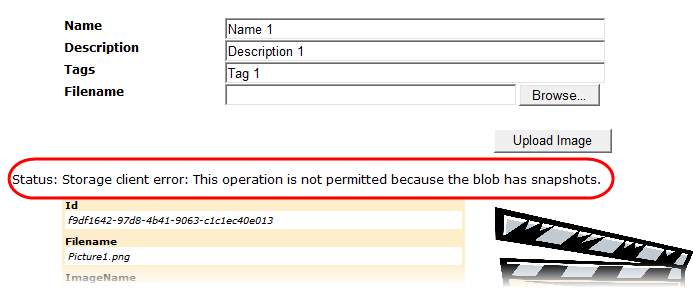
* 1. Windows Azure Blob service has support for making copies of existing blobs. In this task, you will add functionality to the image gallery Web page to copy blobs containing image data from Windows Azure storage that you added earlier.
  2. Update the image list view to add an **asp:LinkButton** control that is used to copy images from the gallery container. Open the **Default.aspx** page in Source mode and locate the **ItemTemplate** for the images **asp:ListView** control. Uncomment the ASP.NET markup located immediately following the delete blob link button control (shown in **bold** text below.)
     1. HTML
     2. ...
     3. <div class="item">
     4. <ul style="width:40em;float:left;clear:left" >
     5. <asp:Repeater ID="blobMetadata" runat="server">
     6. <ItemTemplate>
     7. <li><%# Eval("Name") %><span><%# Eval("Value") %></span></li>
     8. </ItemTemplate>
     9. </asp:Repeater>
     10. <li>
     11. <asp:LinkButton ID="deleteBlob"
     12. OnClientClick="return confirm('Delete image?');"
     13. CommandName="Delete"
     14. CommandArgument='<%# Eval("Uri")%>'
     15. runat="server" Text="Delete" oncommand="OnDeleteImage" />
     16. **<asp:LinkButton ID="CopyBlob"**
     17. **OnClientClick="return confirm('Copy image?');"**
     18. **CommandName="Copy"**
     19. **CommandArgument='<%# Eval("Uri")%>'**
     20. **runat="server" Text="Copy" oncommand="OnCopyImage" />**

...

* + 1. </li>
    2. </ul>
    3. <img src="<%# Eval("Uri") %>" alt="<%# Eval("Uri") %>" style="float:left"/>
    4. </div>
    5. ...
  1. Add code (shown in **bold**) to **Default.aspx.cs** / **Default.aspx.vb** to implement the command handler for the *copyBlob* **asp:LinkButton** control. The code creates a copy of a blob based on an existing blob. It also updates the “*ImageName*” attribute in its metadata to reflect that it is a copy.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-10-OnCopyImageMethod-CS*)
     2. C#
     3. public partial class \_Default : System.Web.UI.Page
     4. {
     5. ...
     6. protected void OnCopyImage(object sender, CommandEventArgs e)
     7. {
     8. **if (e.CommandName == "Copy")**
     9. **{**
     10. **// Prepare an Id for the copied blob**
     11. **var newId = Guid.NewGuid();**
     12. **// Get source blob**
     13. **var blobUri = (string)e.CommandArgument;**
     14. **var srcBlob = this.GetContainer().GetBlobReference(blobUri);**
     15. **// Create new blob**
     16. **var newBlob = this.GetContainer().GetBlobReference(newId.ToString());**
     17. **// Copy content from source blob**
     18. **newBlob.CopyFromBlob(srcBlob);**
     19. **// Explicitly get metadata for new blob**
     20. **newBlob.FetchAttributes(new BlobRequestOptions { BlobListingDetails = BlobListingDetails.Metadata });**
     21. **// Change metadata on the new blob to reflect this is a copy via UI**
     22. **newBlob.Metadata["ImageName"] = "Copy of \"" + newBlob.Metadata["ImageName"] + "\"";**
     23. **newBlob.Metadata["Id"] = newId.ToString();**
     24. **newBlob.SetMetadata();**
     25. **// Render all blobs**
     26. **RefreshGallery();**
     27. **}**
     28. }
     29. ...

1. }
   * 1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-10-OnCopyImageMethod-VB*)
     2. Visual Basic
     3. Partial Public Class \_Default
     4. Inherits System.Web.UI.Page
     5. ...
     6. Protected Sub OnCopyImage(ByVal sender As Object, ByVal e As CommandEventArgs)
     7. **If e.CommandName = "Copy" Then**
     8. **' Prepare an Id for the copied blob**
     9. **Dim newId = Guid.NewGuid()**
     10. **' Get source blob**
     11. **Dim blobUri = DirectCast(e.CommandArgument, String)**
     12. **Dim srcBlob = Me.GetContainer().GetBlobReference(blobUri)**
     13. **' Create new blob**
     14. **Dim newBlob = Me.GetContainer().GetBlobReference(newId.ToString())**
     15. **' Copy content from source blob**
     16. **newBlob.CopyFromBlob(srcBlob)**
     17. **' Explicitly get metadata for new blob**
     18. **newBlob.FetchAttributes(New BlobRequestOptions())**
     19. **' Change metadata on the new blob to reflect this is a copy via UI**
     20. **newBlob.Metadata("ImageName") = "Copy of """ & newBlob.Metadata("ImageName") & """"**
     21. **newBlob.Metadata("Id") = newId.ToString()**
     22. **newBlob.SetMetadata()**
     23. **' Render all blobs**
     24. **RefreshGallery()**
     25. **End If**
     26. End Sub
     27. ...
     28. End Class
   1. Press **F5** to build and run the application.
   2. Upload a few more images from **Source\Assets\Images** and click **Copy** on any of the images displayed to make a copy of the corresponding blob from storage.
      1. 
      2. Figure 21
      3. Copying image blobs from storage
   3. Click **OK** to confirm the copy operation. You should see a copy of the image has been created with **ImageName** metadata stating it is a copy.
      1. 
      2. Figure 22
      3. Verification
   4. Press **Shift+F5** to stop debugging and delete the deployment from the Compute Emulator.

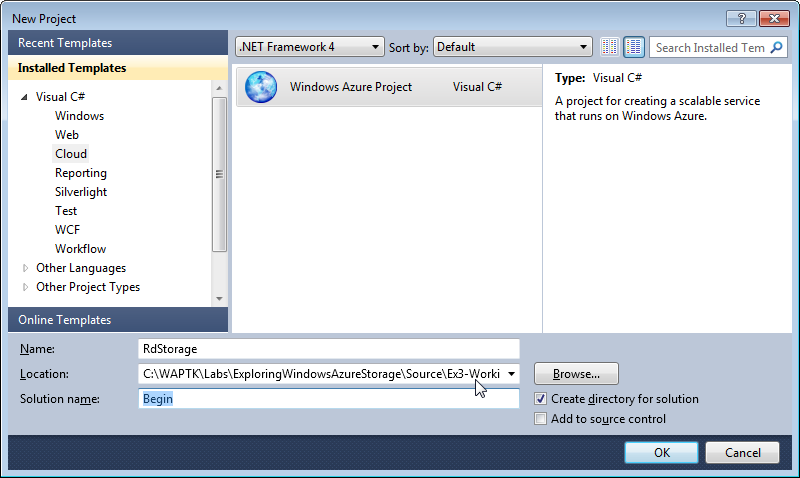
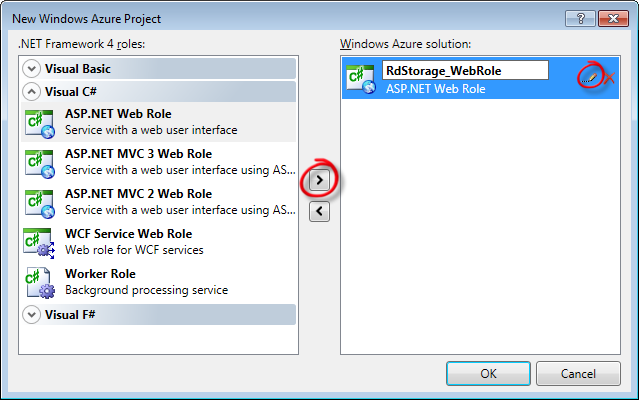
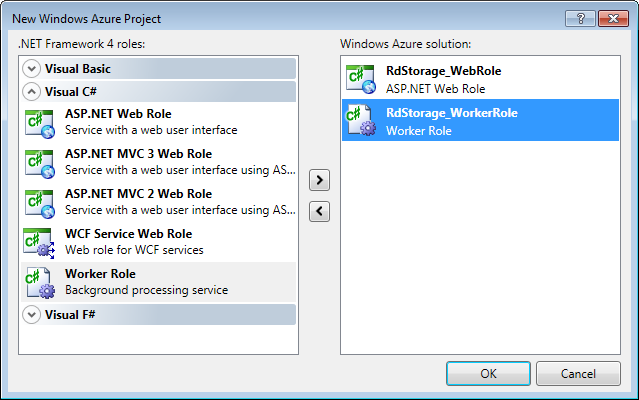
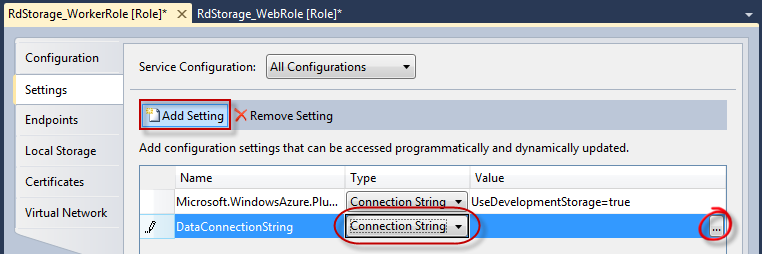
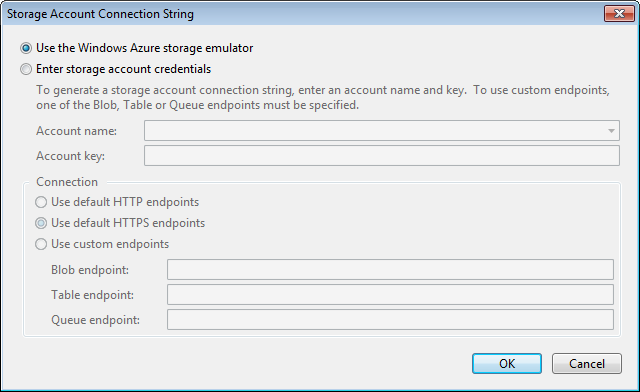
Task 6 – Taking Blob Snapshots

* 1. Windows Azure Blob service has support for taking snapshots of blobs. The difference between a snapshot and a copy is that snapshots are read-only and the original blob maintains a relationship to its snapshots; blob copies on the other hand are editable. Once a snapshot has been taken for a blob, this source blob can no longer be deleted. Before a source blob can be deleted, all of its snapshots must be deleted first.
  2. In this task, you add functionality to take a snapshot of a blob that contains image data from Windows Azure storage.
  3. Update the image list view to add an **asp:LinkButton** control that is used to snapshot images from the gallery container. Open the **Default.aspx** page in Source mode and locate the **ItemTemplate** for the images **asp:ListView** control. Uncomment the ASP.NET markup located immediately following the copy blob link button control (shown **in bold**).
     1. HTML
     2. <div class="item">
     3. <ul style="width:40em;float:left;clear:left" >
     4. <asp:Repeater ID="blobMetadata" runat="server">
     5. <ItemTemplate>
     6. <li><%# Eval("Name") %><span><%# Eval("Value") %></span></li>
     7. </ItemTemplate>
     8. </asp:Repeater>
     9. <li>
     10. <asp:LinkButton ID="deleteBlob"
     11. OnClientClick="return confirm('Delete image?');"
     12. CommandName="Delete"
     13. CommandArgument='<%# Eval("Uri")%>'
     14. runat="server" Text="Delete" oncommand="OnDeleteImage" />
     15. <asp:LinkButton ID="CopyBlob"
     16. OnClientClick="return confirm('Copy image?');"
     17. CommandName="Copy"
     18. CommandArgument='<%# Eval("Uri")%>'
     19. runat="server" Text="Copy" oncommand="OnCopyImage" />
     20. **<asp:LinkButton ID="SnapshotBlob"**
     21. **OnClientClick="return confirm('Snapshot image?');"**
     22. **CommandName="Snapshot"**
     23. **CommandArgument='<%# Eval("Uri")%>'**
     24. **runat="server" Text="Snapshot" oncommand="OnSnapshotImage" />**
     25. </li>
     26. </ul>
     27. <img src="<%# Eval("Uri") %>" alt="<%# Eval("Uri") %>" style="float:left"/>
     28. </div>
  4. Add code (shown in **bold**) to **Default.aspx.cs** / **Default.aspx.vb** to implement the command handler for the *snapshotBlob* **asp:LinkButton** control. The code gets the source blob and takes a snapshot of it.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-11-OnSnapshotImageMethod-CS*)
     2. C#
     3. public partial class \_Default : System.Web.UI.Page
     4. {
     5. ...
     6. protected void OnSnapshotImage(object sender, CommandEventArgs e)
     7. {
     8. **if (e.CommandName == "Snapshot")**
     9. **{**
     10. **// Get source blob**
     11. **var blobUri = (string)e.CommandArgument;**
     12. **var srcBlob = this.GetContainer().GetBlobReference(blobUri);**
     13. **// Create a snapshot**
     14. **var snapshot = srcBlob.CreateSnapshot();**
     15. **status.Text = "A snapshot has been taken for image blob:" + srcBlob.Uri + " at " + snapshot.SnapshotTime;**
     16. **RefreshGallery();**
     17. **}**
     18. }
     19. ...
     20. }
     21. (Code Snippet – *ExploringWindowsAzureStorage-Ex02-11-OnSnapshotImageMethod-VB*)
     22. Visual Basic
     23. Partial Public Class \_Default
     24. Inherits System.Web.UI.Page
     25. ...
     26. Protected Sub OnSnapshotImage(ByVal sender As Object, ByVal e As CommandEventArgs)
     27. **If e.CommandName = "Snapshot" Then**
     28. **' Get source blob**
     29. **Dim blobUri = DirectCast(e.CommandArgument, String)**
     30. **Dim srcBlob = Me.GetContainer().GetBlobReference(blobUri)**
     31. **' Create a snapshot**
     32. **Dim snapshot = srcBlob.CreateSnapshot()**
     33. **status.Text = "A snapshot has been taken for image blob:" + srcBlob.Uri.ToString() + " at " + snapshot.SnapshotTime**
     34. **RefreshGallery()**
     35. **End If**
     36. End Sub
     37. ...
     38. End Class
  5. Press **F5** to build and run the application.
  6. Click **Snapshot** on any of the images displayed to take a snapshot of the corresponding blob from storage.
     1. 
     2. Figure 23
     3. Taking a snapshot of image blobs from storage
  7. Click **OK** to confirm the snapshot operation. You will see a status update confirming that a snapshot has been taken.
  8. Attempt to delete the **original** blob from which the snapshot was taken.
     1. 
     2. Figure 24
     3. Cannot Delete Snapshot error
  9. You will see a status update confirms that the blob cannot be deleted.
     1. 
     2. Figure 25
     3. Cannot Delete Snapshot error
     4. **Note:**  To delete a blob that contains snapshots, all of its snapshots must be deleted first. (That functionality is not provided in this solution)

