

Hands-On Lab

Introduction to Windows Azure

Lab version: 2.0.0

Last updated: 8/5/2011



Contents

[Overview 3](#_Toc300309434)

[Exercise 1: Building Your First Windows Azure Application 5](#_Toc300309435)

[Task 1 – Creating the Visual Studio Project 5](#_Toc300309436)

[Task 2 – Creating a Data Model for Entities in Table Storage 11](#_Toc300309437)

[Task 3 – Creating a Web Role to Display the Guest Book and Process User Input 25](#_Toc300309438)

[Task 4 – Queuing Work Items for Background Processing 35](#_Toc300309439)

[Verification 39](#_Toc300309440)

[Exercise 2: Background Processing with Worker Roles and Queues 45](#_Toc300309441)

[Task 1 – Creating a Worker Role to Process Images in the Background 45](#_Toc300309442)

[Verification 55](#_Toc300309443)

[Exercise 3: Publishing a Windows Azure Application 57](#_Toc300309444)

[Task 1 – Creating a Storage Account and a Hosted Service Component 57](#_Toc300309445)

[Task 2 – Publishing the Application to the Windows Azure Platform Management Portal 65](#_Toc300309446)

[Task 3 – Configuring the Application to Increase the Number of Instances 72](#_Toc300309447)

[Task 4 – Testing the Application in the Staging Environment 75](#_Toc300309448)

[Task 5 – Promoting the Application to Production 76](#_Toc300309449)

[Summary 79](#_Toc300309450)

Overview

A service hosted in Windows Azure consists of one or more web roles and worker roles. A web role is an ASP.NET Web application accessible via an HTTP or HTTPS endpoint and is commonly the front-end for an application. Worker roles are background-processing applications and are typically found in the back-end. Windows Azure services may be comprised of one or both types of roles and can run multiple instances of each type. Role instances can be added or removed based on demand and allow applications to quickly and economically scale-up or down when the need arises.

Windows Azure storage services provide storage in the cloud, which includes Blob services for storing text and binary data, Table services for structured storage that can be queried, and Queue services for reliable and persistent messaging between services.

* 1. In this hands-on lab, you will explore the basic elements of a Windows Azure service by creating a simple GuestBook application that demonstrates many features of the Windows Azure platform, including web and worker roles, blob storage, table storage, and queues.
  2. In the GuestBook application, a web role provides the front-end that allows users to view the contents of the guest book and submit new entries. Each entry contains a name, a message, and an associated picture. The application also contains a worker role that can generate thumbnails for the images that users submit.
  3. When users post a new item, the web role uploads the picture to blob storage and creates an entry in table storage that contains the information entered by the user and a link to the blob with the picture. The web role renders this information to the browser so users can view the contents of the guest book.
  4. After storing the image and creating the entry, the web role posts a work item to a queue to have the image processed. The worker role fetches the work item from the queue, retrieves the image from blob storage, and resizes it to create a thumbnail. Using queues to post work items is a common pattern in cloud applications and enables the separation of compute-bound tasks from the front-end. The advantage of this approach is that front and back ends can be scaled independently.

# Objectives

* 1. In this hands-on lab, you will learn how to:
  + Create applications in Windows Azure using web roles and worker roles
  + Use Windows Azure storage services including blobs, queues and tables
  + Publish an application to Windows Azure
  1. **Note:** For a more advanced scenario based on the GuestBook application that shows how to deal with poison messages in Azure queues and how to design an Azure role to make use of plug-in components that can be dynamically loaded, see the **Asynchronous Workload Handling** lab in this training kit.

# Prerequisites

* 1. The following is required to complete this hands-on lab:
  + IIS 7 (with ASP.NET, WCF HTTP Activation)
  + [Microsoft Visual Studio 2010](http://msdn.microsoft.com/vstudio/products/)
  + Microsoft .NET Framework 4.0
  + [Windows Azure Tools for Microsoft Visual Studio 1.4](http://go.microsoft.com/fwlink/?LinkID=128752)
  + [SQL Server 2005 Express Edition (or later)](http://www.microsoft.com/express/sql/download/)

# 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.

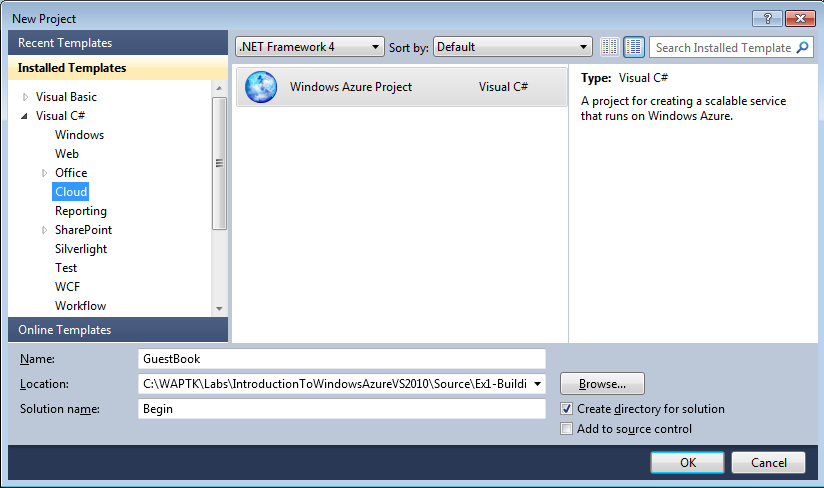
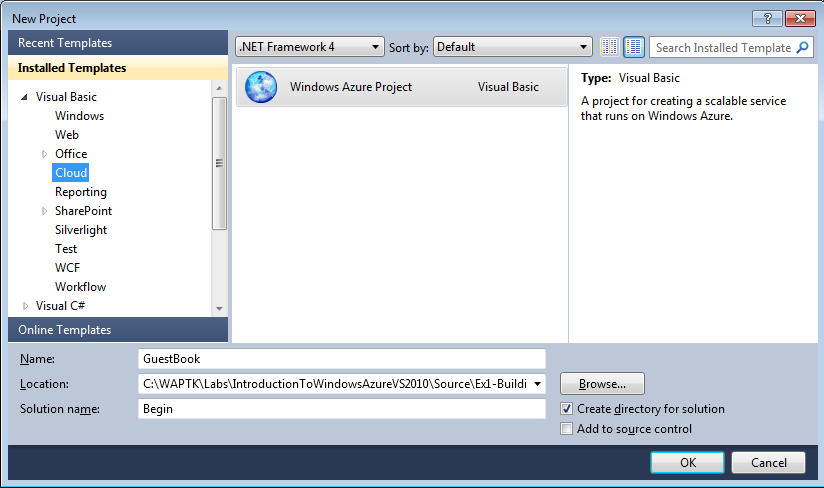
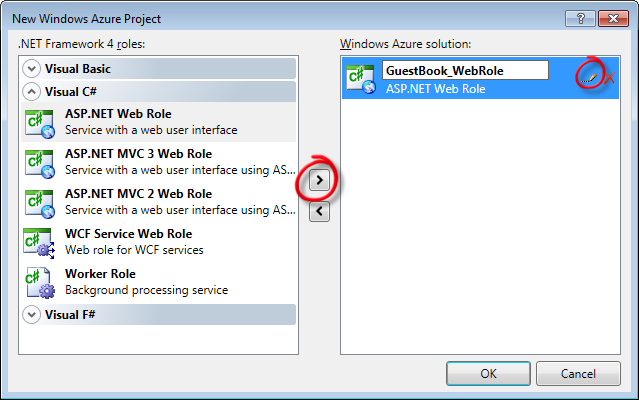
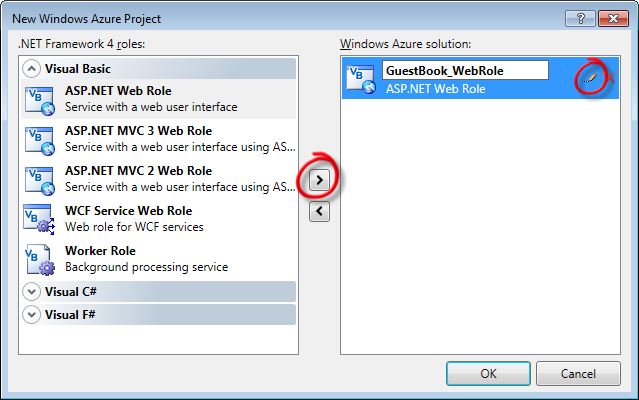
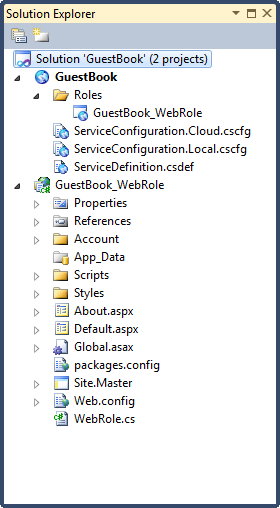
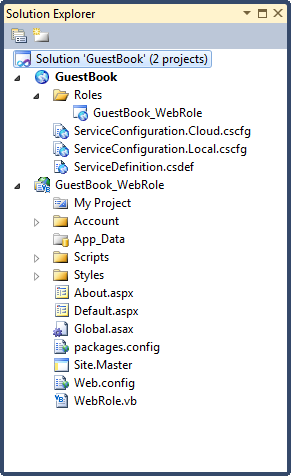
# Exercises

* 1. This hands-on lab includes the following exercises:
  + Building Your First Windows Azure Application
  + Background Processing with Worker Roles and Queues
  + Publishing a Windows Azure Application
  1. Estimated time to complete this lab: **75 minutes**.
  2. **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: Building Your First Windows Azure Application

* 1. In this exercise, you create a guest book application and execute it in the local development fabric. For this purpose, you will use the Windows Azure Tools for Microsoft Visual Studio to create the project using the Cloud Service project template. These tools extend Visual Studio to enable the creation, building and running of Windows Azure services. You will continue to work with this project throughout the remainder of the lab.
  2. **Note:** To reduce typing, you can right-click where you want to insert source code, select Insert Snippet, select My Code Snippets and then select the entry matching the current exercise step.

Task 1 – Creating the Visual Studio Project

* 1. In this task, you create a new Cloud Service project in Visual Studio.
  2. Open Visual Studioas administrator from **Start | All Programs | Microsoft Visual Studio 2010** by right clicking the **Microsoft Visual Studio 2010** shortcut and choosing **Run as administrator**.
  3. If the **User Account Control** dialog appears, click **Continue**.
  4. From the **File** menu, choose **New** and then **Project**.
  5. 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 **GuestBook**, set the location to **Ex1-BuildingYourFirstWindowsAzureApp\[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 1
     3. Creating a new Windows Azure Cloud Service project (C#)
     4. 
     5. Figure 2
     6. Creating a new Windows Azure Cloud Service project (Visual Basic)
     7. **Note:** Windows Azure supports the .NET Framework 4.0. If you use Visual Studio 2010 to create the project, you can select this version for the target framework and take advantage of its new features.
  6. In the **New Windows Azure Project** dialog, inside the **Roles** panel, expand the tab for the language of your choice (Visual C# or Visual Basic), 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. Before closing the dialog, select the new role in the right panel, click the pencil icon and rename the role as **GuestBook\_WebRole**. Click **OK** to create the cloud service solution.
     1. 
     2. Figure 3
     3. Assigning roles to the cloud service project (C#)
     4. 
     5. Figure 4
     6. Assigning roles to the cloud service project (Visual Basic)
  7. In **Solution Explorer**, review the structure of the created solution.
     1. 
     2. Figure 5
     3. Solution Explorer showing the GuestBook application (C#)
     4. 
     5. Figure 6
     6. Solution Explorer showing the GuestBook application (Visual Basic)
     7. **Note:** The generated solution contains two separate projects. The first project, named **GuestBook**, holds the configuration for the web and worker roles that compose the cloud application. It includes the service definition file, **ServiceDefinition.csdef**, which contains metadata needed by the Windows Azure fabric to understand the requirements of your application, such as which roles are used, their trust level, the endpoints exposed by each role, the local storage requirements and the certificates used by the roles. The service definition also establishes configuration settings specific to the application. The service configuration files specify the number of instances to run for each role and sets the value of configuration settings defined in the service definition file. This separation between service definition and configuration allows you to update the settings of a running application by uploading a new service configuration file.  
        You can create many configuration files, each one intended for a specific scenario such as production, development, or QA, and select which to use when publishing the application. By default, Visual Studio creates two files **ServiceConfiguration.Local.cscfg** and **ServiceConfiguration.Cloud.cscfg**.
     8. The **Roles** node in the cloud service project enables you to configure what roles the service includes (web, worker or both) as well as which projects to associate with these roles. Adding and configuring roles through the Roles node will update the **ServiceDefinition.csdef** and **ServiceConfiguration.cscfg** files.
     9. The second project, named **GuestBook\_WebRole**, is a standard ASP.NET Web Application project template modified for the Windows Azure environment. It contains an additional class that provides the entry point for the web role and contains methods to manage the initialization, starting, and stopping of the role.