Exercise 3: Working with Queues

* 1. In this exercise, you create a simple Web application to send messages to a Windows Azure queue. A Worker role in the solution retrieves the messages and writes them to the compute emulator log.
  2. Queue service is a great way to send messages between front-end roles and worker roles. A queue can contain an unlimited number of messages, each of which can be up to 64 KB in size. Messages are pushed to the end of the queue and popped from the front of the queue.

Task 1 – Creating the Initial Solution

* 1. In this task, you create and configure the initial solution to work with queues in Windows Azure.
  2. Open Microsoft Visual Studio 2010 elevated **as Administrator** from **Start** | **All Programs** | **Microsoft Visual Studio 2010** by right-clicking **Microsoft Visual Studio 2010** and choosing **Run as administrator**.
  3. From the **File** menu, choose **New** **Project**.
  4. In the **New Project** dialog, expand the language of your preference (Visual C# or Visual Basic) in the **Installed Templates** list and select **Cloud**. Choose the **Windows Azure Project** template, set the **Name** of the project to **RdStorage**, set the location to **Ex03-WorkingWithQueues\begin\[CS|VB]** in the **Source** folder of the lab, change the solution name to **begin**, and ensure that **Create directory for solution** is checked. Click **OK** to create the project.
     1. 
     2. Figure 26
     3. Creating a WindowsAzure Project
  5. In the **New Windows Azure Project** dialog, select **ASP.NET Web Role** from the list of available roles and click the arrow (>) to add an instance of this role to the solution. Change the name of the role to **RdStorage\_WebRole**. To do this, select the role in the right panel, click the pencil icon and enter the new name. Do not close the dialog. You will add a second role in the next step.
     1. 
     2. Figure 27
     3. Adding a Web Role to the windows azure project (Visual C#)
     4. **Note**: Select the Visual Basic equivalent if you are creating a VB.NET project.
  6. Next, add a second role to the solution. Choose a **Worker Role** and change its name to **RdStorage\_WorkerRole**.
     1. 
     2. Figure 28
     3. Adding a new Worker Role to the cloud service project (C#)
     4. **Note:** Select the Visual Basic equivalent if you are creating a VB.NET project.
  7. Click **OK** to close the **New Windows Azure Project** dialog and create the solution.
  8. Right-click each role under the **Roles** folder from the **RdStorage** cloud project. Choose **Properties**.
  9. In the **Settings** tab, click **Add Setting** and create a **ConnectionString** type called *DataConnectionString*. Click the button labeled with an ellipsis and set the connection string to **Use storage emulator**. Repeat for each role in your project.
     1. 
     2. Figure 29
     3. Creating a storage connection string
     4. 
     5. Figure 30
     6. Configuring a connection string to use Compute Emulator

Task 2 – Sending Messages to the Queue

* 1. In this task, you implement the **RdStorage\_WebRole** web application to send messages to the queue.
  2. Open the **Default.aspx** page of the **RdStorage\_WebRole** web site. Delete the existing content inside the *BodyContent* **Content** control.
  3. Add an asp:**TextBox** control to the page. Change the **ID** for the **TextBox** to **txtMessage**. You may wish to format the page to look good, but this is not required. Also, add an asp:**Button** control just after the **TextBox** inserted previously. Change the **ID** of the button to **btnSend** and set the **Text** to **Send message**. Your code should look similar to the following:
     1. C#
     2. <%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"
     3. CodeBehind="Default.aspx.cs" Inherits="RdStorage\_WebRole.\_Default" %>
     4. <asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">
     5. </asp:Content>
     6. <asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
     7. **<asp:TextBox ID="txtMessage" runat="server"></asp:TextBox>**
     8. **<asp:Button ID="btnSend" runat="server" Text="Send message" />**
     9. </asp:Content>
     10. Visual Basic
     11. <%@ Page Title="Home Page" Language="vb" MasterPageFile="~/Site.Master" AutoEventWireup="false"
     12. CodeBehind="Default.aspx.vb" Inherits="RdStorage\_WebRole.\_Default" %>
     13. <asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">
     14. </asp:Content>
     15. <asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
     16. **<asp:TextBox ID="txtMessage" runat="server"></asp:TextBox>**
     17. **<asp:Button ID="btnSend" runat="server" Text="Send message" />**
     18. </asp:Content>
  4. Open the code-behind file for the **Default.aspx** page. To do this, right-click **Default.aspx**, and select **View Code** (**Default.aspx.cs** / **Default.aspx.vb**).
  5. Add the following namespace directives at the top of the file.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-01-Namespace-CS*)
     2. C#
     3. **using Microsoft.WindowsAzure;**
     4. **using Microsoft.WindowsAzure.StorageClient;**
     5. **using Microsoft.WindowsAzure.ServiceRuntime;**
     6. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-01-Namespace-VB*)
     7. Visual Basic
     8. **Imports Microsoft.WindowsAzure**
     9. **Imports Microsoft.WindowsAzure.StorageClient**
     10. **Imports Microsoft.WindowsAzure.ServiceRuntime**
  6. Right-click **Default.aspx**, select **View** **Designer** and double-click the **Send message** button. Alternatively, you may edit the ASP.NET markup directly to insert the required event handler. Add the following code (shown in **bold** ) to the **btnSend\_Click** event to initialize the account information:
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-02-WebRoleCreateAccount-CS*)
     2. C#
     3. public partial class \_Default : System.Web.UI.Page
     4. {
     5. ...
     6. protected void btnSend\_Click(object sender, EventArgs e)
     7. {
     8. **// initialize the account information**
     9. **var storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");**
     10. }
     11. }
     12. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-02-WebRoleCreateAccount-VB*)
     13. Visual Basic
     14. Public Class \_Default
     15. Inherits System.Web.UI.Page
     16. ...
     17. Protected Sub btnSend\_Click(ByVal sender As Object, ByVal e As EventArgs) Handles btnSend.Click
     18. **' initialize the account information**
     19. **Dim account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString")**
     20. End Sub
     21. End Class
  7. Next, add the following code (shown in **bold**) immediately after the code inserted in the previous step to obtain an instance of the **QueueStorage** helper object and create the message queue if it does not exist.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-03-WebRoleCreateQueue-CS*)
     2. C#
     3. protected void btnSend\_Click(object sender, EventArgs e)
     4. {
     5. ...
     6. // Code inserted in previous steps
     7. **// retrieve a reference to the messages queue**
     8. **var queueClient = storageAccount.CreateCloudQueueClient();**
     9. **var queue = queueClient.GetQueueReference("messagequeue");**
     10. **queue.CreateIfNotExist();**
     11. }
     12. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-03-WebRoleCreateQueue-VB*)
     13. Visual Basic
     14. Protected Sub btnSend\_Click(ByVal sender As Object, ByVal e As EventArgs) Handles btnSend.Click

...

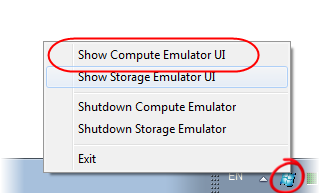
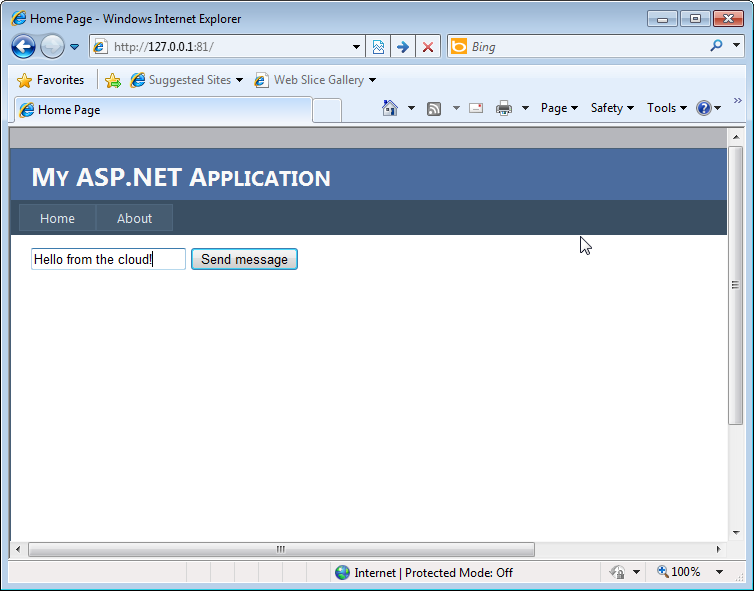
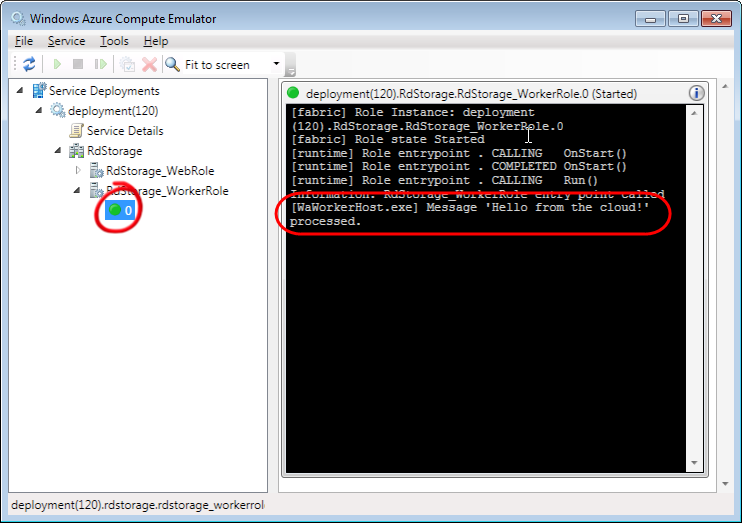
'Code inserted in previous steps

**'retrieve a reference to the messages queue**

* + 1. **Dim queueClient = account.CreateCloudQueueClient()**
    2. **Dim queue As CloudQueue = queueClient.GetQueueReference("messagequeue")**
    3. **queue.CreateIfNotExist()**
    4. End Sub
  1. Add the following code (shown in **bold**) immediately after the code inserted in the previous step to put the message entered by the user into the queue.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-04-WebRoleAddMessage-CS*)
     2. C#
     3. protected void btnSend\_Click(object sender, EventArgs e)
     4. {
     5. ...
     6. // Code inserted in previous steps
     7. **// add the message to the queue**
     8. **var msg = new CloudQueueMessage(txtMessage.Text);**
     9. **queue.AddMessage(msg);**
     10. **txtMessage.Text = string.Empty;**
     11. }
     12. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-04-WebRoleAddMessage-VB*)
     13. Visual Basic
     14. Protected Sub btnSend\_Click(ByVal sender As Object, ByVal e As EventArgs) Handles btnSend.Click
     15. ...
     16. 'Code inserted in previous steps
     17. **' add the message to the queue**
     18. **Dim msg As New CloudQueueMessage(Me.txtMessage.Text)**
     19. **queue.AddMessage(msg)**
     20. **Me.txtMessage.Text = String.Empty**
     21. End Sub
  2. Open the **WebRole.cs** / **WebRole.vb** file from the **RDStorage\_WebRole** project.
  3. Append the following namespace to the existing directives at the top of the code file.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-05-Namespace-CS*)
     2. C#
     3. **using Microsoft.WindowsAzure.StorageClient;**
     4. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-05-Namespace-VB*)
     5. Visual Basic
     6. **Imports Microsoft.WindowsAzure.StorageClient**

Task 3 – Retrieving Messages from the Queue

* 1. In this task, you update the worker role to retrieve messages from the queue and show them in the compute emulator log.
  2. Open the **Global.asax.cs** / **Global.asax.vb** file from the **RdStorage\_WebRole** project.
  3. Add the following code (shown in **bold**) to the **Application\_Start** method to initialize the account information.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-06-InitializeAccount-CS*)
     2. C#
     3. void Application\_Start(object sender, EventArgs e)
     4. {
     5. **CloudStorageAccount.SetConfigurationSettingPublisher((configName, configSetter) =>**
     6. **{**
     7. **configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));**
     8. **});**
     9. }
     10. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-06-InitializeAccount-VB*)
     11. Visual Basic
     12. Sub Application\_Start()
     13. **CloudStorageAccount.SetConfigurationSettingPublisher(Sub(configName, configSetter)**
     14. **configSetter(RoleEnvironment.GetConfigurationSettingValue(configName))**
     15. **End Sub)**
     16. End Sub
  4. Make sure the following namespace directives exist on the top of the code files.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-07-Namespace-CS*)
     2. C#
     3. **using Microsoft.WindowsAzure;**
     4. **using Microsoft.WindowsAzure.ServiceRuntime;**
     5. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-07-Namespace*-VB)
     6. Visual Basic
     7. **Imports Microsoft.WindowsAzure**
     8. **Imports Microsoft.WindowsAzure.ServiceRuntime**
  5. Open the **WorkerRole.cs** / **WorkerRole.vb** file from the **RdStorage\_WorkerRole** project.
  6. Add the following code (shown in **bold**) to the **OnStart** method directly above **return base.OnStart()** in WorkerRole.cs or **Return MyBase.OnStart()** in WorkerRole.vb to initialize the account information.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-08-InitializeAccount-CS*)
     2. C#
     3. public override bool OnStart()
     4. {
     5. // Set the maximum number of concurrent connections
     6. ServicePointManager.DefaultConnectionLimit = 12;
     7. // For information on handling configuration changes
     8. // see the MSDN topic at http://go.microsoft.com/fwlink/?LinkId=166357.
     10. **CloudStorageAccount.SetConfigurationSettingPublisher((configName, configSetter) =>**
     11. **{**
     12. **configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));**
     13. **});**
     14. return base.OnStart();
     15. }
     16. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-08-InitializeAccount-VB*)
     17. Visual Basic
     18. Public Overrides Function OnStart() As Boolean
     19. ' Set the maximum number of concurrent connections
     20. ServicePointManager.DefaultConnectionLimit = 12
     21. ' For information on handling configuration changes
     22. ' see the MSDN topic at http://go.microsoft.com/fwlink/?LinkId=166357.
     24. **CloudStorageAccount.SetConfigurationSettingPublisher(Sub(configName, configSetter)**
     25. **configSetter(RoleEnvironment.GetConfigurationSettingValue(configName))**
     26. **End Sub)**
     27. Return MyBase.OnStart()
     28. End Function
  7. Make sure the following namespace directives exist on the top of the code files.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-09-Namespace-CS*)
     2. C#
     3. **using System.Diagnostics;**
     4. **using System.Threading;**
     5. **using Microsoft.WindowsAzure;**
     6. **using Microsoft.WindowsAzure.Diagnostics;**
     7. **using Microsoft.WindowsAzure.ServiceRuntime;**
     8. **using Microsoft.WindowsAzure.StorageClient;**
     9. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-09-Namespace*-VB)
     10. Visual Basic
     11. **Imports System.Diagnostics**
     12. **Imports System.Threading**
     13. **Imports Microsoft.WindowsAzure**
     14. **Imports Microsoft.WindowsAzure.Diagnostics**
     15. **Imports Microsoft.WindowsAzure.ServiceRuntime**
     16. **Imports Microsoft.WindowsAzure.StorageClient**
  8. In the **Run** method, obtain an instance of the **QueueStorage** helper object and retrieve a reference to the *messages* queue. To do this, add the following code (shown in **bold**) and remove the following lines of code (shown in ~~strikethrough~~) that simulate the worker role latency.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-10-WorkerGetQueue-CS*)
     2. C#
     3. public override void Run()
     4. {
     5. // This is a sample worker implementation. Replace with your logic.
     6. Trace.WriteLine("RdStorage\_WorkerRole entry point called", "Information");
     7. ~~while (true)~~
     8. ~~{~~
     9. ~~Thread.Sleep(10000);~~
     10. ~~Trace.WriteLine("Working", "Information");~~
     11. ~~}~~
     12. **// initialize the account information**
     13. **var storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");**
     14. **// retrieve a reference to the messages queue**
     15. **var queueClient = storageAccount.CreateCloudQueueClient();**
     16. **var queue = queueClient.GetQueueReference("messagequeue");**
     17. }
     18. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-10-WorkerGetQueue-VB*)
     19. Visual Basic
     20. Public Overrides Sub Run()
     21. ' This is a sample worker implementation for RdStorage\_WorkerRole. Replace with your logic.
     22. Trace.WriteLine("RdStorage\_WorkerRole entry point called.", "Information")
     23. ~~While (True)~~
     24. ~~Thread.Sleep(10000)~~
     25. ~~Trace.WriteLine("Working", "Information")~~
     26. ~~End While~~
     27. **' initialize the account information**
     28. **Dim storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString")**
     29. **' retrieve a reference to the messages queue**
     30. **Dim queueClient As CloudQueueClient = storageAccount.CreateCloudQueueClient()**
     31. **Dim queue = queueClient.GetQueueReference("messagequeue")**
     32. End Sub
  9. Next, add the following code (shown in **bold**) to retrieve messages and write them to the compute emulator log. The message is then removed from the queue.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-11-WorkerGetMessages-CS*)
     2. C#
     3. public override void Run()
     4. {
     5. ...
     6. //Code inserted from previous steps
     7. **// retrieve messages and write them to the compute emulator log**
     8. **while (true)**
     9. **{**
     10. **Thread.Sleep(10000);**
     11. **if (queue.Exists())**
     12. **{**
     13. **var msg = queue.GetMessage();**
     14. **if (msg != null)**
     15. **{**
     16. **Trace.TraceInformation(string.Format("Message '{0}' processed.", msg.AsString));**
     17. **queue.DeleteMessage(msg);**
     18. **}**
     19. **}**
     20. **}**
     21. }
     22. (Code Snippet – *ExploringWindowsAzureStorage-Ex03-11-WorkerGetMessages-VB*)
     23. Visual Basic
     24. Public Overrides Sub Run()
     25. ...
     26. ' Code inserted from previous steps
     27. **' retrieve messages and write them to the compute emulator log**
     28. **While (True)**
     29. **Thread.Sleep(10000)**
     30. **If queue.Exists() Then**
     31. **Dim msg = queue.GetMessage()**
     32. **If (msg IsNot Nothing) Then**
     33. **Trace.TraceInformation(String.Format("Message '{0}' processed.", msg.AsString))**
     34. **queue.DeleteMessage(msg)**
     35. **End If**
     36. **End If**
     37. **End While**
     38. End Sub
     39. **Note:** The worker process will try to get a message from the queue every 10 seconds using the **GetMessage** method. If there are messages in the queue, it will show them in the Compute Emulator log.

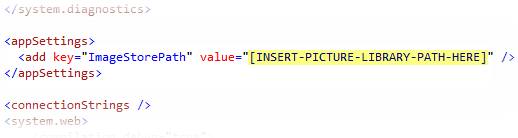
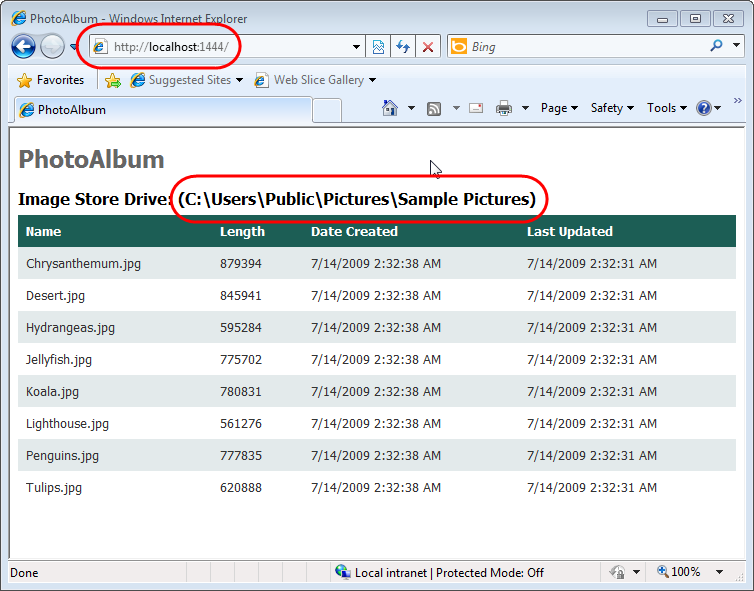
1. Verification
   1. To test your service running in the compute emulator:
   2. In Visual Studio, press **F5** to build and run the application.
   3. Open the compute emulator UI. To do this, right-click its icon located in the system tray and select **Show Compute Emulator UI**. (The icon is an azure colored Window.)
      1. 
      2. Figure 31
      3. Showing the compute emulator UI
   4. Expand the tree to show the **Worker** instance log.
   5. Switch back to Windows Internet Explorer. Ensure that the default page is shown, enter a message, and click **Send message**.
      1. 
      2. Figure 32
      3. Default Web page
   6. Change back to the Compute Emulator UI. You should see the message logged in the worker role log.
      1. 
      2. Figure 33
      3. Worker log showing the message
      4. **Note:** Because of the worker sleep time, it may take several seconds to show the message.

Exercise 4: Working with Drives

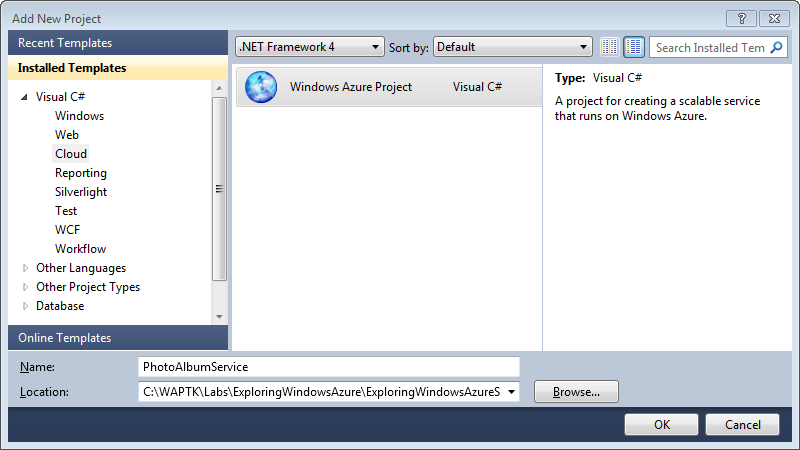
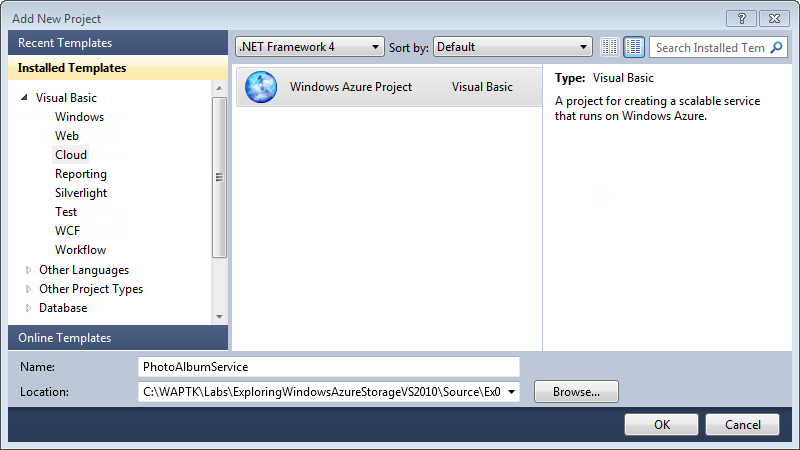
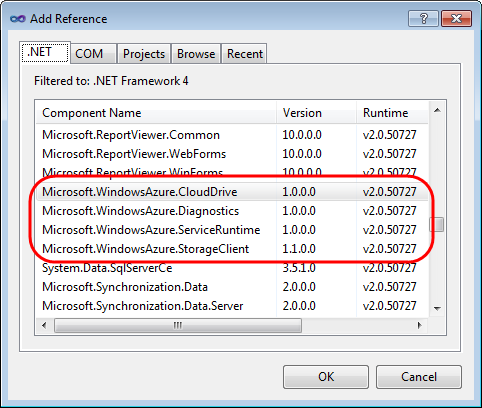
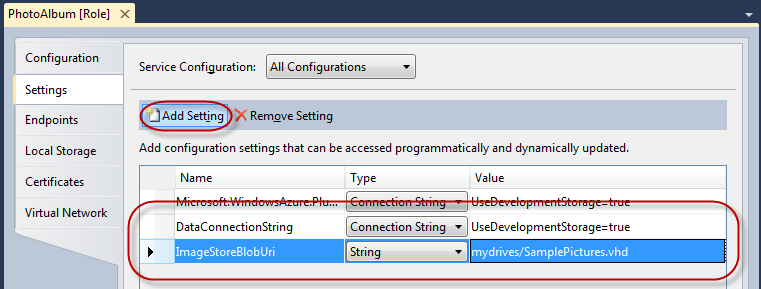
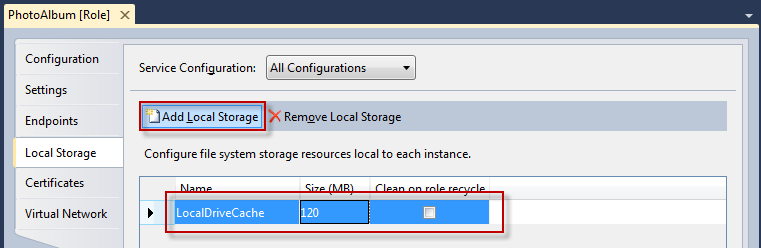
* 1. A Windows Azure Drive is an NTFS formatted virtual hard disk (VHD) file that is stored in a page blob. You can mount this VHD into a Windows Azure Compute instance to provide persistent storage exposed to applications via the Windows file system. The content of an Azure Drive will persist even if the compute role to which it is mounted is recycled.
  2. In this exercise, you take an existing application that makes use of regular Windows file system APIs to access information in local disk storage and run it as a Windows Azure service. You will see that by using a Windows Azure Drive, no changes to the code are necessary to run the same application in the Azure cloud and have it access information stored in Azure storage.