Task 2 – Creating a Data Model for Entities in Table Storage

The application stores guest book entries in Windows Azure Table storage. The Table service offers semi-structured storage in the form of tables that contain collections of entities. Entities have a primary key and a set of properties, where a property is a name, typed-value pair.

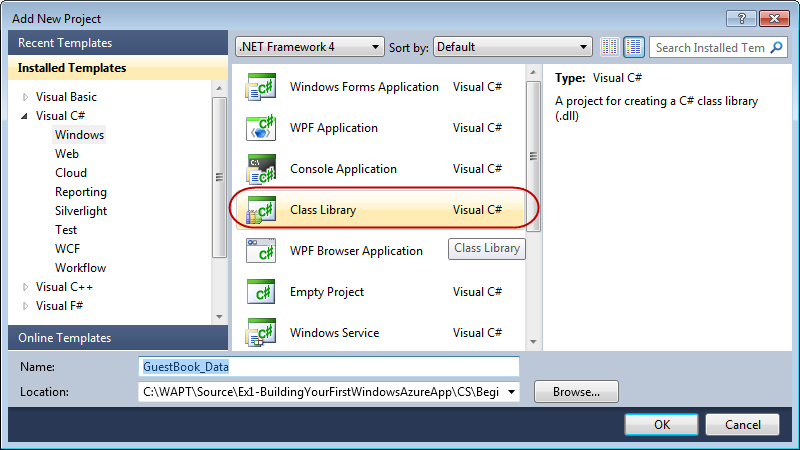
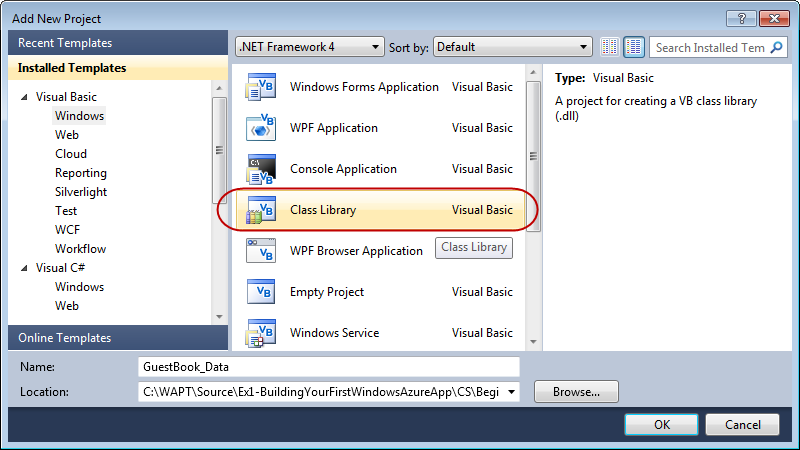
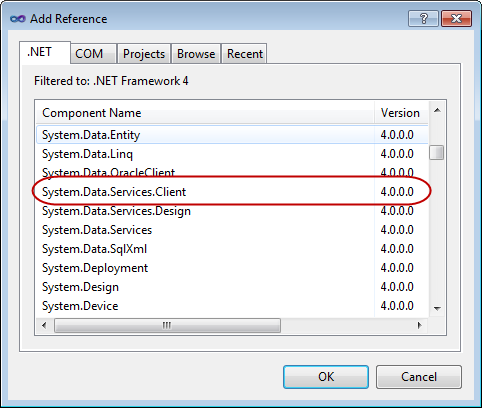
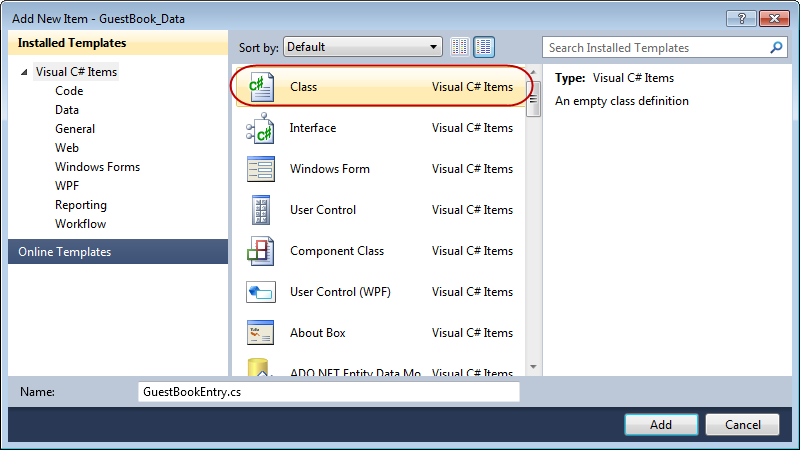
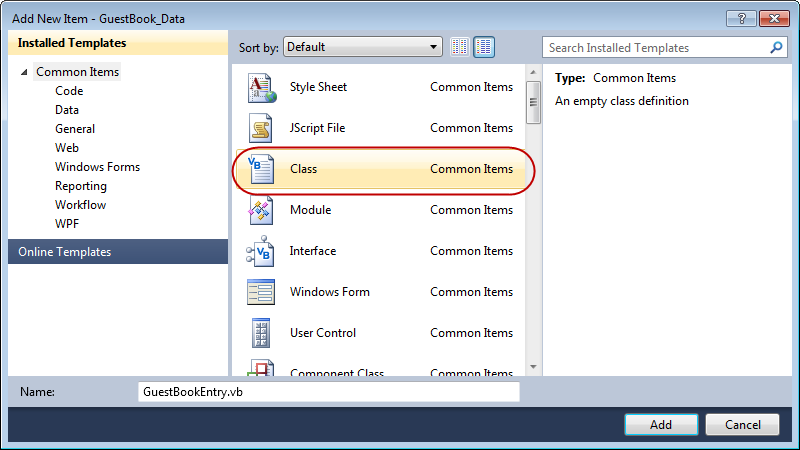
In addition to the properties required by your model, every entity in Table Storage has two key properties: the **PartitionKey** and the **RowKey**. These properties together form the table's primary key and uniquely identify each entity in the table. Entities also have a **Timestamp** system property, which allows the service to keep track of when an entity was last modified. This field is intended for system use and should not be accessed by the application. The Table Storage client API provides a **TableServiceEntity** class that defines the necessary properties. Although you can use the **TableServiceEntity** class as the base class for your entities, this is not required.

The Table service API is compliant with the REST API provided by [WCF Data Services](http://msdn.microsoft.com/en-us/library/cc668792.aspx) (formerly ADO.NET Data Services Framework) allowing you to use the [WCF Data Services Client Library](http://msdn.microsoft.com/en-us/library/cc668772.aspx) (formerly .NET Client Library) to work with data in Table Storage using .NET objects.

The Table service does not enforce any schema for tables making it possible for two entities in the same table to have different sets of properties. Nevertheless, the GuestBook application uses a fixed schema to store its data.

In order to use the WCF Data Services Client Library to access data in table storage, you need to create a context class that derives from **TableServiceContext**, which itself derives from **DataServiceContext** in WCF Data Services. The Table Storage API allows applications to create the tables that they use from these context classes. For this to happen, the context class must expose each required table as a property of type **IQueryable<*SchemaClass*>**, where ***SchemaClass*** is the class that models the entities stored in the table.

In this task, you model the schema of the entities stored by the GuestBook application and create a context class to use WCF Data Services to access the information in table storage. To complete the task, you create an object that can be data bound to data controls in ASP.NET and implements the basic data access operations: read, update, and delete.

* 1. Create a new project for the schema classes. To create the project, in the **File** menu, point to **Add** and then select **New Project**.
  2. In the **Add New Project** dialog, expand the language of your choice under the **Installed Templates** tree view, select the **Windows** category, and then choose the **Class Library** project template. Set the name to **GuestBook\_Data**, leave the proposed location inside the solution folder unchanged, and then click **OK**.
     1. 
     2. Figure 7
     3. Creating a class library for GuestBook entities (C#)
     4. 
     5. Figure 8
     6. Creating a class library for GuestBook entities (Visual Basic)
  3. Delete the default class file generated by the class library template. To do this, right-click **Class1.cs** (for Visual C# Projects) or **Class1.vb** (for Visual Basic Projects) and choose **Delete**. Click **OK** in the confirmation dialog.
  4. Add a reference to the .NET Client Library for WCF Data Services in the **GuestBook\_Data** project. In **Solution Explorer,** right-click the **GuestBook\_Data** project node, select **Add Reference**, click the **.NET** tab, select the **System.Data.Services.Client** component and click **OK**.
     1. 
     2. Figure 9
     3. Adding a reference to the System.Data.Service.Client component
  5. Repeat the previous step to add a reference to the Windows Azure storage client API assembly, this time choosing the **Microsoft.WindowsAzure.StorageClient** component instead.
  6. Before you can store an entity in a table, you must first define its schema. To do this, right-click **GuestBook\_Data** in **Solution Explorer**, point to **Add** and select **Class**. In the **Add New Item** dialog, set the name to **GuestBookEntry.cs** (for Visual C# projects) or **GuestBookEntry.vb** (for Visual Basic projects) and click **Add**.
     1. 
     2. Figure 10
     3. Adding the GuestBookEntry class (C#)
     4. 
     5. Figure 11
     6. Adding the GuestBookEntry class (Visual Basic)
  7. At the top of the file, insert the following namespace declaration to import the types contained in the **Microsoft.WindowsAzure.StorageClient** namespace.
     1. C#
     2. **using Microsoft.WindowsAzure.StorageClient;**
     3. Visual Basic
     4. **Imports Microsoft.WindowsAzure.StorageClient**
  8. If not already opened, open the **GuestBookEntry.cs** file (for Visual C# projects) or **GuestBookEntry.vb** file (for Visual Basic projects) and then update the declaration of the **GuestBookEntry** class to make it public and derive from the **TableServiceEntity** class.
     1. **Note:** In Visual Basic, the template for a new class already declares the class as Public.
     2. C#
     3. **public** class GuestBookEntry
     4. **: Microsoft.WindowsAzure.StorageClient.TableServiceEntity**
     5. {
     6. }
     7. Visual Basic
     8. **Public** Class GuestBookEntry
     9. **Inherits Microsoft.WindowsAzure.StorageClient.TableServiceEntity**
     10. End Class

**Note: TableServiceEntity** is a class found in the Storage Client API. This class defines the **PartititionKey**, **RowKey** and **TimeStamp** system properties required by every entity stored in a Windows Azure table.

* + 1. Together, the **PartitionKey** and **RowKey** define the **DataServiceKey** that uniquely identifies every entity within a table.
  1. Add a default constructor to the **GuestBookEntry** class that initializes its **PartitionKey** and **RowKey** properties.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookEntry constructor – CS*)
     2. C#
     3. **public GuestBookEntry()**
     4. **{**
     5. **PartitionKey = DateTime.UtcNow.ToString("MMddyyyy");**
     6. **// Row key allows sorting, so we make sure the rows come back in time order.**
     7. **RowKey = string.Format("{0:10}\_{1}", DateTime.MaxValue.Ticks - DateTime.Now.Ticks, Guid.NewGuid());**
     8. **}**
     9. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookEntry constructor – VB*)
     10. Visual Basic
     11. **Public Sub New()**
     12. **PartitionKey = DateTime.UtcNow.ToString("MMddyyyy")**
     13. **' Row key allows sorting, so we make sure the rows come back in time order.**
     14. **RowKey = String.Format("{0:10}\_{1}", DateTime.MaxValue.Ticks - DateTime.Now.Ticks, Guid.NewGuid())**
     15. **End Sub**
     16. **Note:** To partition the data, the GuestBook application uses the date of the entry as the **PartitionKey**, which means that there will be a separate partition for each day of guest book entries. In general, you choose the value of the partition key to ensure load balancing of the data across storage nodes.
     17. The **RowKey** is a reverse DateTime field with a GUID appended for uniqueness. Tables within partitions are sorted in RowKey order, so this will sort the tables into the correct order to be shown on the home page, with the newest entry shown at the top.
  2. To complete the definition of the **GuestBookEntry** class, add properties for **Message**, **GuestName**, **PhotoUrl**, and **ThumbnailUrl** to hold information about the entry.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 Table Schema Properties – CS*)
     2. C#
     3. **public string Message { get; set; }**
     4. **public string GuestName { get; set; }**
     5. **public string PhotoUrl { get; set; }**
     6. **public string ThumbnailUrl { get; set; }**
     7. (Code Snippet – *Introduction to Windows Azure - Ex1 Table Schema Properties – VB*)
     8. Visual Basic
     9. **Public Property Message As String**
     10. **Public Property GuestName As String**
     11. **Public Property PhotoUrl As String**
     12. **Public Property ThumbnailUrl As String**
  3. Save the **GuestBookEntry.cs** file (for Visual C# projects) or **GuestBookEntry.vb** file (for Visual Basic projects).
  4. Next, you need to create the context class required to access the *GuestBook* table using WCF Data Services. To do this, in **Solution Explorer**, right-click the **GuestBook\_Data** project, point to **Add** and select **Class**. In the **Add New Item** dialog, set the **Name** to **GuestBookDataContext.cs** (for Visual C# projects) or **GuestBookDataContext.vb** (for Visual Basic projects) and click **Add**.
  5. In the new class file, update the declaration of the new class to make it public and inherit the **TableServiceContext** class.
     1. **Note:** In Visual Basic, the template for a new class already declares the class as Public.
     2. C#
     3. **public** class GuestBookDataContext
     4. **: Microsoft.WindowsAzure.StorageClient.TableServiceContext**
     5. {
     6. }
     7. Visual Basic
     8. **Public** Class GuestBookDataContext
     9. **Inherits Microsoft.WindowsAzure.StorageClient.TableServiceContext**
     10. End Class
  6. Now, add a default constructor to initialize the base class with storage account information.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataContext Class – CS*)
     2. C#
     3. public class GuestBookDataContext
     4. : Microsoft.WindowsAzure.StorageClient.TableServiceContext
     5. {
     6. **public GuestBookDataContext(string baseAddress, Microsoft.WindowsAzure.StorageCredentials credentials)**
     7. **: base(baseAddress, credentials)**
     8. **{ }**
     9. }
     10. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataContext Class – VB*)
     11. Visual Basic
     12. Public Class GuestBookDataContext
     13. Inherits Microsoft.WindowsAzure.StorageClient.TableServiceContext
     14. **Public Sub New(ByVal baseAddress As String, ByVal credentials As Microsoft.WindowsAzure.StorageCredentials)**
     15. **MyBase.New(baseAddress, credentials)**
     16. **End Sub**
     17. End Class
     18. **Note:** You can find the **TableServiceContext** class in the storage client API. This class derives from **DataServiceContext** in WCF Data Services and manages the credentials required to access your storage account as well as providing support for a retry policy for its operations.
  7. Add a property to the **GuestBookDataContext** class to expose the **GuestBookEntry** table. To do this, insert the following (highlighted) code into the class.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookEntry Property – CS*)
     2. C#
     3. public class GuestBookDataContext
     4. : Microsoft.WindowsAzure.StorageClient.TableServiceContext
     5. {
     6. ...
     7. **public IQueryable<GuestBookEntry> GuestBookEntry**
     8. **{**
     9. **get**
     10. **{**
     11. **return this.CreateQuery<GuestBookEntry>("GuestBookEntry");**
     12. **}**
     13. **}**
     14. }
     15. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookEntry Property – VB*)
     16. Visual Basic
     17. Public Class GuestBookDataContext
     18. Inherits Microsoft.WindowsAzure.StorageClient.TableServiceContext
     19. ...
     20. **Public ReadOnly Property GuestBookEntry() As IQueryable(Of GuestBookEntry)**
     21. **Get**
     22. **Return Me.CreateQuery(Of GuestBookEntry)("GuestBookEntry")**
     23. **End Get**
     24. **End Property**
     25. End Class
     26. **Note:** You can use the **CreateTablesFromModel** method in the **CloudTableClient** class to create the tables needed by the application. When you supply a **DataServiceContext** (or **TableServiceContext**) derived class to this method, it locates any properties that return an **IQueryable<*T*>**, where the generic parameter ***T*** identifies the class that models the table schema, and creates a table in storage named after the property.
  8. Finally, you need to implement an object that can be bound to data controls in ASP.NET. In **Solution Explorer**, right-click **GuestBook\_Data**, point to **Add**, and select **Class**. In the **Add New Item** dialog, set the name to **GuestBookDataSource.cs** (for Visual C# projects) or **GuestBookDataSource.vb** (for Visual Basic projects) and click **Add**.
  9. In the new class file, add the following namespace declarations to import the types contained in the **Microsoft.WindowsAzure** and **Microsoft.WindowsAzure.StorageClient** namespaces.
     1. C#
     2. **using Microsoft.WindowsAzure;**
     3. **using Microsoft.WindowsAzure.StorageClient;**
     4. Visual Basic
     5. **Imports Microsoft.WindowsAzure**
     6. **Imports Microsoft.WindowsAzure.StorageClient**
  10. In the **GuestBookDataSource** class, make the class **public** and define member fields for the data context and the storage account information, as shown below.
      1. **Note:** In Visual Basic, the template for a new class already declares the class as Public.
      2. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource Fields – CS*)
      3. C#
      4. public class GuestBookDataSource
      5. {
      6. **private static CloudStorageAccount storageAccount;**
      7. **private GuestBookDataContext context;**

}

* + 1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource Fields – VB*)
    2. Visual Basic
    3. Public Class GuestBookDataSource
    4. **Private Shared storageAccount As CloudStorageAccount**
    5. **Private context As GuestBookDataContext**
    6. End Class
  1. Now, add a static (Shared in Visual Basic) constructor to the data source class as shown in the following (highlighted) code. This code creates the tables from the **GuestBookDataContext** class.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource Static Constructor – CS*)
     2. C#
     3. public class GuestBookDataSource
     4. {
     5. ...
     6. **static GuestBookDataSource()**
     7. **{**
     8. **storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");**
     9. **CloudTableClient.CreateTablesFromModel(**
     10. **typeof(GuestBookDataContext),**
     11. **storageAccount.TableEndpoint.AbsoluteUri,**
     12. **storageAccount.Credentials);**
     13. **}**

}

* + 1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource Shared Constructor – VB*)
    2. Visual Basic
    3. Public Class GuestBookDataSource
    4. ...
    5. **Shared Sub New()**
    6. **storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString")**
    7. **CloudTableClient.CreateTablesFromModel(GetType(GuestBookDataContext), storageAccount.TableEndpoint.AbsoluteUri, storageAccount.Credentials)**
    8. **End Sub**
    9. End Class
    10. **Note:** The static (Shared in Visual Basic) constructor initializes the storage account by reading its settings from the configuration and then uses the **CreateTablesFromModel** method in the **CloudTableClient** class to create the tables used by the application from the model defined by the **GuestBookDataContext** class. By using the static constructor, you ensure that this initialization task is executed only once.
  1. Add a default constructor to the **GuestBookDataSource** class to initialize the data context class used to access table storage.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource Constructor – CS*)
     2. C#
     3. public class GuestBookDataSource
     4. {
     5. ...
     6. **public GuestBookDataSource()**
     7. **{**
     8. **this.context = new GuestBookDataContext(storageAccount.TableEndpoint.AbsoluteUri, storageAccount.Credentials);**
     9. **this.context.RetryPolicy = RetryPolicies.Retry(3, TimeSpan.FromSeconds(1));**
     10. **}**

}

* + 1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource Constructor – VB*)
    2. Visual Basic
    3. Public Class GuestBookDataSource
    4. ...
    5. **Public Sub New()**
    6. **Me.context = New GuestBookDataContext(storageAccount.TableEndpoint.AbsoluteUri, storageAccount.Credentials)**
    7. **Me.context.RetryPolicy = RetryPolicies.Retry(3, TimeSpan.FromSeconds(1))**
    8. **End Sub**
    9. End Class
  1. Next, insert the following method to return the contents of the *GuestBookEntry* table.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource Select – CS*)
     2. C#
     3. public class GuestBookDataSource
     4. {
     5. ...
     6. **public IEnumerable<GuestBookEntry> GetGuestBookEntries()**
     7. **{**
     8. **var results = from g in this.context.GuestBookEntry**
     9. **where g.PartitionKey == DateTime.UtcNow.ToString("MMddyyyy")**
     10. **select g;**
     11. **return results;**
     12. **}**

}

* + 1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource Select – VB*)
    2. Visual Basic
    3. Public Class GuestBookDataSource
    4. ...
    5. **Public Function GetGuestBookEntries() As IEnumerable(Of GuestBookEntry)**
    6. **Dim results = From g In Me.context.GuestBookEntry \_**
    7. **Where g.PartitionKey = DateTime.UtcNow.ToString("MMddyyyy") \_**
    8. **Select g**
    9. **Return results**
    10. **End Function**
    11. End Class
    12. **Note:** The **GetGuestBookEntries** method retrieves today's guest book entries by constructing a LINQ statement that filters the retrieved information using the current date as the partition key value. The web role uses this method to bind to a data grid and display the guest book.
  1. Now, add the following method to insert new entries into the *GuestBookEntry* table.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource AddGuestBookEntry – CS*)
     2. C#
     3. public class GuestBookDataSource
     4. {
     5. ...
     6. **public void AddGuestBookEntry(GuestBookEntry newItem)**
     7. **{**
     8. **this.context.AddObject("GuestBookEntry", newItem);**
     9. **this.context.SaveChanges();**
     10. **}**

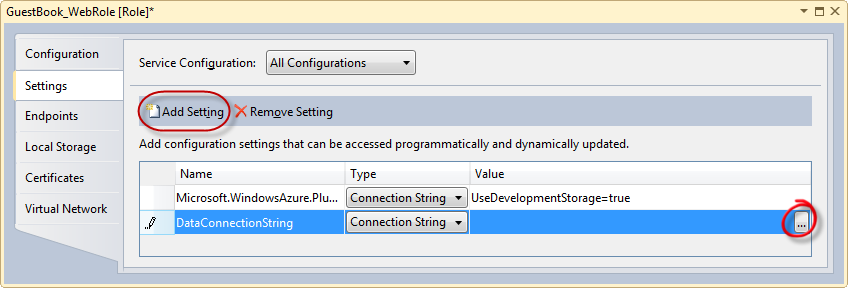
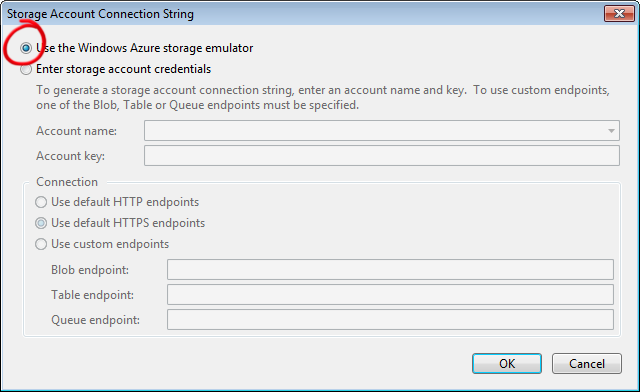
}

* + 1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource AddGuestBookEntry – VB*)
    2. Visual Basic
    3. Public Class GuestBookDataSource
    4. ...
    5. **Public Sub AddGuestBookEntry(ByVal newItem As GuestBookEntry)**
    6. **Me.context.AddObject("GuestBookEntry", newItem)**
    7. **Me.context.SaveChanges()**
    8. **End Sub**
    9. End Class
    10. **Note:** This method adds a new **GuestBookEntry** object to the data context and then calls **SaveChanges** to write the entity to storage.
  1. Finally, add a method to the data source class to update the thumbnail URL property for an entry.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource UpdateImageThumbnail – CS*)
     2. C#
     3. public class GuestBookDataSource
     4. {
     5. ...
     6. **public void UpdateImageThumbnail(string partitionKey, string rowKey, string thumbUrl)**
     7. **{**
     8. **var results = from g in this.context.GuestBookEntry**
     9. **where g.PartitionKey == partitionKey && g.RowKey == rowKey**
     10. **select g;**
     11. **var entry = results.FirstOrDefault<GuestBookEntry>();**
     12. **entry.ThumbnailUrl = thumbUrl;**
     13. **this.context.UpdateObject(entry);**
     14. **this.context.SaveChanges();**
     15. **}**

}

* + 1. (Code Snippet – *Introduction to Windows Azure - Ex1 GuestBookDataSource UpdateImageThumbnail – VB*)
    2. Visual Basic
    3. Public Class GuestBookDataSource
    4. ...
    5. **Public Sub UpdateImageThumbnail(ByVal partitionKey As String, ByVal rowKey As String, ByVal thumbUrl As String)**
    6. **Dim results = From g In Me.context.GuestBookEntry \_**
    7. **Where g.PartitionKey = partitionKey AndAlso g.RowKey = rowKey \_**
    8. **Select g**
    9. **Dim entry = results.FirstOrDefault()**
    10. **entry.ThumbnailUrl = thumbUrl**
    11. **Me.context.UpdateObject(entry)**
    12. **Me.context.SaveChanges()**
    13. **End Sub**
    14. End Class
    15. **Note:** The **UpdateImageThumbnail** method locates an entry using its partition key and row key; it updates the thumbnail URL, notifies the data context of the update, and then saves the changes.
  1. Save the **GuestBookDataSource.cs** file (for Visual C# projects) or **GuestBookDataSource.vb** file (for Visual Basic projects).