1. In the first part of the exercise, you execute the original application in the ASP.NET development server to familiarize yourself with its operation. Next, you create a cloud service project, associate the application as a Web role, and run it in the compute emulator using simulated Azure Drives. Finally, you create a VHD on your local machine, upload it to blob storage, deploy the application to Windows Azure and mount the drive in a Windows Azure instance.

Task 1 – Exploring the PhotoAlbum Application

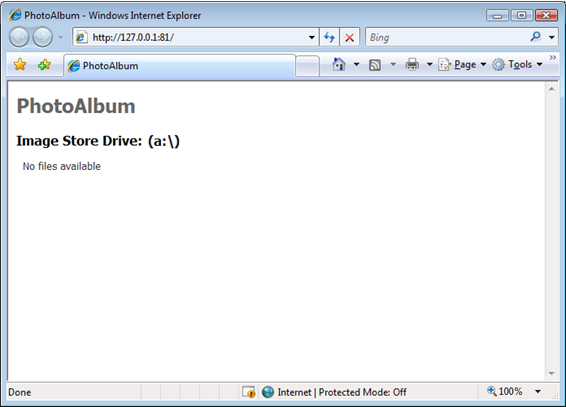
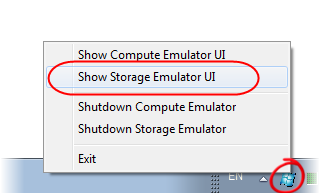
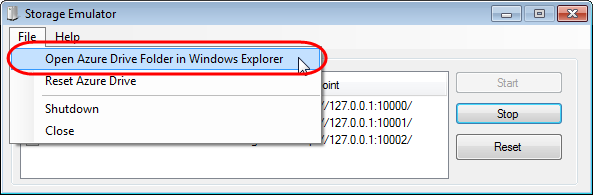
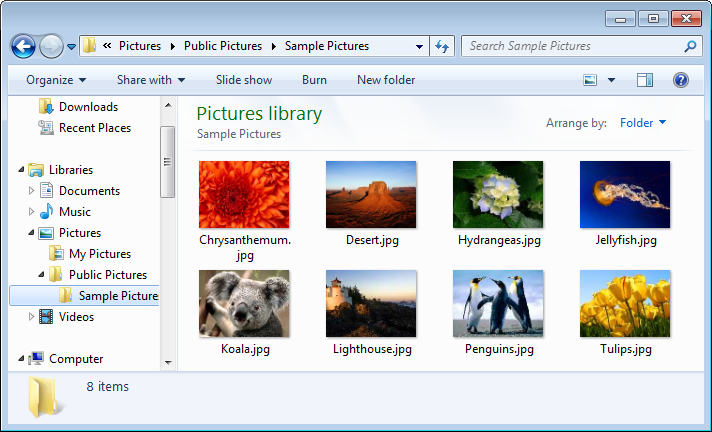
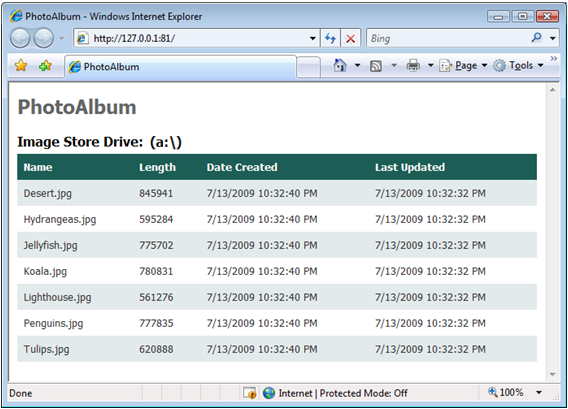
* 1. PhotoAlbum is a sample application that uses standard file system APIs to obtain a directory listing of the contents of its image store. In this task, you briefly examine the application and then configure the location of the image store to a folder in your machine to run the application locally.
  2. Open Microsoft Visual Studio 2010 in elevated administrator mode. To do this, in **Start** | **All Programs** | **Microsoft Visual Studio 2010**, right-click the **Microsoft Visual Studio 2010** shortcut and choose **Run as administrator**.
     1. **Note:** You are not required to use elevated administrator mode to run the application using the ASP.NET Web development server. However, you will need to do this later, when you launch the application in the compute emulator.
  3. If the **User Account Control** dialog appears, click **Continue**.
  4. Open the begin solution provided for this exercise. In the **File** menu, choose **Open Project**. Then, in the **Open Project** dialog, navigate to **\Source\Ex04-WorkingWithDrives\begin**. Select the folder for the language of your preference (C# or VB), and then open the **PhotoAlbum.sln** solution.
  5. Take a minute to browse the files in the application. First, double-click **Default.aspx** in **Solution Explorer** to open this file. Notice that the page uses a **GridView** control to display a directory listing of the image store and that its data source is a **LinqDataSource** control bound to a **PhotoAlbumDataSource** context object.
  6. Next, open the **PhotoAlbumDataSource.cs** or **PhotoAlbumDataSource.vb** file to examine the context object class used by the **LinqDataSource**. Notice that the **Files** property used by the data source control returns a collection of [FileInfo](http://msdn.microsoft.com/en-us/library/system.io.fileinfo.aspx) objects and that it uses standard file system APIs to enumerate PNG and JPEG files in the image store directory.
  7. Now, open the **Global.asax** file and see that it contains an **ImageStorePath** property that returns the location of the image store folder and that the **Application\_Start** event handler initializes this property with a path that it retrieves from the application settings.
  8. Before you run the application, you need to configure the location of the image store. In **Solution Explorer**, double-click **Web.config** to open this file in the text editor. In the **appSettings** section, locate the *ImageStorePath* setting and update its value to the path of the **Sample Pictures** folder in your machine.
     1. 
     2. Figure 34
     3. Configuring the location of the image store
     4. **Note:** The sample pictures library is typically available in most Windows installations and is located at “*%PUBLIC%\Pictures\Sample Pictures”*. It contains a small set of image files. If this folder is not available in your environment, you may substitute it with any folder that contains a suitable collection of JPEG or PNG image files, for example, those found in *Assets\Images* in the *Source* folder of this lab.
     5. Note that *%PUBLIC%* is an environment variable that points to the location of *Public* in the User profiles folder. You must expand the variable to the proper value when you configure the *ImageStorePath* setting (e.g. *C:\Users\Public\Pictures\Sample Pictures*).
  9. Press **F5** to build and run the PhotoAlbum application. Notice that the default page shows a listing of the files contained in the image store. Also, notice that the path to the image store is the folder on your machine that you set up in the application configuration file.
     1. 
     2. Figure 35
     3. Running the PhotoAlbum application locally
  10. Close the browser window. You have now seen that the application uses standard file system APIs to access the files in its store. In the next task, you will update the application to run as a cloud service.