Task 3 – Creating a Web Role to Display the Guest Book and Process User Input

* 1. In this task, you update the web role project that you generated in Task 1, when you created the Windows Azure Cloud Service solution. This involves updating the UI to render the list of guest book entries. For this purpose, you will find a page that has the necessary elements in the **Assets** folder of this exercise, which you will add to the project. Next, you implement the code necessary to store submitted entries in table storage and images in blob storage. To complete this task, you configure the storage account used by the Web role.
  2. Add a reference in the web role to the **GuestBook\_Data** project. In **Solution Explorer**, right-click the **GuestBook\_WebRole** project node and select **Add Reference**, switch to the **Projects** tab, select the **GuestBook\_Data** project, and then click then **OK**.
  3. The web role template generates a default page. You will replace it with another page that contains the UI of the guest book application. To delete the page, in **Solution Explorer**, right-click **Default.aspx** in the **GuestBook\_WebRole** project and select **Delete**.
  4. Add the main page and its associated assets to the web role. To do this, right-click **GuestBook\_WebRole** in **Solution Explorer**, point to **Add** and select **Existing Item**. In the **Add Existing Item** dialog, browse to the **Assets** folder in the **Source\Ex1-BuildingYourFirstWindowsAzureApp** for the language of your project (Visual C# or Visual Basic), hold the **CTRL** key down while you select every file in this folder and click **Add**.
     1. **Note:** The **Assets** folder contains five files that you need to add to the project, a Default.aspx file with its code-behind and designer files, a CSS file, and an image file.
  5. Open the code-behind file for the main page in the **GuestBook\_WebRole** project. To do this, right-click the **Default.aspx** file in **Solution Explorer** and select **View Code**.
  6. In the code-behind file, insert the following namespace declarations.
     1. (Code Snippet *– Introduction to Windows Azure - Ex1 Web Role Namespace Declarations – CS*)
     2. C#
     3. **using System.IO;**
     4. **using System.Net;**
     5. **using Microsoft.WindowsAzure;**
     6. **using Microsoft.WindowsAzure.ServiceRuntime;**
     7. **using Microsoft.WindowsAzure.StorageClient;**
     8. **using GuestBook\_Data;**
     9. (Code Snippet *– Introduction to Windows Azure - Ex1 Web Role Namespace Declarations – VB*)
     10. Visual Basic
     11. **Imports System.IO**
     12. **Imports System.Net**
     13. **Imports Microsoft.WindowsAzure**
     14. **Imports Microsoft.WindowsAzure.ServiceRuntime**
     15. **Imports Microsoft.WindowsAzure.StorageClient**
     16. **Imports GuestBook\_Data**
  7. Declare the following member fields in the **Default** class.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 Web Role Member Fields – CS*)
     2. C#
     3. public partial class Default : System.Web.UI.Page
     4. {
     5. **private static bool storageInitialized = false;**
     6. **private static object gate = new Object();**
     7. **private static CloudBlobClient blobStorage;**
     9. ...
     10. }
     11. (Code Snippet – *Introduction to Windows Azure - Ex1 Web Role Member Fields – VB*)
     12. Visual Basic
     13. Partial Public Class \_Default
     14. Inherits System.Web.UI.Page
     15. **Private Shared storageInitialized As Boolean = False**
     16. **Private Shared gate As New Object()**
     17. **Private Shared blobStorage As CloudBlobClient**
     19. ...
     20. End Class
  8. Locate the **SignButton\_Click** event handler in the code-behind file and insert the following code.
     1. (Code Snippet *– Introduction to Windows Azure - Ex1 SignButton\_Click – CS*)
     2. C#
     3. public partial class Default : System.Web.UI.Page
     4. {
     5. ...
     6. protected void SignButton\_Click(object sender, EventArgs e)
     7. {
     8. **if (FileUpload1.HasFile)**
     9. **{**
     10. **InitializeStorage();**
     11. **// upload the image to blob storage**
     12. **string uniqueBlobName = string.Format("guestbookpics/image\_{0}{1}", Guid.NewGuid().ToString(), Path.GetExtension(FileUpload1.FileName));**
     13. **CloudBlockBlob blob = blobStorage.GetBlockBlobReference(uniqueBlobName);**
     14. **blob.Properties.ContentType = FileUpload1.PostedFile.ContentType;**
     15. **blob.UploadFromStream(FileUpload1.FileContent);**
     16. **System.Diagnostics.Trace.TraceInformation("Uploaded image '{0}' to blob storage as '{1}'", FileUpload1.FileName, uniqueBlobName);**
     17. **// create a new entry in table storage**
     18. **GuestBookEntry entry = new GuestBookEntry() { GuestName = NameTextBox.Text, Message = MessageTextBox.Text, PhotoUrl = blob.Uri.ToString(), ThumbnailUrl = blob.Uri.ToString() };**
     19. **GuestBookDataSource ds = new GuestBookDataSource();**
     20. **ds.AddGuestBookEntry(entry);**
     21. **System.Diagnostics.Trace.TraceInformation("Added entry {0}-{1} in table storage for guest '{2}'", entry.PartitionKey, entry.RowKey, entry.GuestName);**
     22. **}**
     23. **NameTextBox.Text = "";**
     24. **MessageTextBox.Text = "";**
     25. **DataList1.DataBind();**
     26. }
     27. }
     28. (Code Snippet – *Introduction to Windows Azure - Ex1 SignButton\_Click – VB*)
     29. Visual Basic
     30. Public Class \_Default
     31. Inherits System.Web.UI.Page
     32. ...
     33. Protected Sub SignButton\_Click(ByVal sender As Object, ByVal e As EventArgs) Handles SignButton.Click
     34. **If FileUpload1.HasFile Then**
     35. **InitializeStorage()**
     36. **' upload the image to blob storage**
     37. **Dim uniqueBlobName As String = String.Format("guestbookpics/image\_{0}{1}", Guid.NewGuid().ToString(), Path.GetExtension(FileUpload1.FileName))**
     38. **Dim blob As CloudBlockBlob = blobStorage.GetBlockBlobReference(uniqueBlobName)**
     39. **blob.Properties.ContentType = FileUpload1.PostedFile.ContentType**
     40. **blob.UploadFromStream(FileUpload1.FileContent)**
     41. **System.Diagnostics.Trace.TraceInformation("Uploaded image '{0}' to blob storage as '{1}'", FileUpload1.FileName, uniqueBlobName)**
     42. **' create a new entry in table storage**
     43. **Dim entry As New GuestBookEntry() With {.GuestName = NameTextBox.Text, .Message = MessageTextBox.Text, .PhotoUrl = blob.Uri.ToString(), .ThumbnailUrl = blob.Uri.ToString()}**
     44. **Dim ds As New GuestBookDataSource()**
     45. **ds.AddGuestBookEntry(entry)**
     46. **System.Diagnostics.Trace.TraceInformation("Added entry {0}-{1} in table storage for guest '{2}'", entry.PartitionKey, entry.RowKey, entry.GuestName)**
     47. **End If**
     48. **NameTextBox.Text = ""**
     49. **MessageTextBox.Text = ""**
     50. **DataList1.DataBind()**
     51. End Sub
     52. End Class
     53. **Note:** To process a new guest book entry after the user submits the page, the handler first calls the **InitializeStorage** method to ensure that the blob container used to store images exists and allows public access. You will implement this method shortly.
     54. It then obtains a reference to the blob container, generates a unique name and creates a new blob, and then uploads the image submitted by the user into this blob. Notice that the method initializes the **ContentType** property of the blob from the content type of the file submitted by the user. When the guest book page reads the blob back from storage, the response returns this content type, allowing a page to display the image contained in the blob simply by referring to its URL.
     55. After that, it creates a new **GuestBookEntry** entity, which is the entity you defined in the previous task, initializes it with the information submitted by the user, and then uses the **GuestBookDataSource** class to save the entry to table storage using the .NET Client Library for WCF Data Services.
     56. Finally, it data binds the guest book entries list to refresh its contents.
  9. Update the body of the **Timer1\_Tick** method with the code shown below.
     1. (Code Snippet *– Introduction to Windows Azure - Ex1 Timer1\_Tick – CS*)
     2. C#
     3. public partial class Default : System.Web.UI.Page
     4. {
     5. ...
     6. protected void Timer1\_Tick(object sender, EventArgs e)
     7. {
     8. **DataList1.DataBind();**
     9. }
     10. }
     11. (Code Snippet – *Introduction to Windows Azure - Ex1 Timer1\_Tick – VB*)
     12. Visual Basic
     13. Public Class \_Default
     14. Inherits System.Web.UI.Page
     15. ...
     16. Protected Sub Timer1\_Tick(ByVal sender As Object, ByVal e As EventArgs) Handles Timer1.Tick
     17. **DataList1.DataBind()**
     18. End Sub
     19. End Class
     20. **Note:** The timer periodically forces the page to refresh the contents of the guest book entries list.
  10. Locate the **Page\_Load** event handler and update its body with the following code to enable the page refresh timer.
      1. (Code Snippet – *Introduction to Windows Azure - Ex1 Page\_Load – 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. **if (!Page.IsPostBack)**
      9. **{**
      10. **Timer1.Enabled = true;**
      11. **}**
      12. }
      13. }
      14. (Code Snippet – *Introduction to Windows Azure - Ex1 Page\_Load – VB*)
      15. Visual Basic
      16. Public Class \_Default
      17. Inherits System.Web.UI.Page
      18. ...
      19. Protected Sub Page\_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
      20. **If Not (Page.IsPostBack) Then**
      21. **Timer1.Enabled = True**
      22. **End If**
      23. End Sub
      24. End Class
  11. Implement the **InitializeStorage** method by replacing its body with the following (highlighted) code.
      1. (Code Snippet – *Introduction to Windows Azure - Ex1 InitializeStorage – CS*)
      2. C#
      3. public partial class Default : System.Web.UI.Page
      4. {
      5. ...
      6. private void InitializeStorage()
      7. {
      8. **if (storageInitialized)**
      9. **{**
      10. **return;**
      11. **}**
      12. **lock (gate)**
      13. **{**
      14. **if (storageInitialized)**
      15. **{**
      16. **return;**
      17. **}**
      18. **try**
      19. **{**
      20. **// read account configuration settings**
      21. **var storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");**
      22. **// create blob container for images**
      23. **blobStorage = storageAccount.CreateCloudBlobClient();**
      24. **CloudBlobContainer container = blobStorage.GetContainerReference("guestbookpics");**
      25. **container.CreateIfNotExist();**
      26. **// configure container for public access**
      27. **var permissions = container.GetPermissions();**
      28. **permissions.PublicAccess = BlobContainerPublicAccessType.Container;**
      29. **container.SetPermissions(permissions);**
      30. **}**
      31. **catch (WebException)**
      32. **{**
      33. **throw new WebException("Storage services initialization failure. "**
      34. **+ "Check your storage account configuration settings. If running locally, "**
      35. **+ "ensure that the Development Storage service is running.");**
      36. **}**
      37. **storageInitialized = true;**
      38. **}**
      39. }
      40. }
      41. (Code Snippet – *Introduction to Windows Azure - Ex1 InitializeStorage – VB*)
      42. Visual Basic
      43. Public Class \_Default
      44. Inherits System.Web.UI.Page
      45. ...
      46. Private Sub InitializeStorage()
      47. **If storageInitialized Then**
      48. **Return**
      49. **End If**
      50. **SyncLock gate**
      51. **If storageInitialized Then**
      52. **Return**
      53. **End If**
      54. **Try**
      55. **' read account configuration settings**
      56. **Dim storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString")**
      57. **' create blob container for images**
      58. **blobStorage = storageAccount.CreateCloudBlobClient()**
      59. **Dim container As CloudBlobContainer = blobStorage.GetContainerReference("guestbookpics")**
      60. **container.CreateIfNotExist()**
      61. **' configure container for public access**
      62. **Dim permissions = container.GetPermissions()**
      63. **permissions.PublicAccess = BlobContainerPublicAccessType.Container**
      64. **container.SetPermissions(permissions)**
      65. **Catch e1 As WebException**
      66. **Throw New WebException("Storage services initialization failure. " \_**
      67. **& "Check your storage account configuration settings. If running locally, " \_**
      68. **& "ensure that the Development Storage service is running.")**
      69. **End Try**
      70. **storageInitialized = True**
      71. **End SyncLock**
      72. End Sub
      73. End Class
      74. **Note:** The **InitializeStorage** method first ensures that it executes only once. It reads the storage account settings from the Web role configuration, creates a blob container for the images uploaded with each guest book entry and configures it for public access.
  12. Because the web role uses Windows Azure storage services, you need to provide your storage account settings. To create a new setting, in **Solution Explorer**, expand the **Roles** node in the **GuestBook** project, double-click **GuestBook\_WebRole** to open the properties for this role and select the **Settings** tab. Click **Add Setting**, type “*DataConnectionString*” in the **Name** column, change the **Type** to *ConnectionString*, and then click the button labeled with an ellipsis.
      1. 
      2. Figure 12
      3. Configuring the storage account settings
  13. In the **Storage Account Connection String** dialog, choose the option labeled **Use the Windows Azure storage emulator** and then click **OK**.
      1. 
      2. Figure 13
      3. Creating a connection string for the storage emulator
      4. **Note:** A storage account is a unique endpoint for the Windows Azure Blob, Queue, and Table services. You must create a storage account in the Management Portal to use these services. In this exercise, you use Windows Azure storage emulator, which is included in the Windows Azure SDK development environment to simulate the Blob, Queue, and Table services available in the cloud. If you are building a hosted service that employs storage services or writing any external application that calls storage services, you can test locally against the Windows Azure storage emulator.
      5. To use the storage emulator, you set the value of the **UseDevelopmentStorage** keyword in the connection string for the storage account to *true*. When you publish your application to Windows Azure, you need to update the connection string to specify storage account settings including your account name and shared key. For example,
      6. <Setting name="DataConnectionString" value="DefaultEndpointsProtocol=https;AccountName=YourAccountName;AccountKey=YourAccountKey" />
      7. where *YourAccountName* is the name of your Azure Storage account and *YourAccountKey* is your access key.
  14. Press **CTRL + S** to save changes to the role configuration.
  15. Finally, you need to set up the environment for the configuration publisher. In the **GuestBook\_WebRole** project, open the **Global.asax.cs** file (for Visual C# projects) or the **Global.asax.vb** file (for Visual Basic projects).
  16. At the top of the file, insert the following namespace declaration to import the types contained in the **Microsoft.WindowsAzure** and **Microsoft.WindowsAzure.ServiceRuntime** namespaces.
      1. C#
      2. using Microsoft.WindowsAzure;
      3. using Microsoft.WindowsAzure.ServiceRuntime;
      4. Visual Basic
      5. Imports Microsoft.WindowsAzure
      6. Imports Microsoft.WindowsAzure.ServiceRuntime
  17. Insert the following code into the **Application\_Start** method replacing the default comment.
      1. (Code Snippet – *Introduction to Windows Azure - Ex1 SetConfigurationSettingPublisher – CS*)
      2. C#
      3. void Application\_Start(object sender, EventArgs e)
      4. {
      5. **Microsoft.WindowsAzure.CloudStorageAccount.SetConfigurationSettingPublisher((configName, configSetter) =>**
      6. **{**
      7. **configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));**
      8. **});**
      9. }
      10. (Code Snippet – *Introduction to Windows Azure - Ex1 SetConfigurationSettingPublisher – VB*)
      11. Visual Basic
      12. Sub Application\_Start(ByVal sender As Object, ByVal e As EventArgs)
      13. **CloudStorageAccount.SetConfigurationSettingPublisher(Function(configName, configSetter) configSetter(RoleEnvironment.GetConfigurationSettingValue(configName)))**
      14. End Sub

Task 4 – Queuing Work Items for Background Processing

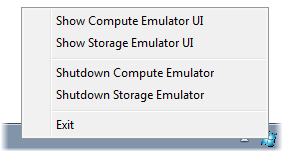
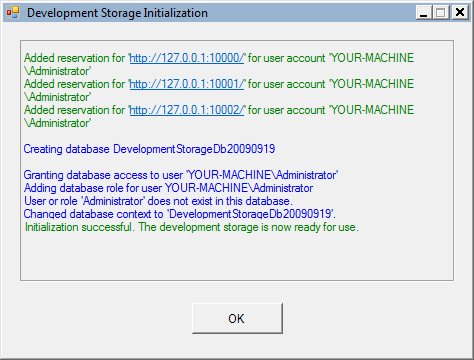
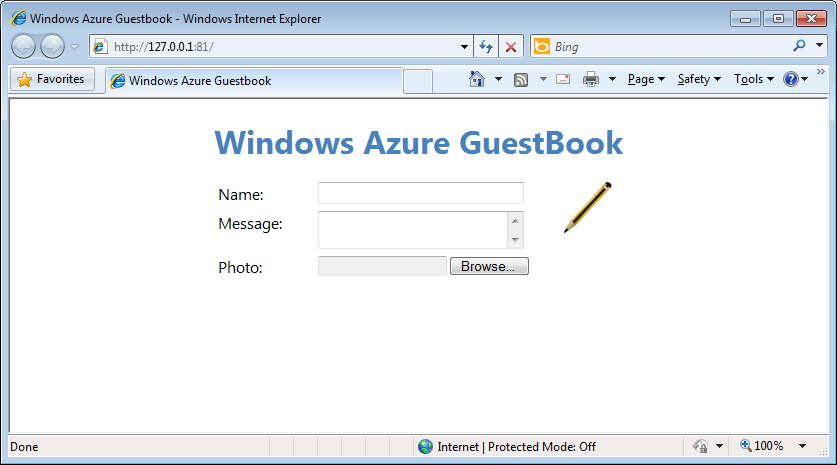
* 1. In preparation for the next exercise, you now update the front-end web role to dispatch work items to an Azure queue for background processing. These work items will remain in the queue until you add a worker role that picks items from the queue and generates thumbnails for each uploaded image.
  2. Open the code-behind file for the main page of the web role project. To do this, right-click the **Default.aspx** file in **Solution Explorer** and select **View Code**.
  3. Declare a queue client member by inserting the following (highlighted) declaration into the **Default** class.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 CloudQueueClient member – CS*)
     2. C#
     3. public partial class \_Default : System.Web.UI.Page
     4. {
     5. private static bool storageInitialized = false;
     6. private static object gate = new Object();
     7. private static CloudBlobClient blobStorage;
     8. **private static CloudQueueClient queueStorage;**
     9. ...
     10. }
     11. (Code Snippet – *Introduction to Windows Azure - Ex1 CloudQueueClient member – VB*)
     12. Visual Basic
     13. Public Class \_Default
     14. Inherits System.Web.UI.Page
     15. Private Shared storageInitialized As Boolean = False
     16. Private Shared gate As New Object()
     17. Private Shared blobStorage As CloudBlobClient
     18. **Private Shared queueStorage As CloudQueueClient**
     19. ...
     20. End Class
  4. Now, update the storage initialization code to create the queue, if it does not exist, and then initialize the queue reference created in the previous step. To do this, locate the **InitializeStorage** method and insert the following (highlighted) code into this method immediately after the code that configures the blob container for public access.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 Create Queue – CS*)
     2. C#
     3. public partial class Default : System.Web.UI.Page
     4. {
     5. ...
     6. private void InitializeStorage()
     7. {
     8. ...
     9. try
     10. {
     11. ...
     12. // configure container for public access
     13. var permissions = container.GetPermissions();
     14. permissions.PublicAccess = BlobContainerPublicAccessType.Container;
     15. container.SetPermissions(permissions);
     16. **// create queue to communicate with worker role**
     17. **queueStorage = storageAccount.CreateCloudQueueClient();**
     18. **CloudQueue queue = queueStorage.GetQueueReference("guestthumbs");**
     19. **queue.CreateIfNotExist();**
     20. }
     21. catch (WebException)
     22. {
     23. ...