Task 2 – Using a Windows Azure Drive to Move the Application to the Cloud

* 1. When moving the application to Windows Azure, the natural choice is to relocate the image store to blob storage. Regardless, the application expects its images to be stored in the file system. You can always update the code that accesses the images in the store and change it to use the Blob service APIs instead. For a simple application such as this one, this would not be too challenging, but it can represent a barrier for a more complex application. Using Windows Azure Drives enables you to move the application to the cloud without any changes to its code, other than mounting the drive on a page blob.
  2. In this task, you update the application to run as a Windows Azure cloud service and to use a Windows Azure Drive for its image store.
  3. Add a cloud service project to the solution. In **Solution Explorer**, right-click the root solution node, point to **Add** and then select **New Project**.
  4. In the **Add** **New Project** dialog, expand the language of your preference (Visual C# or Visual Basic) in the **Installed Templates** list and select **Cloud**. Choose the **Windows Azure Project** template, set the **Name** of the project to **PhotoAlbumService**, leave the proposed location unchanged and then click **OK**.
     1. 
     2. Figure 36
     3. Creating a Windows Azure Project (Visual C#)
     4. 
     5. Figure 37
     6. Creating a Windows Azure Project (Visual Basic)
  5. In the **New Windows Azure Project** dialog, click **OK** without adding any new roles. You will use the existing application as a web role.
  6. Add a reference to the Windows Azure support assemblies. In **Solution Explorer**, right-click the **PhotoAlbum** project and select **Add Reference**. In the **Add Reference** dialog, switch to the **.NET** tab, select the **Microsoft.WindowsAzure.CloudDrive**, **Microsoft.WindowsAzure.Diagnostics**, **Microsoft.WindowsAzure.ServiceRuntime**, and **Microsoft.WindowsAzure.StorageClient** assemblies, and then click **OK**.
     1. 
     2. Figure 38
     3. Adding a reference to the Windows Azure support assemblies
  7. Now, in **Solution Explorer**, right-click the **Roles** node in the **PhotoAlbumService** project, point to **Add** and then select **Web Role Project in solution**.
  8. In the **Associate with Role Project** dialog, select the **PhotoAlbum** project and click **OK**.
  9. You will now configure the web role. To do this, double-click the **PhotoAlbum** role under the **Roles** node in the **PhotoAlbumService** project.
  10. In the **PhotoAlbum [Role]** properties window, switch to the **Settings** tab and then click **Add Setting**. Set the **Name** of the new setting to *DataConnectionString*, set the **Type** as *Connection String*, and then click the button labeled with an ellipsis located on the right side of the **Value** column. In the **Storage Connection String** dialog, choose the option labeled **Use storage emulator** and click **OK**.
  11. Add a second setting to configure the URL of the cloud drive in blob storage. For this setting, set the **Name** to *ImageStoreBlobUri*, the **Type** as *String*, and the **Value** as “*mydrives/SamplePictures.vhd*”.
      1. **Note:** The *ImageStoreBlobUri* setting identifies a Blob service URI and is case sensitive. Make sure that you enter the value exactly as shown.
      2. 
      3. Figure 39
      4. Configuring the Web role settings
  12. Switch to the **Local Storage** tab and then click **Add Local Storage**. Set the **Name** of the new storage setting to *LocalDriveCache*, set the **Size** to *120* and leave **Clean on Role Recycle** unselected.
      1. 
      2. Figure 40
      3. Configuring local storage for the Web role to cache Azure drive contents
      4. **Note:** When **Clean on Role Recycled** is disabled, the contents of the cache will persist even if the role instance is recycled.
      5. Due to the way that the system allocates local storage resources, part of the requested space is unavailable for cache usage. Beginning with Windows Azure Guest OS 1.8, the entire Windows Azure Drive cache is allocated up-front and if there is not enough space in the local resource for the cache, the call to Mount will fail. To avoid this issue when configuring the size of the local resource destined for the drive cache, specify an additional 20MB to the required size. For example, to use a 1000MB cache size, set the size of the local resource to 1020MB.
  13. Press **CTRL+S** to save the changes to the role configuration.
  14. Configure a trace listener to output diagnostics information to the Windows Azure log. To do this, double-click **Web.config** in **Solution Explorer** to open this file and insert the following **system.diagnostics** section into the configuration, as shown below.
      1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-01-DiagnosticMonitorTraceListener*)
      2. XML
      3. <?xml version="1.0"?>
      4. <configuration>
      5. **<system.diagnostics>**
      6. **<trace>**
      7. **<listeners>**
      8. **<add type="Microsoft.WindowsAzure.Diagnostics.DiagnosticMonitorTraceListener, Microsoft.WindowsAzure.Diagnostics, Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"**
      9. **name="AzureDiagnostics">**
      10. **<filter type="" />**
      11. **</add>**
      12. **</listeners>**
      13. **</trace>**
      14. **</system.diagnostics>**
      15. ...
      16. </configuration>
  15. Add a class to the Web role project to manage its initialization and shutdown. In **Solution Explorer**, right-click the **PhotoAlbum** project, point to **Add** and then select **Existing Item**. In the **Add Existing Item** dialog, browse to **Ex04-WorkingWithDrives\Assets**, select **WebRole.cs** or **WebRole.vb**, depending on the language of the project (Visual C# or Visual Basic) and then click **Add**.
      1. **Note:** The **WebRole(.cs|.vb)** file contains a standard **RoleEntryPoint** derived class, similar to the one generated when you select a new Windows Azure Web Role project template in Visual Studio.
  16. Locate the **Application\_Start** method in **Global.asax.cs** / **Global.asax.vb** file and insert the following code (shown in **bold**) to this method.
      1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-02-ApplicationStartMethod-CS*)
      2. C#
      3. protected void Application\_Start(object sender, EventArgs e)
      4. {
      5. if (imageStorePath == null)
      6. {
      7. ImageStorePath = WebConfigurationManager.AppSettings["ImageStorePath"];
      8. }
      9. **// initialize storage account configuration setting publisher**
      10. **CloudStorageAccount.SetConfigurationSettingPublisher((configName, configSetter) =>**
      11. **{**
      12. **string connectionString = RoleEnvironment.GetConfigurationSettingValue(configName);**
      13. **configSetter(connectionString);**
      14. **});**
      15. **try**
      16. **{**
      17. **// initialize the local cache for the Azure drive**
      18. **LocalResource cache = RoleEnvironment.GetLocalResource("LocalDriveCache");**
      19. **CloudDrive.InitializeCache(cache.RootPath + "cache", cache.MaximumSizeInMegabytes);**
      20. **// retrieve storage account**
      21. **CloudStorageAccount account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");**
      22. **// retrieve URI for the page blob that contains the cloud drive from configuration settings**
      23. **string imageStoreBlobUri = RoleEnvironment.GetConfigurationSettingValue("ImageStoreBlobUri");**
      24. **// unmount any previously mounted drive.**
      25. **foreach (var drive in CloudDrive.GetMountedDrives())**
      26. **{**
      27. **var mountedDrive = account.CreateCloudDrive(drive.Value.PathAndQuery);**
      28. **mountedDrive.Unmount();**
      29. **}**
      30. **// create the Windows Azure drive and its associated page blob**
      31. **CloudDrive imageStoreDrive = account.CreateCloudDrive(imageStoreBlobUri);**
      32. **try**
      33. **{**
      34. **imageStoreDrive.Create(16);**
      35. **}**
      36. **catch (CloudDriveException)**
      37. **{**
      38. **// drive already exists**
      39. **}**
      40. **// mount the drive and initialize the application with the path to the image store on the Azure drive**
      41. **Global.ImageStorePath = imageStoreDrive.Mount(cache.MaximumSizeInMegabytes / 2, DriveMountOptions.None);**
      42. **}**
      43. **catch (CloudDriveException driveException)**
      44. **{**
      45. **Trace.WriteLine("Error: " + driveException.Message);**
      46. **}**
      47. }
      48. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-02-ApplicationStartMethod-VB*)
      49. Visual Basic
      50. Protected Sub Application\_Start(sender As Object, e As EventArgs)
      51. If ImageStorePath Is Nothing Then
      52. ImageStorePath = WebConfigurationManager.AppSettings("ImageStorePath")
      53. End If
      54. **' initialize storage account configuration setting publisher**
      55. **CloudStorageAccount.SetConfigurationSettingPublisher(Function(configName, configSetter) ConfigurationSettingPublisher(configName, configSetter))**
      56. **Try**
      57. **' initialize the local cache for the Azure drive**
      58. **Dim cache As LocalResource = RoleEnvironment.GetLocalResource("LocalDriveCache")**
      59. **CloudDrive.InitializeCache(cache.RootPath & "cache", cache.MaximumSizeInMegabytes)**
      60. **' retrieve storage account**
      61. **Dim account As CloudStorageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString")**
      62. **' retrieve URI for the page blob that contains the cloud drive from configuration settings**
      63. **Dim imageStoreBlobUri As String = RoleEnvironment.GetConfigurationSettingValue("ImageStoreBlobUri")**
      64. **' unmount any previously mount drive**
      65. **For Each drive In CloudDrive.GetMountedDrives()**
      66. **Dim mountedDrive = account.CreateCloudDrive(drive.Value.PathAndQuery)**
      67. **mountedDrive.Unmount()**
      68. **Next**
      69. **' create the Windows Azure drive and its associated page blob**
      70. **Dim imageStoreDrive As CloudDrive = account.CreateCloudDrive(imageStoreBlobUri)**
      71. **Try**
      72. **imageStoreDrive.Create(16)**
      73. **Catch e1 As CloudDriveException**
      74. **' drive already exists**
      75. **End Try**
      76. **' mount the drive to retrieve its path and initialize the application with the path to the image store on the Azure drive**
      77. **Global\_asax.ImageStorePath = imageStoreDrive.Mount(cache.MaximumSizeInMegabytes / 2, DriveMountOptions.None)**
      78. **Catch driveException As CloudDriveException**
      79. **Trace.WriteLine("Error: " & driveException.Message)**
      80. **End Try**
      81. End Sub

**Note:** The preceding code retrieves the path to the local storage that you defined earlier, when you configured the Web role, and then uses this path to initialize the drive cache and set the maximum amount of local disk space it will use. Next, it creates a **CloudDrive** object specifying the URL of the page blob, which you also defined earlier in the role configuration settings. Finally, it mounts the formatted page blob to a drive letter for the Windows Azure application to start using.

Notice that the cache assigned to the drive is only half the total amount of storage reserved for the cache. Later in the exercise, you will create a second drive and assign the remainder to that drive.

* 1. If you are using VB you will also need to add the following code at the bottom of the **Global.asax.vb** code file
     1. (Code Snippet – ExploringWindowsAzureStorage-Ex04-03-ConfigurationSettingPublisher-VB)
     2. Visual Basic
     3. Public Class Global\_asax
     4. Inherits System.Web.HttpApplication
     5. . . .
     6. **Private Function ConfigurationSettingPublisher(ByVal configName As Object, ByVal configSetter As Func(Of String, Boolean)) As Object**
     7. **Dim connectionString As String = RoleEnvironment.GetConfigurationSettingValue(configName)**
     8. **configSetter(connectionString)**
     9. **Return Nothing**
     10. **End Function**
     11. End Class
  2. Make sure the following namespace directives exist at the top of the **Global.asax.cs** / **Global.asax.vb** code file.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-04-GlobalNamespace-CS*)
     2. C#
     3. **using System.Diagnostics;**
     4. **using Microsoft.WindowsAzure;**
     5. **using Microsoft.WindowsAzure.ServiceRuntime;**
     6. **using Microsoft.WindowsAzure.StorageClient;**
     7. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-04-GlobalNamespace-VB*)
     8. Visual Basic
     9. **Imports System.Diagnostics**
     10. **Imports Microsoft.WindowsAzure**
     11. **Imports Microsoft.WindowsAzure.ServiceRuntime**
     12. **Imports Microsoft.WindowsAzure.StorageClient**
  3. Next, insert the following highlighted code into the **Application\_End** method to unmount the Windows Azure Drive when the Web role shuts down. Place the code at the start of the method.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-05-ApplicationEndMethod-CS*)
     2. C#
     3. protected void Application\_End(object sender, EventArgs e)
     4. {
     5. **// obtain a reference to the cloud drive and unmount it**
     6. **CloudStorageAccount account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");**
     7. **string imageStoreBlobUri = RoleEnvironment.GetConfigurationSettingValue("ImageStoreBlobUri");**
     8. **CloudDrive imageStoreDrive = account.CreateCloudDrive(imageStoreBlobUri);**
     9. **imageStoreDrive.Unmount();**
     10. }
     11. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-05-ApplicationEndMethod-VB*)
     12. Visual Basic
     13. Sub Application\_End(ByVal sender As Object, ByVal e As EventArgs)
     14. **' obtain a reference to the cloud drive and unmount it**
     15. **Dim account As CloudStorageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString")**
     16. **Dim imageStoreBlobUri As String = RoleEnvironment.GetConfigurationSettingValue("ImageStoreBlobUri")**
     17. **Dim imageStoreDrive As CloudDrive = account.CreateCloudDrive(imageStoreBlobUri)**
     18. **imageStoreDrive.Unmount()**
     19. End Sub
     20. **Note:** The code above retrieves a reference to the previously mounted cloud drive and then unmounts it.
  4. The application is now ready to run as a Windows Azure service. Press **F5** to build and launch the application in the Compute Emulator. When the application starts, it displays the contents for the Windows Azure Drive, which is initially empty. Notice that the page shows that the Image Store Drive is mapped to a drive letter. Keep the browser window open for the moment.
     1. 
     2. Figure 41
     3. Running the application in the compute emulator with an empty image store drive
  5. Next, you determine the location of the folder used by storage emulator to simulate the cloud drive. To display the compute emulator UI, right-click the Windows Azure tray icon and then select **Show Storage Emulator UI**.
     1. 
     2. Figure 42
     3. Showing the Compute Emulator UI
  6. In the Storage Emulator UI, open the **File** menu and select **Open Azure Drive Folder in Windows Explorer**.
     1. 
     2. Figure 43
     3. Opening the Azure Drive simulation folder
     4. **Note:** When running locally, the storage emulator does not use blob storage to simulate the cloud drive. Instead, it maps the drive to a local folder. From the storage emulator UI, you can open a Windows Explorer window pointing at the temporary folder used by storage emulator to store simulated Windows Azure drives.
  7. Inside the Azure Drive folder, navigate to ***devstoreaccount1\mydrives\SamplePictures.vhd***. Note that this path matches the URI of the blob—in the storage emulator, hence the ***devstoreaccount1*** prefix—that you assigned to store the drive.
  8. Now, open the **Start** menu, select **Pictures** to open your pictures library and then double-click the **Sample Pictures** folder to open a window with sample image files.
     1. 
     2. Figure 44
     3. Sample pictures library in Windows
     4. **Note:** The sample pictures library is typically present in most Windows installations and is located at “*%PUBLIC%\Pictures\Sample Pictures”*. It contains a small set of image files. If this folder is not available in your environment, you may substitute it with any folder that contains a suitable collection of JPEG or PNG image files, for example, those found in *Assets\Images* in the *Source* folder of this lab.
  9. Copy one or more files from the pictures library to the simulated cloud drive that you determined previously.
  10. Switch back to the browser window showing the contents of the image store and refresh the page. Notice that the updated page shows the files that you copied in the previous step.
      1. 
      2. Figure 45
      3. Application showing the updated contents of the Windows Azure Drive
  11. Close the browser window.

Task 3 – Creating a New Drive in the Cloud

* 1. In this task, you update the application to create a new drive in the cloud, mount it, and then copy the contents of the original drive into it.
  2. In **Solution Explorer**, right-click **Default.aspx** and then select **View Markup**.
  3. Insert the following markup inside the body of the page, between the **h1** and **h2** heading tags, as shown below.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-06-MountedDrives Panel*)
     2. HTML

...

<body>

<form id="form1" runat="server">

<div>

<h1>PhotoAlbum</h1>

**<asp:Panel ID="SelectDrive" runat="server" Visible="false">**

**<asp:LinkButton ID="NewDrive" runat="server" Text="New Drive" onclick="NewDrive\_Click" CssClass="newdrive" />**

**Mounted Drives:**

**<asp:DropDownList ID="MountedDrives" runat="server" AutoPostBack="true"**

**DataTextField="Name" DataValueField="Value"**

**OnSelectedIndexChanged="MountedDrives\_SelectedIndexChanged" />**

**</asp:Panel>**

<h2>Image Store Drive: (<%=this.CurrentPath%>)</h2>

<asp:GridView DataSourceID="LinqDataSource1" AutoGenerateColumns="False"

...