}

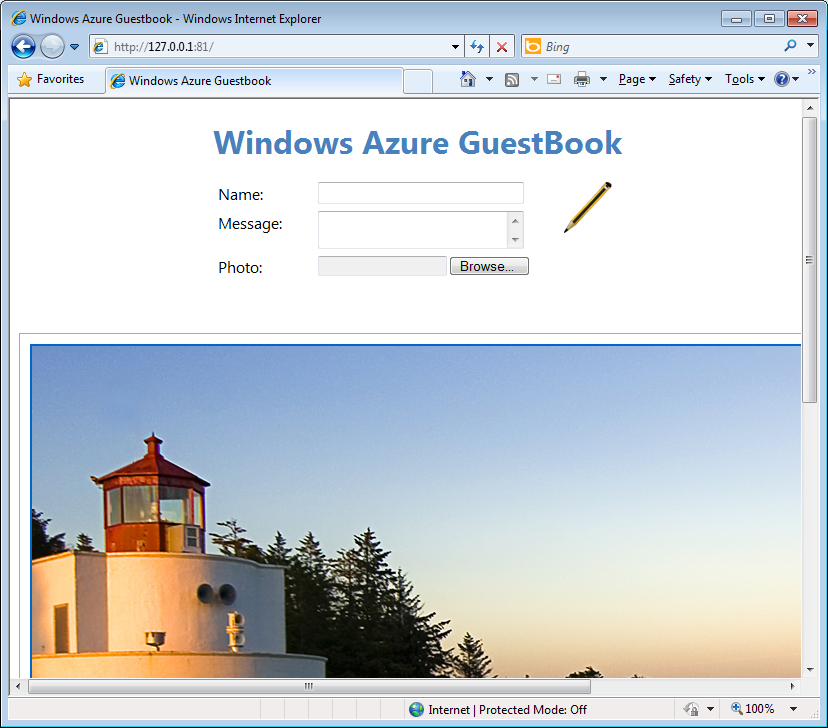
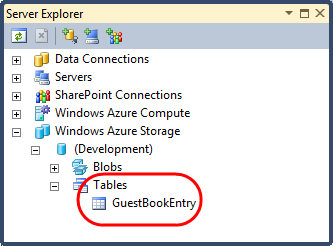
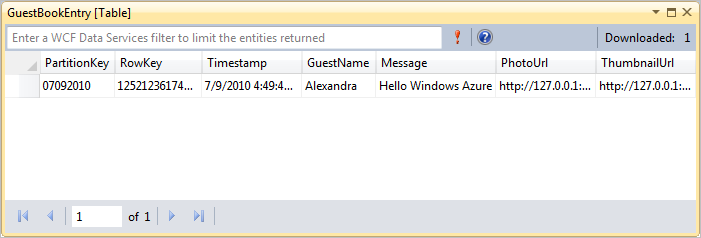
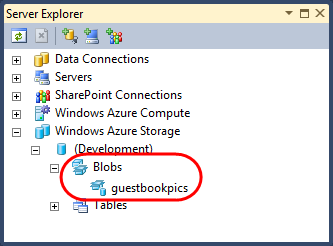
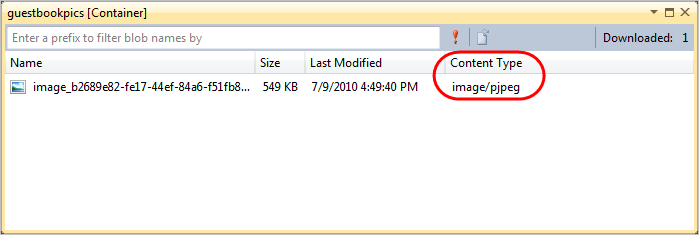
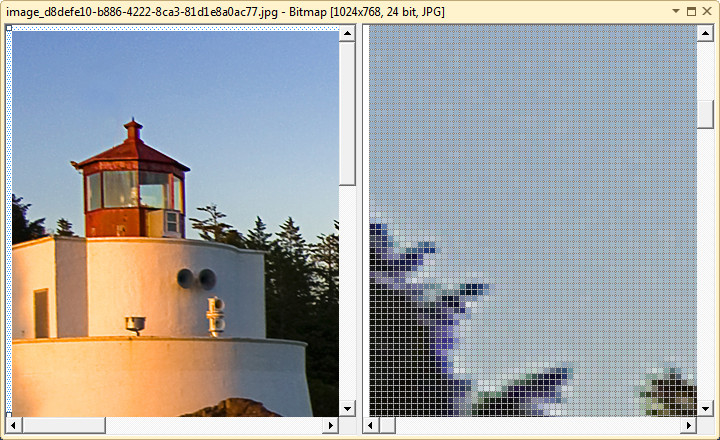
* + 1. (Code Snippet – *Introduction to Windows Azure - Ex1 Create Queue – VB*)
    2. Visual Basic
    3. Partial Public Class \_Default
    4. Inherits System.Web.UI.Page
    5. ...
    6. Private Sub InitializeStorage()
    7. ...
    8. Try
    9. ...
    10. ' configure container for public access
    11. Dim permissions = container.GetPermissions()
    12. permissions.PublicAccess = BlobContainerPublicAccessType.Container
    13. container.SetPermissions(permissions)
    14. **' create queue to communicate with worker role**
    15. **queueStorage = storageAccount.CreateCloudQueueClient()**
    16. **Dim queue As CloudQueue = queueStorage.GetQueueReference("guestthumbs")**
    17. **queue.CreateIfNotExist()**
    18. Catch e1 As WebException
    19. ...
    20. End Class
    21. **Note:** The updated code creates a queue that the web role uses to submit new jobs to the worker role.
  1. Finally, add code to post a work item to the queue. To do this, locate the **SignButton\_Click** event handler and insert the following (highlighted) code immediately after the lines that create a new entry in table storage.
     1. (Code Snippet – *Introduction to Windows Azure - Ex1 Queueing work items – CS*)
     2. C#
     3. protected void SignButton\_Click(object sender, EventArgs e)
     4. {
     5. if (FileUpload1.HasFile)
     6. {
     7. ...
     8. // create a new entry in table storage
     9. GuestBookEntry entry = new GuestBookEntry() { GuestName = NameTextBox.Text, Message = MessageTextBox.Text, PhotoUrl = blob.Uri.ToString(), ThumbnailUrl = blob.Uri.ToString() };
     10. GuestBookDataSource ds = new GuestBookDataSource();
     11. ds.AddGuestBookEntry(entry);
     12. System.Diagnostics.Trace.TraceInformation("Added entry {0}-{1} in table storage for guest '{2}'", entry.PartitionKey, entry.RowKey, entry.GuestName);
     13. **// queue a message to process the image**
     14. **var queue = queueStorage.GetQueueReference("guestthumbs");**
     15. **var message = new CloudQueueMessage(String.Format("{0},{1},{2}", blob.Uri.ToString(), entry.PartitionKey, entry.RowKey));**
     16. **queue.AddMessage(message);**
     17. **System.Diagnostics.Trace.TraceInformation("Queued message to process blob '{0}'", uniqueBlobName);**
     18. }
     19. NameTextBox.Text = "";
     20. MessageTextBox.Text = "";
     21. DataList1.DataBind();
     22. }
     23. (Code Snippet – *Introduction to Windows Azure - Ex1 Queueing work items – VB*)
     24. Visual Basic
     25. Protected Sub SignButton\_Click(ByVal sender As Object, ByVal e As EventArgs) Handles SignButton.Click
     26. If FileUpload1.HasFile Then
     27. ...
     28. ' create a new entry in table storage
     29. Dim entry As New GuestBookEntry() With {.GuestName = NameTextBox.Text, .Message = MessageTextBox.Text, .PhotoUrl = blob.Uri.ToString(), .ThumbnailUrl = blob.Uri.ToString()}
     30. Dim ds As New GuestBookDataSource()
     31. ds.AddGuestBookEntry(entry)
     32. System.Diagnostics.Trace.TraceInformation("Added entry {0}-{1} in table storage for guest '{2}'", entry.PartitionKey, entry.RowKey, entry.GuestName)
     33. **' queue a message to process the image**
     34. **Dim queue = queueStorage.GetQueueReference("guestthumbs")**
     35. **Dim message = New CloudQueueMessage(String.Format("{0},{1},{2}", blob.Uri.ToString(), entry.PartitionKey, entry.RowKey))**
     36. **queue.AddMessage(message)**
     37. **System.Diagnostics.Trace.TraceInformation("Queued message to process blob '{0}'", uniqueBlobName)**
     38. End If
     39. NameTextBox.Text = ""
     40. MessageTextBox.Text = ""
     41. DataList1.DataBind()
     42. End Sub

**Note:** The updated code obtains a reference to the “*guestthumbs*” queue. It constructs a new message that consists of a comma-separated string with the name of the blob that contains the image, the partition key, and the row key of the entity that was added. The worker role can easily parse messages with this format. The method then submits the message to the queue.

Verification

* 1. The Windows Azure compute emulator, formerly Development Fabric or devfabric, is a simulated environment for developing and testing Windows Azure applications in your machine. In this task, you launch the GuestBook application in the emulator and create one or more guest book entries.
  2. Among the features available in the Windows Azure Tools for Microsoft Visual Studio is a Windows Azure Storage browser that allows you to connect to a storage account and browse the blobs and tables it contains. If you are using this version of Visual Studio, you will use it during this task to examine the storage resources created by the application.
  3. Press **F5** to execute the service. The service builds and then launches the local Windows Azure compute emulator. To show the Compute Emulator UI, right-click its icon located in the system tray and select **Show Compute Emulator UI**.
     1. 
     2. Figure 14
     3. Showing the Compute Emulator UI
     4. **Note:** When you use the storage emulator for the first time, it needs to execute a one-time initialization procedure to create the necessary database and tables. If this is the case, wait for the procedure to complete and examine the **Development Storage Initialization** dialog to ensure that it completes successfully.
     5. 
     6. Figure 15
     7. Storage emulator initialization process
  4. Switch to Internet Explorer to view the GuestBook application.
  5. Add a new entry to the guest book. To do this, type your name and a message, choose an image to upload, and then click the pencil icon to submit the entry.
     1. 
     2. Figure 16
     3. Windows Azure GuestBook home page
     4. **Note:** It is a good idea to choose a large hi-resolution image because, once the application is complete, the guestbook service will resize uploaded images.

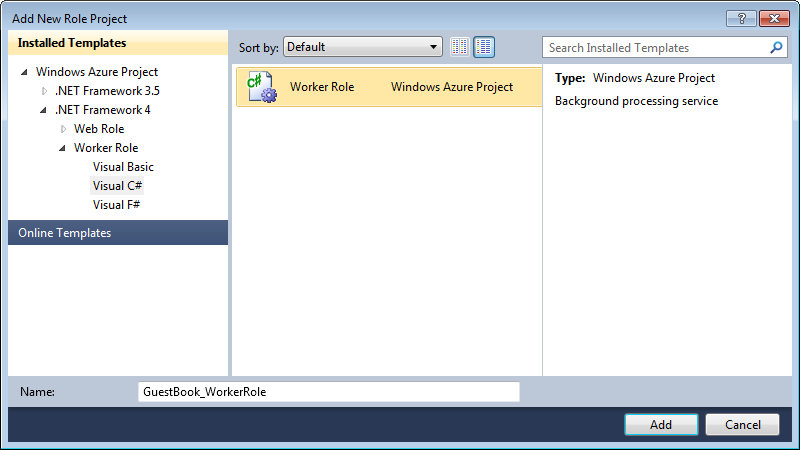
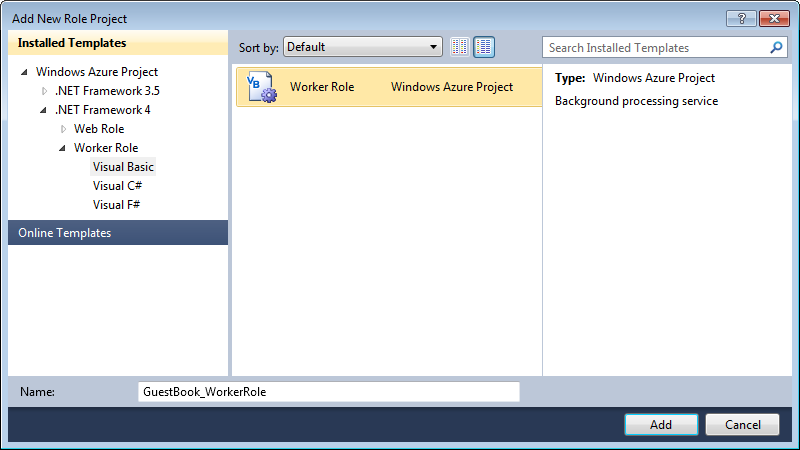
Once you submit an entry, the web role creates a new entity in the guest book table and uploads the photo to blob storage. The page contains a timer that triggers a page refresh every 5 seconds, so the new entry should appear on the page after a brief interval. Initially, the new entry contains a link to the blob that contains the uploaded image so it will appear with the same size as the original image.

* + 1. 
    2. Figure 17
    3. GuestBook application showing an uploaded image in its original size
    4. If you are using Visual Studio 2010, you can use the Windows Azure Storage Explorer to view storage resources directly from Visual Studio. This functionality is not available in Visual Studio 2008.
  1. To open the Storage Explorer in Visual Studio 2010, open the **View** menu, select **Server Explorer**, and then expand the **Windows Azure Storage** node.
     1. The **Windows Azure Storage** node lists the storage accounts that you have currently registered and, by default, includes an entry for the storage emulator account labeled as **(Development)**.
     2. **Note:** Windows Azure Storage Explorer is not available in Visual Studio 2008.
  2. Expand the **(Development)** node and then the **Tables** node inside it. Notice that it contains a table named *GuestBookEntry* created by the application that should contain details for each entry.
     1. 
     2. Figure 18
     3. Viewing tables in the Windows Azure storage emulator
  3. Double-click the *GuestBookEntry* node in the **Windows Azure Storage** explorer to show the contents of this table. The *GuestBookEntry* table contains information for the entry that you created earlier in this task, including its *GuestName*, *Message*, *PhotoUrl*, and *ThumbnailUrl* properties, as well as the *PartitionKey*, *RowKey*, and *Timestamp* properties common to all table storage entities. Notice that the *PhotoUrl* and *ThumbnailUrl* properties are currently the same. In the next exercise, you will modify the application to generate image thumbnails and to update the corresponding URL.
     1. 
     2. Figure 19
     3. Viewing tables using the Windows Azure Tools for Visual Studio
  4. Now, expand the **Blobs** node in the **Windows Azure Storage** explorer. Inside this node, you will find an entry for a container named *guestbookpics* that contains blobs with raw data for the images uploaded by the application.
     1. 
     2. Figure 20
     3. Viewing blobs using the Windows Azure Tools for Visual Studio
  5. Double-click the node for the *guestbookpics* container to list the blobs it contains. It should include an entry for the image that you uploaded earlier.
     1. 
     2. Figure 21
     3. Viewing the contents of a blob container in Visual Studio
  6. Each blob in blob storage has an associated content type that Visual Studio uses to select a suitable viewer for the blob. To display the contents of the blob, double-click the corresponding entry in the container listing to display the image.
     1. 
     2. Figure 22
     3. Viewing blob contents in Visual Studio
  7. Press **SHIFT + F5** to stop the debugger and shut down the deployment in the development fabric.