* + 1. **Note:** The markup above displays a drop down list that enumerates the drives that the web role has mounted as well as a link button that triggers the creation of a new Windows Azure drive.
  1. In **Solution Explorer**, right-click **Default.aspx** and then select **View Code** to open its code-behind file.
  2. Add the following namespace directives at the top of the file to declare the Windows Azure supporting assemblies.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-07-AzureNamespaces-CS*)
     2. C#

**using Microsoft.WindowsAzure;**

**using Microsoft.WindowsAzure.Diagnostics;**

**using Microsoft.WindowsAzure.ServiceRuntime;**

**using Microsoft.WindowsAzure.StorageClient;**

* + 1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-07-AzureNamespaces-VB*)
    2. Visual Basic

**Imports Microsoft.WindowsAzure**

**Imports Microsoft.WindowsAzure.Diagnostics**

**Imports Microsoft.WindowsAzure.ServiceRuntime**

**Imports Microsoft.WindowsAzure.StorageClient**

* 1. Locate the **Page\_PreRender** method and insert the following highlighted code at the end of the method, as shown below.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-08-Page\_PreRender-CS*)
     2. C#

protected void Page\_PreRender(object sender, EventArgs e)

{

GridView1.Columns[GridView1.Columns.Count - 1].Visible = (this.CurrentPath != Global.ImageStorePath);

**if (RoleEnvironment.IsAvailable)**

**{**

**MountedDrives.DataSource = from item in CloudDrive.GetMountedDrives()**

**select new**

**{**

**Name = item.Key + " => " + item.Value,**

**Value = item.Key**

**};**

**MountedDrives.DataBind();**

**MountedDrives.SelectedValue = this.CurrentPath;**

**SelectDrive.Visible = true;**

**NewDrive.Text = MountedDrives.Items.Count < 2 ? "New Drive" : "Delete Drive";**

**}**

}

* + 1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-08-Page\_PreRender-VB*)
    2. Visual Basic

Protected Sub Page\_PreRender(ByVal sender As Object, ByVal e As EventArgs) Handles Me.PreRender

GridView1.Columns(GridView1.Columns.Count - 1).Visible = (Me.CurrentPath <> Global\_asax.ImageStorePath)

**If RoleEnvironment.IsAvailable Then**

**MountedDrives.DataSource = From item In CloudDrive.GetMountedDrives() \_**

**Select New With {.Name = item.Key & " => " & item.Value.ToString(), .Value = item.Key}**

**MountedDrives.DataBind()**

**MountedDrives.SelectedValue = Me.CurrentPath**

**SelectDrive.Visible = True**

**NewDrive.Text = IIf(MountedDrives.Items.Count < 2, "New Drive", "Delete Drive")**

**End If**

End Sub

* + 1. **Note:** The preceding code populates a drop down list with the drives currently mounted by the web role. The drop down shows the mapping between the page blob URI and the corresponding drive letter for the mounted drive.
  1. Add code to implement an event handler for the **New Drive** link button. To do this, paste the following code into the \_**Default** class.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-09-NewDrive\_Click-CS*)
     2. C#

**protected void NewDrive\_Click(object sender, EventArgs e)**

**{**

**if (RoleEnvironment.IsAvailable)**

**{**

**// retrieve storage account**

**CloudStorageAccount account = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");**

**// build page blob URI for the new cloud drive by changing the extension in the original URI**

**string imageStoreBlobUri = RoleEnvironment.GetConfigurationSettingValue("ImageStoreBlobUri");**

**string cloneStoreBlobUri = Path.ChangeExtension(imageStoreBlobUri, "bak");**

**// create drive and its associated page blob**

**CloudDrive clonedDrive = account.CreateCloudDrive(cloneStoreBlobUri);**

**if (MountedDrives.Items.Count < 2)**

**{**

**try**

**{**

**clonedDrive.Create(16);**

**}**

**catch (CloudDriveException)**

**{**

**// cloud drive already exists**

**}**

**// mount the drive and retrieve its path**

**LocalResource cache = RoleEnvironment.GetLocalResource("LocalDriveCache");**

**string clonedStorePath = clonedDrive.Mount(cache.MaximumSizeInMegabytes / 2, DriveMountOptions.None);**

**// copy the contents from the original drive to the new drive**

**foreach (string sourceFileName in Directory.GetFiles(Global.ImageStorePath, "\*.\*").Where(name => name.EndsWith(".jpg") || name.EndsWith(".png")))**

**{**

**string destinationFileName = Path.Combine(clonedStorePath, Path.GetFileName(sourceFileName));**

**File.Copy(sourceFileName, destinationFileName, true);**

**}**

**SelectImageStore(clonedStorePath);**

**}**

**else**

**{**

**clonedDrive.Unmount();**

**clonedDrive.Delete();**

**SelectImageStore(Global.ImageStorePath);**

**}**

**}**

**}**

* + 1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-09-NewDrive\_Click-VB*)
    2. Visual Basic

**Protected Sub NewDrive\_Click(ByVal sender As Object, ByVal e As EventArgs)**

**If RoleEnvironment.IsAvailable Then**

**' retrieve storage account**

**Dim account As CloudStorageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString")**

**' build page blob URI for the new cloud drive by changing the extension in the original URI**

**Dim imageStoreBlobUri As String = RoleEnvironment.GetConfigurationSettingValue("ImageStoreBlobUri")**

**Dim cloneStoreBlobUri As String = Path.ChangeExtension(imageStoreBlobUri, "bak")**

**' create drive and its associated page blob**

**Dim clonedDrive As CloudDrive = account.CreateCloudDrive(cloneStoreBlobUri)**

**If (MountedDrives.Items.Count < 2) Then**

**Try**

**clonedDrive.Create(16)**

**Catch e1 As CloudDriveException**

**' cloud drive already exists**

**End Try**

**' mount the drive and retrieve its path**

**Dim cache As LocalResource = RoleEnvironment.GetLocalResource("LocalDriveCache")**

**Dim clonedStorePath As String = clonedDrive.Mount(cache.MaximumSizeInMegabytes / 2, DriveMountOptions.None)**

**' copy the contents from the original drive to the new drive**

**For Each sourceFileName As String In Directory.GetFiles(Global\_asax.ImageStorePath, "\*.\*").Where(Function(name) name.EndsWith(".jpg") Or name.EndsWith(".png"))**

**Dim destinationFileName As String = Path.Combine(clonedStorePath, Path.GetFileName(sourceFileName))**

**File.Copy(sourceFileName, destinationFileName, True)**

**Next**

**SelectImageStore(clonedStorePath)**

**Else**

**clonedDrive.Unmount()**

**clonedDrive.Delete()**

**SelectImageStore(Global\_asax.ImageStorePath)**

**End If**

**End If**

**End Sub**

* + 1. **Note:** The preceding code checks if it has already mounted a second drive and if not, it creates a new NTFS formatted cloud drive and its associated page blob. It then copies the contents of the originally mounted drive into this second drive. If the drive already exists, it first unmounts it and then deletes it.
  1. Finally, add an event handler for the **SelectedIndexChanged** event of the mounted drives drop down list. To do this, insert the following method into the \_**Default** class.
     1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-10-MountedDrives\_SelectedIndexChanged-CS*)
     2. C#

**protected void MountedDrives\_SelectedIndexChanged(object sender, EventArgs e)**

**{**

**SelectImageStore(MountedDrives.SelectedValue);**

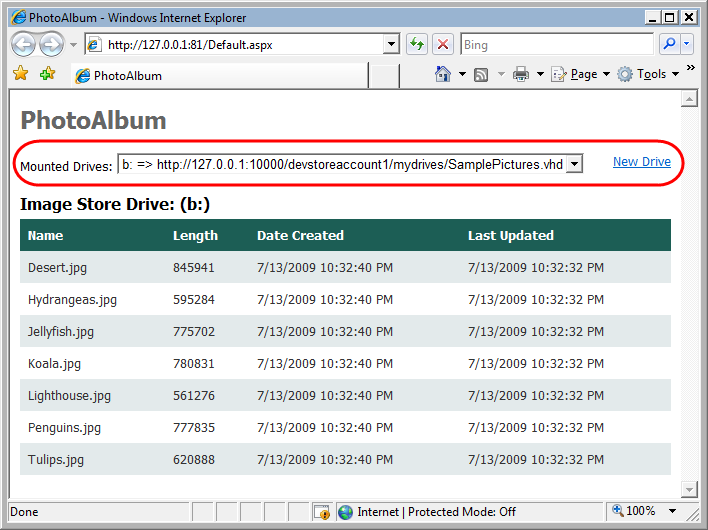
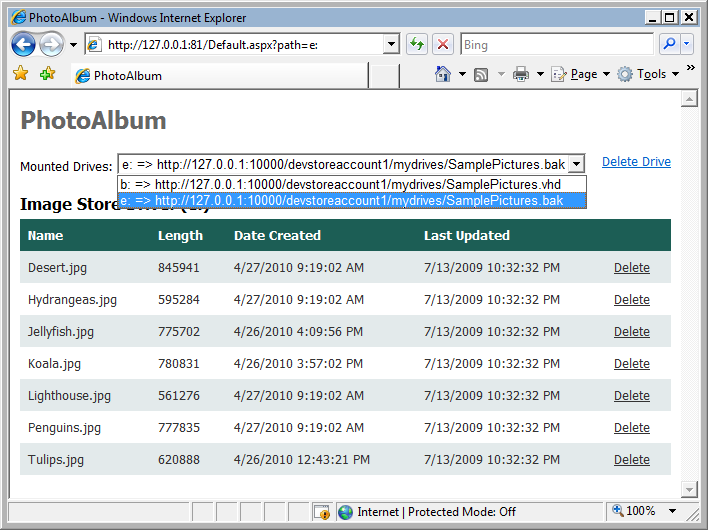
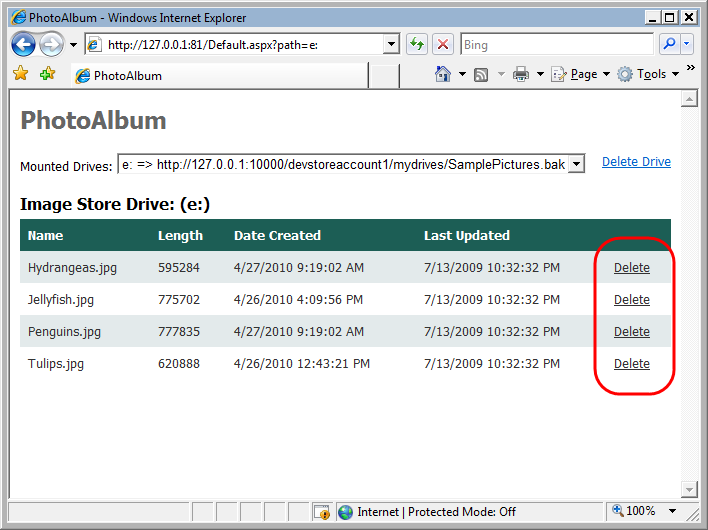
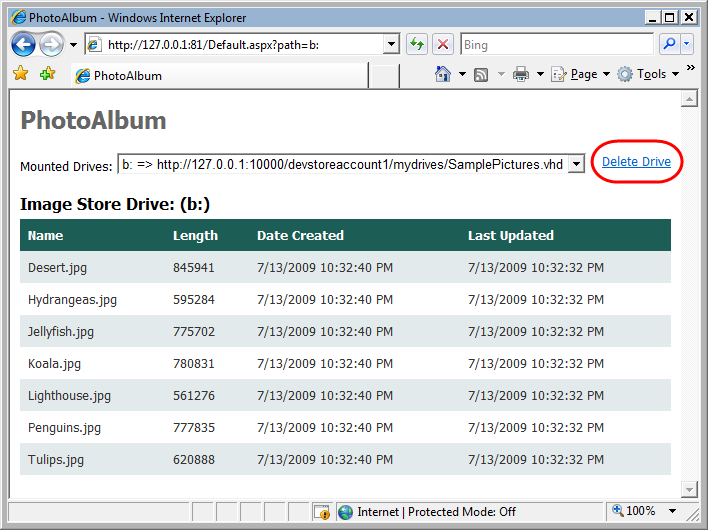
**}**

* + 1. (Code Snippet – *ExploringWindowsAzureStorage-Ex04-10-MountedDrives\_SelectedIndexChanged-VB*)
    2. Visual Basic