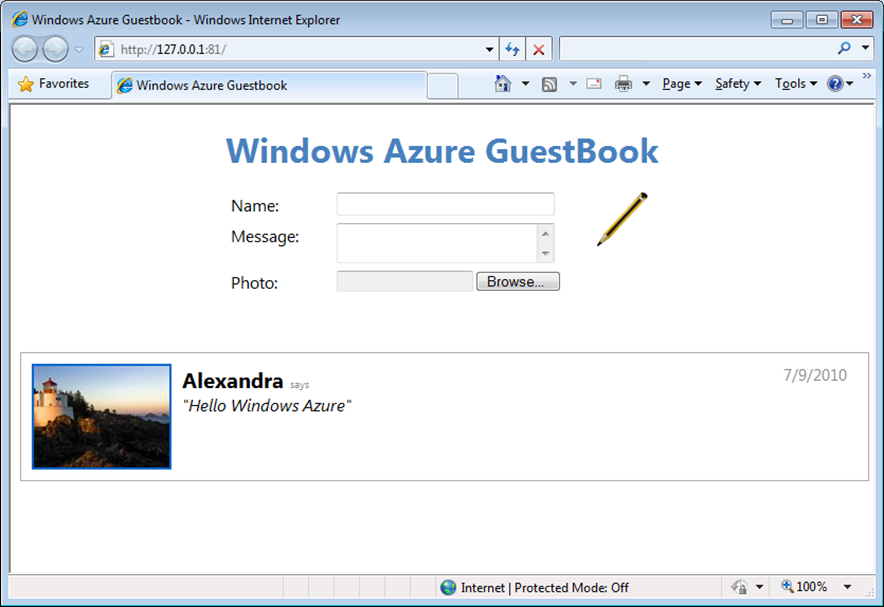
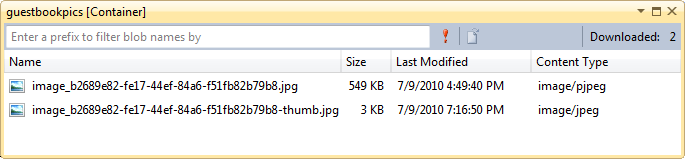
Exercise 2: Background Processing with Worker Roles and Queues

* 1. A worker role runs in the background to provide services or execute time related tasks like a service process.
  2. In this exercise, you create a worker role to read work items posted to a queue by the web role front-end. To process the work item, the worker role extracts information about a guest book entry from the message and then retrieves the corresponding entity from table storage. It then fetches the associated image from blob storage and creates its thumbnail, which it also stores as a blob. Finally, to complete the processing, it updates the URL of the generated thumbnail blob in the guest book entry.

Task 1 – Creating a Worker Role to Process Images in the Background

* 1. In this task, you add a worker role project to the solution and update it so that it reads items posted by the front-end from the queue and processes them.
  2. If not already open, launch Visual Studioas administrator from **Start | All Programs | Microsoft Visual Studio 2010** by right clicking the **Microsoft Visual Studio 2010** shortcut and choosing **Run as administrator**.
  3. In the **File** menu, choose **Open** and then **Project/Solution**. In the **Open Project** dialog, browse to **Ex2-UsingWorkerRolesAndQueues\Begin** in the **Source** folder of the lab, select **Begin.sln** in the folder for the language of your preference (Visual C# or Visual Basic) and click **Open**. Alternatively, you may continue with the solution that you obtained after completing the previous exercise.
  4. In **Solution Explorer**, right-click the **Roles** node in the **GuestBook** project, point to **Add** and then select **New Worker Role Project**.
  5. In the **Add New Role Project** dialog, select the **Worker Role** category and choose the **Worker Role** template for the language of your choice (Visual C# or Visual Basic). Set the name of the worker role to **GuestBook\_WorkerRole** and click **Add**.
     1. 
     2. Figure 23
     3. Adding a worker role project to the solution (C#)
     4. 
     5. Figure 24
     6. Adding a worker role project to the solution (Visual Basic)
  6. In the new worker role project, add a reference to the data model project. In **Solution Explorer**, right-click the **GuestBook\_WorkerRole** project and select **Add Reference**, switch to the **Projects** tab, select **GuestBook\_Data** and then click **OK**.
  7. Next, add a reference to the **System.Drawing** assembly, only this time, in the **Add Reference** dialog, switch to the **.NET** tab instead, select the **System.Drawing** component and then click **OK**.
  8. Now, open the **WorkerRole.cs** file (for Visual C# projects) or **WorkerRole.vb** file (for Visual Basic projects) of the **GuestBook\_WorkerRole** project and insert the followings namespace declarations.
     1. (Code Snippet – *Introduction to Windows Azure - Ex2 WorkerRole Namespaces – CS*)
     2. C#
     3. **using System.Drawing;**
     4. **using System.Drawing.Drawing2D;**
     5. **using System.Drawing.Imaging;**
     6. **using System.IO;**
     7. **using GuestBook\_Data;**
     8. (Code Snippet – *Introduction to Windows Azure - Ex2 WorkerRole Namespaces – VB*)
     9. Visual Basic
     10. **Imports System.Drawing**
     11. **Imports System.Drawing.Drawing2D**
     12. **Imports System.Drawing.Imaging**
     13. **Imports System.IO**
     14. **Imports GuestBook\_Data**
  9. Add member fields to the **WorkerRole** class for the blob container and the queue, as shown below.
     1. (Code Snippet – *Introduction to Windows Azure - Ex2 WorkerRole Fields – CS*)
     2. C#
     3. public class WorkerRole : RoleEntryPoint
     4. {
     5. **private CloudQueue queue;**
     6. **private CloudBlobContainer container;**
     7. ...
     8. }
     9. (Code Snippet – *Introduction to Windows Azure - Ex2 WorkerRole Fields – VB*)
     10. Visual Basic
     11. Public Class WorkerRole
     12. Inherits RoleEntryPoint
     13. **Private queue As CloudQueue**
     14. **Private container As CloudBlobContainer**
     15. ...
     16. End Class
  10. Insert the following code into the body of the **OnStart** method immediately after the line that subscribes the **RoleEnvironmentChanging** event and before the call to the **OnStart** method in the base class.
      1. (Code Snippet – *Introduction to Windows Azure - Ex2 WorkerRole OnStart – CS*)
      2. C#
      3. public class WorkerRole : RoleEntryPoint
      4. {
      5. ...
      6. public override bool OnStart()
      7. {
      8. // Set the maximum number of concurrent connections
      9. ServicePointManager.DefaultConnectionLimit = 12;
      10. **// read storage account configuration settings**
      11. **CloudStorageAccount.SetConfigurationSettingPublisher((configName, configSetter) =>**
      12. **{**
      13. **configSetter(RoleEnvironment.GetConfigurationSettingValue(configName));**
      14. **});**
      15. **var storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString");**
      16. **// initialize blob storage**
      17. **CloudBlobClient blobStorage = storageAccount.CreateCloudBlobClient();**
      18. **container = blobStorage.GetContainerReference("guestbookpics");**
      19. **// initialize queue storage**
      20. **CloudQueueClient queueStorage = storageAccount.CreateCloudQueueClient();**
      21. **queue = queueStorage.GetQueueReference("guestthumbs");**
      22. **Trace.TraceInformation("Creating container and queue...");**
      23. **bool storageInitialized = false;**
      24. **while (!storageInitialized)**
      25. **{**
      26. **try**
      27. **{**
      28. **// create the blob container and allow public access**
      29. **container.CreateIfNotExist();**
      30. **var permissions = container.GetPermissions();**
      31. **permissions.PublicAccess = BlobContainerPublicAccessType.Container;**
      32. **container.SetPermissions(permissions);**
      33. **// create the message queue(s)**
      34. **queue.CreateIfNotExist();**
      35. **storageInitialized = true;**
      36. **}**
      37. **catch (StorageClientException e)**
      38. **{**
      39. **if (e.ErrorCode == StorageErrorCode.TransportError)**
      40. **{**
      41. **Trace.TraceError("Storage services initialization failure. "**
      42. **+ "Check your storage account configuration settings. If running locally, "**
      43. **+ "ensure that the Development Storage service is running. Message: '{0}'", e.Message);**
      44. **System.Threading.Thread.Sleep(5000);**
      45. **}**
      46. **else**
      47. **{**
      48. **throw;**
      49. **}**
      50. **}**
      51. **}**
      52. return base.OnStart();
      53. }
      54. ...
      55. }
      56. (Code Snippet – *Introduction to Windows Azure - Ex2 WorkerRole OnStart – VB*)
      57. Visual Basic
      58. Public Class WorkerRole
      59. Inherits RoleEntryPoint
      60. ...
      61. Public Overrides Function OnStart() As Boolean
      62. ' Set the maximum number of concurrent connections
      63. ServicePointManager.DefaultConnectionLimit = 12
      64. **' read storage account configuration settings**
      65. **CloudStorageAccount.SetConfigurationSettingPublisher(Function(configName, configSetter) configSetter(RoleEnvironment.GetConfigurationSettingValue(configName)))**
      66. **Dim storageAccount = CloudStorageAccount.FromConfigurationSetting("DataConnectionString")**
      67. **' initialize blob storage**
      68. **Dim blobStorage = storageAccount.CreateCloudBlobClient()**
      69. **container = blobStorage.GetContainerReference("guestbookpics")**
      70. **' initialize queue storage**
      71. **Dim queueStorage = storageAccount.CreateCloudQueueClient()**
      72. **queue = queueStorage.GetQueueReference("guestthumbs")**
      73. **Trace.TraceInformation("Creating container and queue...")**
      74. **Dim storageInitialized = False**
      75. **Do While (Not storageInitialized)**
      76. **Try**
      77. **' create the blob container and allow public access**
      78. **container.CreateIfNotExist()**
      79. **Dim permissions = container.GetPermissions()**
      80. **permissions.PublicAccess = BlobContainerPublicAccessType.Container**
      81. **container.SetPermissions(permissions)**
      82. **' create the message queue(s)**
      83. **queue.CreateIfNotExist()**
      84. **storageInitialized = True**
      85. **Catch e As StorageClientException**
      86. **If (e.ErrorCode = StorageErrorCode.TransportError) Then**
      87. **Trace.TraceError("Storage services initialization failure. " \_**
      88. **& "Check your storage account configuration settings. If running locally, " \_**
      89. **& "ensure that the Development Storage service is running. Message: '{0}'", e.Message)**
      90. **System.Threading.Thread.Sleep(5000)**
      91. **Else**
      92. **Throw**
      93. **End If**
      94. **End Try**
      95. **Loop**
      96. Return MyBase.OnStart()
      97. End Function
      98. ...
      99. End Class
  11. Replace the body of the **Run** method with the code shown below.
      1. (Code Snippet – *Introduction to Windows Azure - Ex2 WorkerRole Run – CS*)
      2. C#
      3. public class WorkerRole : RoleEntryPoint
      4. {
      5. ...
      6. public override void Run()
      7. {
      8. **Trace.TraceInformation("Listening for queue messages...");**
      9. **while (true)**
      10. **{**
      11. **try**
      12. **{**
      13. **// retrieve a new message from the queue**
      14. **CloudQueueMessage msg = queue.GetMessage();**
      15. **if (msg != null)**
      16. **{**
      17. **// parse message retrieved from queue**
      18. **var messageParts = msg.AsString.Split(new char[] { ',' });**
      19. **var imageBlobUri = messageParts[0];**
      20. **var partitionKey = messageParts[1];**
      21. **var rowkey = messageParts[2];**
      22. **Trace.TraceInformation("Processing image in blob '{0}'.", imageBlobUri);**
      23. **string thumbnailBlobUri = System.Text.RegularExpressions.Regex.Replace(imageBlobUri, "([^\\.]+)(\\.[^\\.]+)?$", "$1-thumb$2");**
      24. **CloudBlob inputBlob = container.GetBlobReference(imageBlobUri);**
      25. **CloudBlob outputBlob = container.GetBlobReference(thumbnailBlobUri);**
      26. **using (BlobStream input = inputBlob.OpenRead())**
      27. **using (BlobStream output = outputBlob.OpenWrite())**
      28. **{**
      29. **ProcessImage(input, output);**
      30. **// commit the blob and set its properties**
      31. **output.Commit();**
      32. **outputBlob.Properties.ContentType = "image/jpeg";**
      33. **outputBlob.SetProperties();**
      34. **// update the entry in table storage to point to the thumbnail**
      35. **GuestBookDataSource ds = new GuestBookDataSource();**
      36. **ds.UpdateImageThumbnail(partitionKey, rowkey, thumbnailBlobUri);**
      37. **// remove message from queue**
      38. **queue.DeleteMessage(msg);**
      39. **Trace.TraceInformation("Generated thumbnail in blob '{0}'.", thumbnailBlobUri);**
      40. **}**
      41. **}**
      42. **else**
      43. **{**
      44. **System.Threading.Thread.Sleep(1000);**
      45. **}**
      46. **}**
      47. **catch (StorageClientException e)**
      48. **{**
      49. **Trace.TraceError("Exception when processing queue item. Message: '{0}'", e.Message);**
      50. **System.Threading.Thread.Sleep(5000);**
      51. **}**
      52. **}**
      53. }
      54. ...
      55. }
      56. (Code Snippet – *Introduction to Windows Azure - Ex2 WorkerRole Run – VB*)
      57. Visual Basic
      58. Public Class WorkerRole
      59. Inherits RoleEntryPoint
      60. ...
      61. Public Overrides Sub Run()
      62. **Trace.TraceInformation("Listening for queue messages...")**
      63. **Do**
      64. **Try**
      65. **' retrieve a new message from the queue**
      66. **Dim msg As CloudQueueMessage = queue.GetMessage()**
      67. **If msg IsNot Nothing Then**
      68. **' parse message retrieved from queue**
      69. **Dim messageParts = msg.AsString.Split(New Char() {","c})**
      70. **Dim imageBlobUri = messageParts(0)**
      71. **Dim partitionKey = messageParts(1)**
      72. **Dim rowKey = messageParts(2)**
      73. **Trace.TraceInformation("Processing image in blob '{0}'.", imageBlobUri)**
      74. **Dim thumbnailBlobUri As String = System.Text.RegularExpressions.Regex.Replace(imageBlobUri, "([^\\.]+)(\\.[^\\.]+)?$", "$1-thumb$2")**
      75. **' download original image from blob storage**
      76. **Dim inputBlob As CloudBlockBlob = container.GetBlockBlobReference(imageBlobUri)**
      77. **Dim outputBlob As CloudBlockBlob = container.GetBlockBlobReference(thumbnailBlobUri)**
      78. **Using input As BlobStream = inputBlob.OpenRead()**
      79. **Using output As BlobStream = outputBlob.OpenWrite()**
      80. **ProcessImage(input, output)**
      81. **' commit the blob and set its properties**
      82. **output.Commit()**
      83. **outputBlob.Properties.ContentType = "image/jpeg"**
      84. **outputBlob.SetProperties()**
      85. **' update the entry in table storage to point to the thumbnail**
      86. **Dim ds = New GuestBookDataSource()**
      87. **ds.UpdateImageThumbnail(partitionKey, rowKey, thumbnailBlobUri)**
      88. **' remove message from queue**
      89. **queue.DeleteMessage(msg)**
      90. **Trace.TraceInformation("Generated thumbnail in blob '{0}'.", thumbnailBlobUri)**
      91. **End Using**
      92. **End Using**
      93. **Else**
      94. **System.Threading.Thread.Sleep(1000)**
      95. **End If**
      96. **Catch e As StorageClientException**
      97. **Trace.TraceError("Exception when processing queue item. Message: '{0}'", e.Message)**
      98. **System.Threading.Thread.Sleep(5000)**
      99. **End Try**
      100. **Loop**
      101. End Sub
      102. ...
      103. End Class
  12. Finally, add the following method to the **WorkerRole** class to create thumbnails from a given image.
      1. (Code Snippet – *Introduction to Windows Azure -* *Ex2 ProcessImage – CS*)
      2. C#
      3. public class WorkerRole : RoleEntryPoint
      4. {
      5. ...
      6. **public void ProcessImage(Stream input, Stream output)**
      7. **{**
      8. **int width;**
      9. **int height;**
      10. **var originalImage = new Bitmap(input);**
      11. **if (originalImage.Width > originalImage.Height)**
      12. **{**
      13. **width = 128;**
      14. **height = 128 \* originalImage.Height / originalImage.Width;**
      15. **}**
      16. **else**
      17. **{**
      18. **height = 128;**
      19. **width = 128 \* originalImage.Width / originalImage.Height;**
      20. **}**
      21. **var thumbnailImage = new Bitmap(width, height);**
      22. **using (Graphics graphics = Graphics.FromImage(thumbnailImage))**
      23. **{**
      24. **graphics.InterpolationMode = InterpolationMode.HighQualityBicubic;**
      25. **graphics.SmoothingMode = SmoothingMode.AntiAlias;**
      26. **graphics.PixelOffsetMode = PixelOffsetMode.HighQuality;**
      27. **graphics.DrawImage(originalImage, 0, 0, width, height);**
      28. **}**
      29. **thumbnailImage.Save(output, ImageFormat.Jpeg);**
      30. **}**
      31. }
      32. (Code Snippet – *Introduction to Windows Azure - Ex2 ProcessImage – VB*)
      33. Visual Basic
      34. Public Class WorkerRole
      35. Inherits RoleEntryPoint
      36. ...
      37. **Private Sub ProcessImage(ByVal input As Stream, ByVal output As Stream)**
      38. **Dim width As Integer**
      39. **Dim height As Integer**
      40. **Dim originalImage As New Bitmap(input)**
      41. **If originalImage.Width > originalImage.Height Then**
      42. **width = 128**
      43. **height = 128 \* originalImage.Height / originalImage.Width**
      44. **Else**
      45. **height = 128**
      46. **width = 128 \* originalImage.Width / originalImage.Height**
      47. **End If**
      48. **Dim thumbnailImage As New Bitmap(width, height)**
      49. **Using graphic = Graphics.FromImage(thumbnailImage)**
      50. **graphic.InterpolationMode = InterpolationMode.HighQualityBicubic**
      51. **graphic.SmoothingMode = SmoothingMode.AntiAlias**
      52. **graphic.PixelOffsetMode = PixelOffsetMode.HighQuality**
      53. **graphic.DrawImage(originalImage, 0, 0, width, height)**
      54. **End Using**
      55. **thumbnailImage.Save(output, ImageFormat.Jpeg)**
      56. **End Sub**
      57. End Class
      58. **Note:** Even though the code shown above uses classes in the System.Drawing namespace for simplicity, you should be aware that the classes in this namespace were designed for use with Windows Forms. They are not supported for use within a Windows or ASP.NET service. You should conduct exhaustive testing if you intend to use these classes in your own Windows Azure applications.
  13. The worker role also uses Windows Azure storage services and you need to configure your storage account settings, just as you did in the case of the web role. To create the storage account setting, in **Solution Explorer**, expand the **Roles** node of the **GuestBook** project, double-click **GuestBook\_WorkerRole** to open the properties for this role and select the **Settings** tab. Click **Add Setting**, type “*DataConnectionString*” in the **Name** column, change the **Type** to *ConnectionString*, and then click the button labeled with an ellipsis. In the **Storage Account Connection String** dialog, choose the option labeled **Use the Windows Azure storage emulator** and click **OK**. Press **CTRL + S** to save your changes.