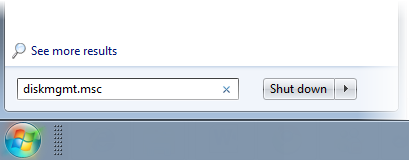
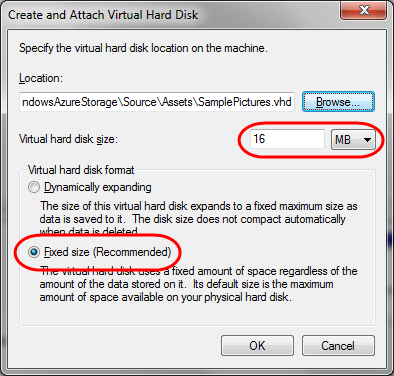
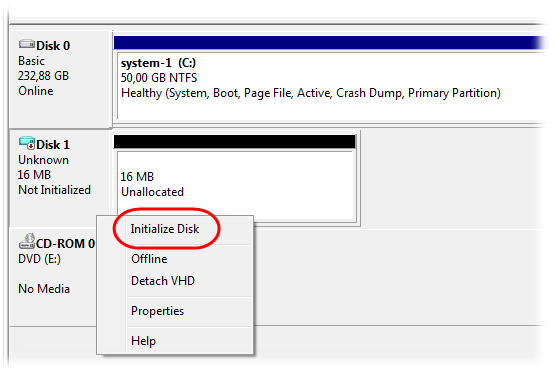
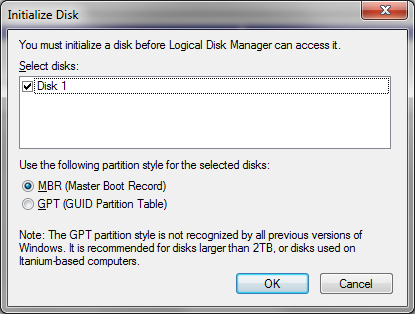
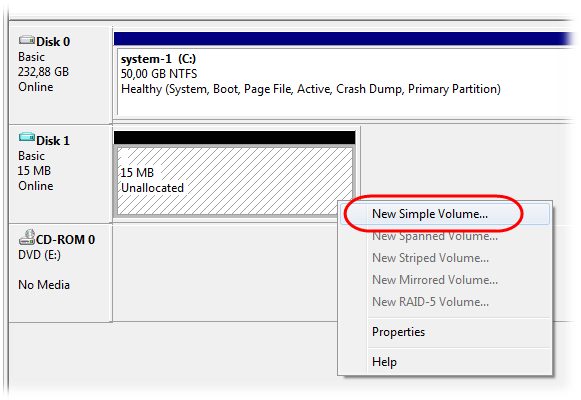
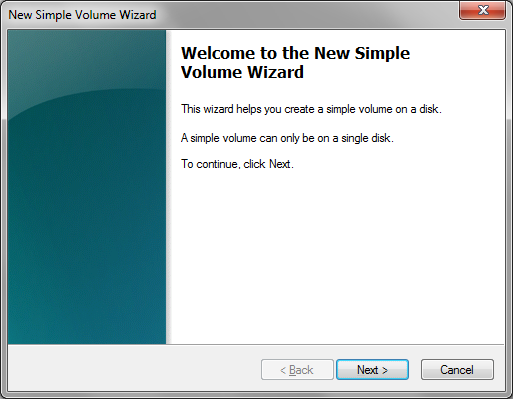
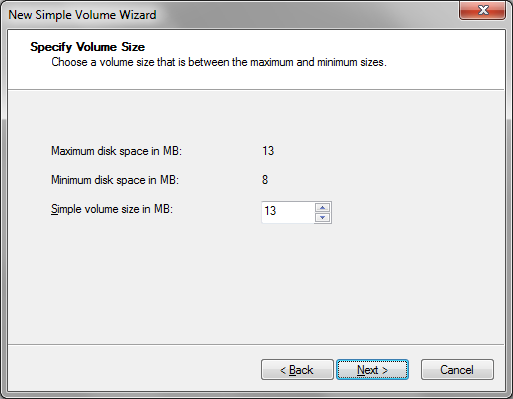
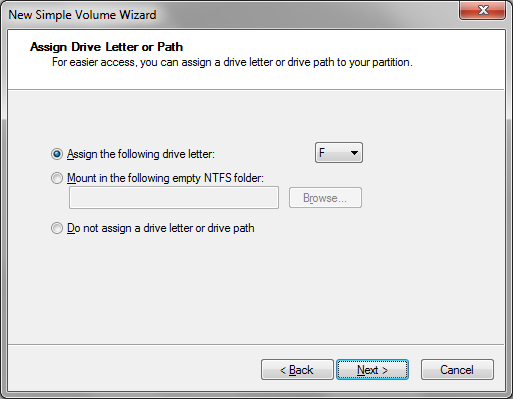
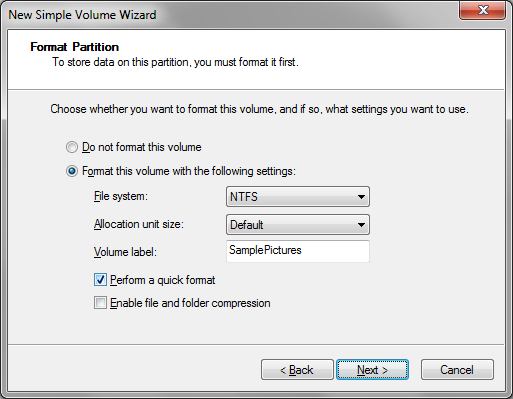
**Protected Sub MountedDrives\_SelectedIndexChanged(ByVal sender As Object, ByVal e As EventArgs)**

**SelectImageStore(MountedDrives.SelectedValue)**

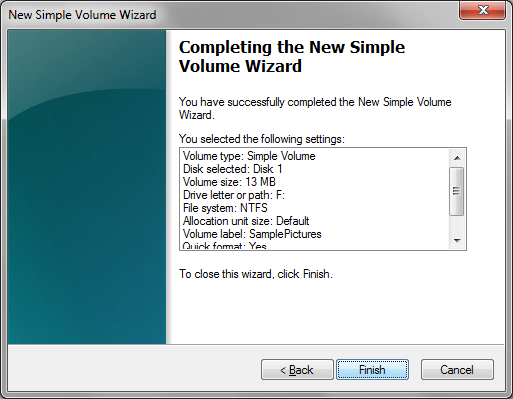
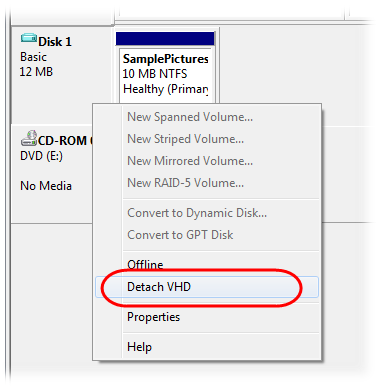
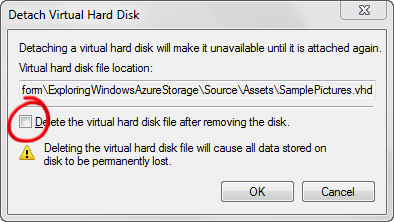
**End Sub**

* + 1. **Note:** The preceding code triggers a redirection that will refresh the contents of the page and show the directory of the newly selected drive.
  1. You are now ready to test the changes to the solution. Press **F5** to launch the application in the compute emulator. When the applications starts, notice that the **Mounted Drives** drop down list shows the letter assigned to the drive and the corresponding URL of the page blob that provides backing storage.
     1. 
     2. Figure 46
     3. Application showing currently mounted drives
  2. Click the **New Drive** link to create a new drive and copy the contents of the original drive into it. After you create the drive, the page refreshes and shows the contents of the new drive. Notice that the drop down list of mounted drives now shows the second drive.
     1. 
     2. Figure 47
     3. Application showing the newly created drive
  3. Delete one or more files in the new drive by clicking the **Delete** link.
     1. 
     2. Figure 48
     3. Deleting files in the second drive
  4. Now, choose the original drive in the **Mounted Drives** drop down list and verify that its contents are intact.
  5. Finally, click **Delete Drive** to unmount the drive and delete its corresponding page blob.
     1. 
     2. Figure 49
     3. Unmounting the drive and deleting it
     4. **Note:** This action only deletes the second drive. The original drive is not affected.
  6. Close the browser window. You will now deploy and test the application in Windows Azure.

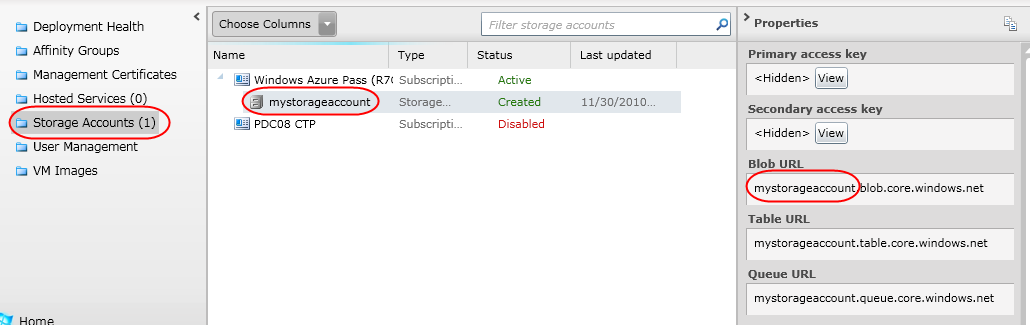
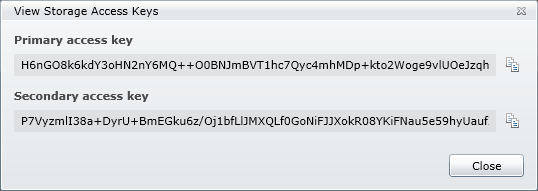
Task 4 – Creating an NTFS Formatted VHD on Your Local Machine

* 1. So far, you have explored Windows Azure Drives using the simulated environment provided by the compute emulator. When you deploy the application to the Windows Azure environment, you need a mechanism to upload the information used by the application to blob storage. One alternative is creating a Virtual Hard Drive (VHD) locally in your machine, copying the required information, and then uploading the VHD file to a Windows Azure page blob.
  2. In this task, you create an NTFS-formatted VHD file to contain sample images that you can upload to Windows Azure Storage and then use as the backing store for a Windows Azure Drive.
  3. **Note:** This task is optional and is dependent on a feature that is currently available only in Windows 7 and Windows Server 2008 R2. If you do not have these operating system versions, you may skip this task and use the pre-built VHD included in the lab’s *Assets* folder instead.
  4. **Important:** To complete the remaining tasks in this exercise, you require a valid Windows Azure subscription.
  5. For more information, visit the [Windows Azure Portal](http://www.microsoft.com/windowsazure/).
  6. Open the **Disk Management** console. To do this, click the **Start** button, type **diskmgmt.msc** in the search box, and then press **Enter** to launch this tool.
     1. 
     2. Figure 50
     3. Launching the Disk Management console
  7. In the Disk Management console, open the **Action** menu and select **Create VHD**.
  8. In the **Create and Attach Virtual Hard Disk** dialog, click **Browse** and navigate to **\Source\Ex04-WorkingWithDrives**, set the file name to **SamplePictures.vhd** and click **Save**. Next, set the **Virtual hard disk size** to *16 MB*, the **Virtual hard disk format** to **Fixed size**, and then click **OK** to create and attach the virtual hard disk.
     1. 
     2. Figure 51
     3. Creating a virtual hard disk (VHD)
  9. Before you can use the new disk, you need to initialize it. To do this, right-click the disk icon for the newly created disk in the lower pane of the **Disk Management** console and select **Initialize Disk**.
     1. 
     2. Figure 52
     3. Initializing the virtual hard disk (VHD)
  10. In the **Initialize Disk** dialog, make sure to select the disk that corresponds to the attached VHD, set the partition style to **MBR (Master Boot Record)**, and then click **OK**.
      1. 
      2. Figure 53
      3. Configuring disk initialization options
  11. Next, right-click the partition area in the attached virtual hard disk identified as unallocated and select **New Simple Volume**.
      1. 
      2. Figure 54
      3. Creating a volume in the virtual hard disk (VHD)
  12. In the **New Simple Volume Wizard**, click **Next** to dismiss the Welcome page.
      1. 
      2. Figure 55
      3. New Simple Volume Wizard welcome page
  13. Leave the **Simple volume size** unchanged—it should match the **Maximum disk space**—and click **Next** to proceed.
      1. 
      2. Figure 56
      3. Specifying the size of the disk volume
  14. Next, assign a suitable drive letter and click **Next**.
      1. 
      2. Figure 57
      3. Assigning a drive letter to the volume
  15. Now, choose to format the new partition. Set the **File system** to *NTFS*, leave the default **Allocation unit size** unchanged and set the **Volume label** to *SamplePictures*. Make sure that you check the option labeled **Perform a quick format** and leave the **Enable file and folder compression** option turned off. Click **Next** to proceed.
      1. 
      2. Figure 58
      3. Formatting the new disk partition

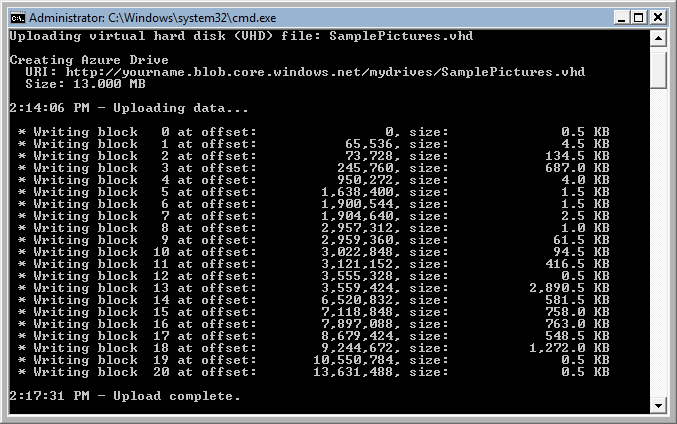
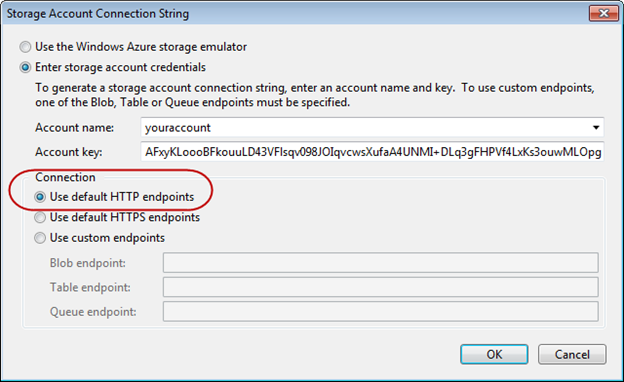
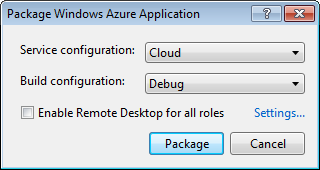
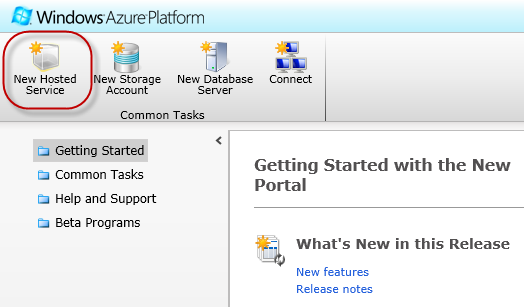
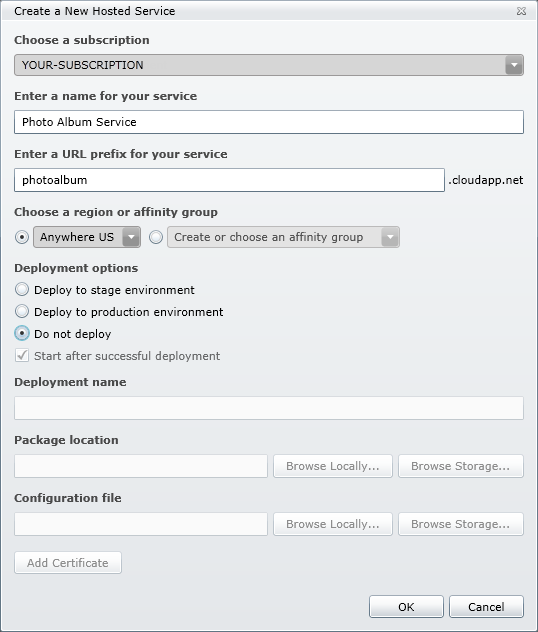
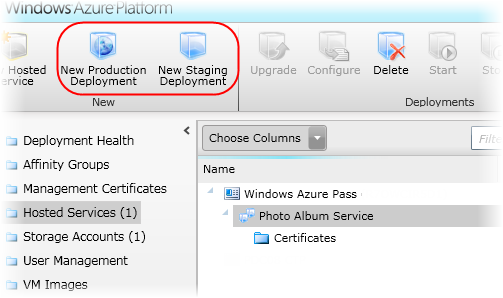
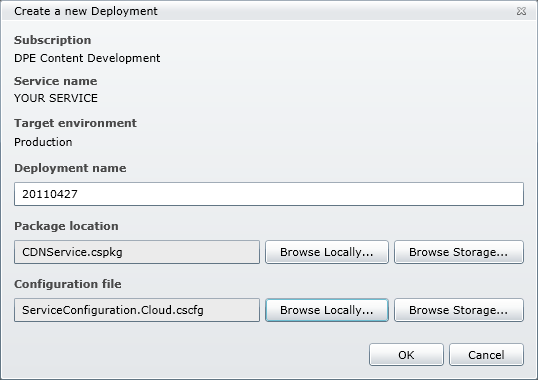
**Note:** Currently, Windows Azure Drives require an NTFS-formatted Virtual Hard Drive (VHD).

* 1. Finally, verify the information presented by the wizard in its summary screen and click **Finish** to create the new volume.
     1. 
     2. Figure 59
     3. Completing the New Simple Volume Wizard
  2. Wait for the formatting process to complete—it should only take a few seconds. If enabled, **AutoPlay** then prompts you to view the files in the newly attached disk. If that is the case, click **Open folder to view files**. Otherwise, right-click the volume in the Disk Management console and select **Open** to show the contents of the new VHD drive. Leave this new window open for the time being.
  3. Now, open the **Start** menu, select **Pictures** to open your pictures library and then browse to the **Sample Pictures** folder to view the sample image files.
     1. **Note:** If this folder is not available in your environment, you may substitute it with any folder that contains a suitable collection of JPEG or PNG image files, for example, those found in *Assets\Images* in the *Source* folder of this lab.
  4. Next, copy and paste all the files from the **Sample Pictures** folder into the first window to copy all the images contained in this folder into the VHD drive. If the drive becomes full, do not worry. You only require a few files to test the application.
  5. Switch back to the Disk Management console, right-click the attached disk—make sure to point to the disk and not the partition area—and then select **Detach VHD**.
     1. 
     2. Figure 60
     3. Preparing to detach the virtual hard disk (VHD) file
  6. In the **Detach Virtual Hard Disk** dialog, make sure not to check the option labeled **Delete the virtual hard disk file after removing the disk** and then click **OK**.
     1. 
     2. Figure 61
     3. Detaching a virtual hard disk
  7. You are now ready to upload the virtual hard disk (VHD) file to Windows Azure Storage.

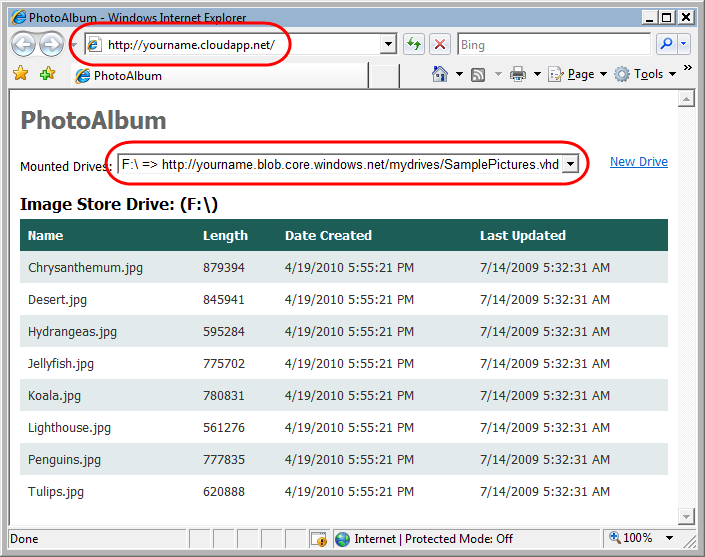
Task 5 – Deploying the Application and Uploading the Drive to Windows Azure

* 1. In this task, you upload the NTFS-formatted Virtual Hard Drive (VHD) created previously to a Windows Azure Page Blob. The lab material includes a tool that you can use for this purpose.
  2. Before you access your Windows Azure Storage account to upload the VHD file, you need to determine the name and primary key of the account. To obtain your storage account information, sign in at the [Windows Azure Management Portal](http://windows.azure.com/) and select the subscription where you will deploy your application. Choose your storage service from the list of services and record the value shown, in the Properties panel, for the **name**—this is the first segment in the endpoint URL—and the **Primary Access Key** of the storage account by clicking on the **View** button (use the **Copy to Clipboard** button to get the full access key).
     1. 
     2. Figure 62
     3. Viewing Windows Azure storage account information
     4. 
     5. Figure 63
     6. Retrieving the access keys for your storage account
  3. Next, open a command prompt and change the current directory to **ExploringWindowsAzureStorageVS2010\Source\Assets\VHDUpload**.
  4. At the command prompt, type the following command line replacing **<vhdFilePath>** with the path to the VHD file created in task 4, and **<accountName>** and **<accountKey>** with the name and the primary access key of your Windows Azure storage account , respectively.
     1. Command Prompt

**VHDUPLOAD <vhdFilePath> mydrives/SamplePictures.vhd <accountName> <accountKey>**

* + 1. **Note:** The Blob service URI that specifies the location of the .vhd file in blob storage is case sensitive. Make sure that you enter the value exactly as shown.
    2. **Note:** If you are currently on a platform that does not support creating and mounting VHD files and you were unable to complete the previous task, use the **SamplePictures.vhd** file located in the same folder as the tool instead.
  1. Press **Enter** to start uploading the virtual hard disk (VHD) file to blob storage. Wait for the process to complete; it may take several minutes.
     1. 
     2. Figure 64
     3. Uploading a VHD file to a Windows Azure page blob
  2. Now that you have uploaded the VHD file to Azure Storage, you are now ready to deploy the application.
  3. Switch back to Visual Studio.
  4. First, you need to update the web role configuration with your storage account information. To do this, expand the **Roles** node in the **PhotoAlbumService** project and double-click the **PhotoAlbum** role to open its properties window.
  5. In the **PhotoAlbum [Role]** window, switch to the **Settings** tab, locate the *Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString* setting and then click the button labeled with an ellipsis, on the right edge of the row. In the **Storage Account Connection String** dialog, choose **Enter Storage Account Credentials** and enter the **Account name** and **Account key** for your Windows Azure storage account. In the **Connection** area, make sure that the **Use default HTTPS endpoints** option is selected, and click **OK**.
  6. Repeat the previous step to configure the *DataConnectionString* setting. Enter the same information used previously except for the **Connection** setting, which should be set to **Use default HTTP endpoints**.
     1. 
     2. Figure 65
     3. Configuring the storage account for the cloud drive to use HTTP endpoints
     4. **Note:** Windows Azure Drives only support HTTP endpoints currently.
  7. Now that you have configured the application with the settings for your storage account, you can proceed to deploy it. To create a service package, right-click the **PhotoAlbumService** cloud service project and select **Package**.
     1. **Note:** For additional information about deploying your application to Windows Azure, see the **Windows Azure Deployment** hands-on lab in this training kit. In particular, Exercise 3 of this lab discusses deploying applications from Visual Studio.
     2. 
     3. Figure 66
     4. Creating a service package in Visual Studio
  8. In the **Package Windows Azure Application** dialog box, click **Package** to generate the package. This opens Windows Explorer to a folder within your solution folders that contains the generated package. Although you could use the integrated deployment feature in the Windows Azure Tools to publish the service to Windows Azure directly from Visual Studio, in this lab, you deploy it using the Windows Azure Management Portal.
  9. To deploy the service package, go to the [Management Portal](http://windows.azure.com/) and sign in.
  10. At the portal, select the project where you will deploy your application. If you have not previously created a service, you will need to create one at this time; otherwise, you may use an existing service. In that case, skip the following steps and go directly to step 20.
  11. Create the compute component that executes the application code. Click **Hosted Services** on the left pane. Click **New Hosted Service** button on the ribbon.
      1. 
      2. Figure 67
      3. Creating a new hosted service
  12. In the **Create a new Hosted Service** dialog, select the subscription where you wish to create the service from the drop down list labeled **Choose a subscription**.
  13. Enter a service name in the textbox labeled **Enter a name for your service** and choose its URL by entering a prefix in the textbox labeled **Enter a URL prefix for your service**.
  14. Select the option labeled **Choose a region** and then pick any of the available regions.
  15. Select the option labeled **Do not Deploy**.
      1. **Note:** While you can create and deploy your service to Windows Azure in a single operation by completing the **Deployment Options** section, for this hands-on lab, you will defer the deployment step until the next task.
      2. 
      3. Figure 68
      4. Configuring the hosted service
  16. Click **OK** to create the hosted service and then wait until the provisioning process completes.
  17. Choose the service where you wish to deploy the application.
  18. Now, click **New Production Deployment** or **New Staging Deployment** according to where you want to initially run the service, Production or Staging.
      1. 
      2. Figure 69
      3. Deploying an application to the hosted service
  19. In the **Create a new Deployment** dialog, you will be prompted to select the **.cspkg** and **.cscfg** files that you generated in Visual Studio: you can find them in the file system location indicated by the Windows Explorer window that opened when you published the package. Once you select both file paths, enter an arbitrary label for the deployment in the Deployment name field, for example, the current date, and then click **OK**.
  20. If prompted, ignore the warning message. This message indicates that the SLA will not apply since you selected only one instance.
      1. 
      2. Figure 70

Creating a new deployment

* + 1. **Note:** A Windows Azure virtual machine runs a guest operating system into which your service application is deployed. To support the Windows Azure Drive feature, the operating system version must be compatible with the Windows Azure SDK version 1.1 or higher. For information on available versions of the Windows Azure guest operating system, see [Windows Azure Guest OS Releases and SDK Compatibility Matrix](http://msdn.microsoft.com/en-us/library/ee924680(v=MSDN.10).aspx).
    2. In general, it is recommended to use the latest available OS to take advantage of new features and security fixes. If you do not specify a version in your configuration file, the OS upgrade method is set to automatic and Windows Azure automatically upgrades your VMs to the latest release of the guest OS, once it becomes available. For this lab, you will not choose a specific version and instead use automatic upgrade mode to ensure that the application runs under a guest OS that supports Windows Azure Drives.
  1. Wait for the transfer to complete. The status of the service should show **Ready** when completed.
  2. Now, click **DNS name** in the **Properties** pane to open the application in your browser. The application should behave in essentially the same manner as it did when you deployed it locally. However, notice that the URL of the page blob for the mounted drive is now pointing to the page blob where you uploaded the VHD file.
     1. 
     2. Figure 71
     3. Running the application in the Windows Azure environment
  3. Click the **New Drive** link and verify that you are able to create a new drive in the cloud programmatically and copy files into it. Next, test deleting some files from the cloned drive.
  4. When you are done, click **Delete Drive** to unmount the drive and delete the associated page blob.

Summary

* 1. In this lab, you have learnt how to work with the Windows Azure Storage using tables, blobs, queues and drives.
  2. Tables store data as collections of entities. Entities are similar to rows. An entity has a primary key and a set of properties. A property is a name/value pair, similar to a column.
  3. Blobs can store any binary data. In this lab, you have used the Blob service to store and show images and associate metadata using a Web application.
  4. Using queues, you have seen how to dispatch simple messages (with string, xml or binary content). This is an excellent way of enabling communication between web and worker roles.
  5. Finally, you explored the use of Windows Azure Drives that allow you to read and write data to blob storage using standard file system functions.