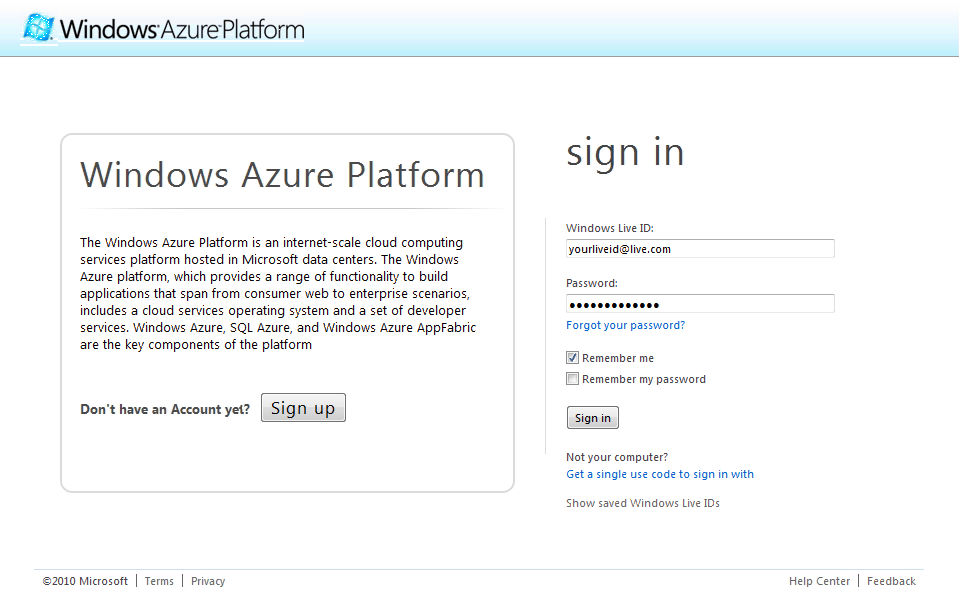
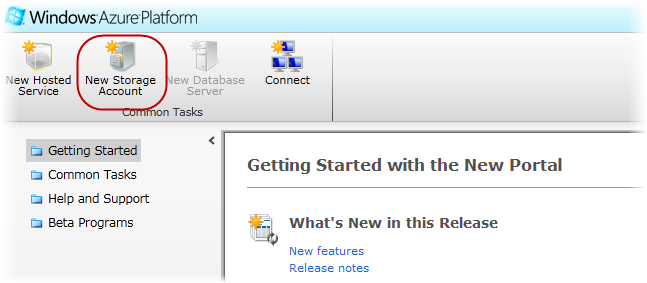
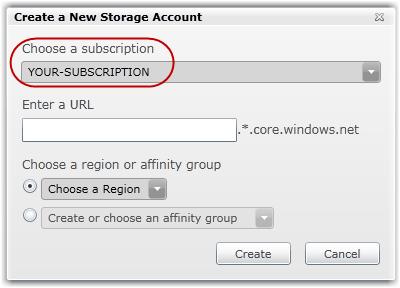
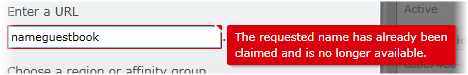
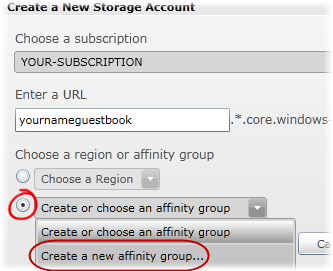
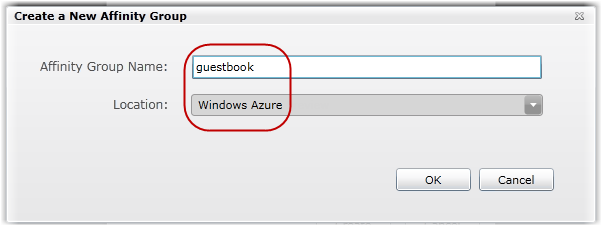
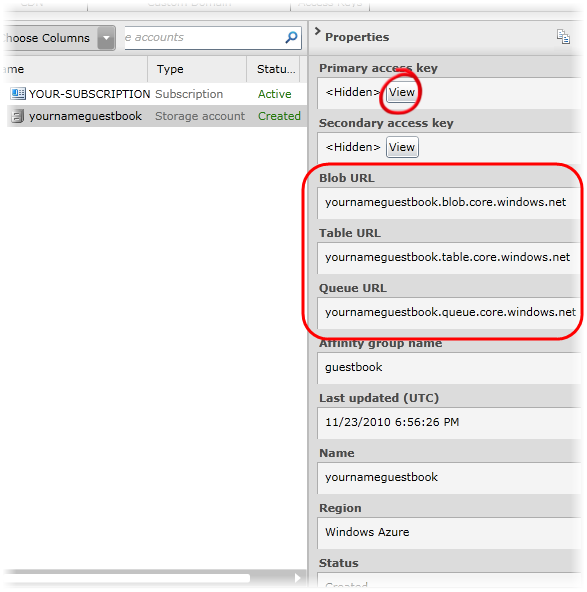
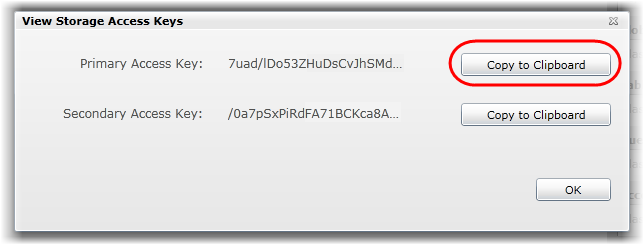
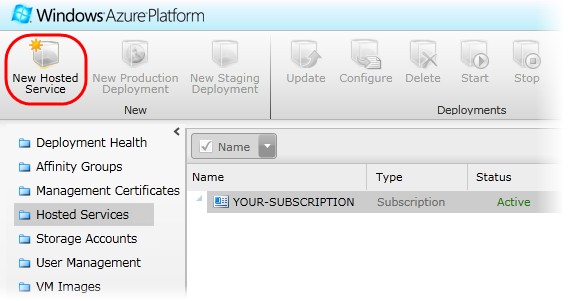
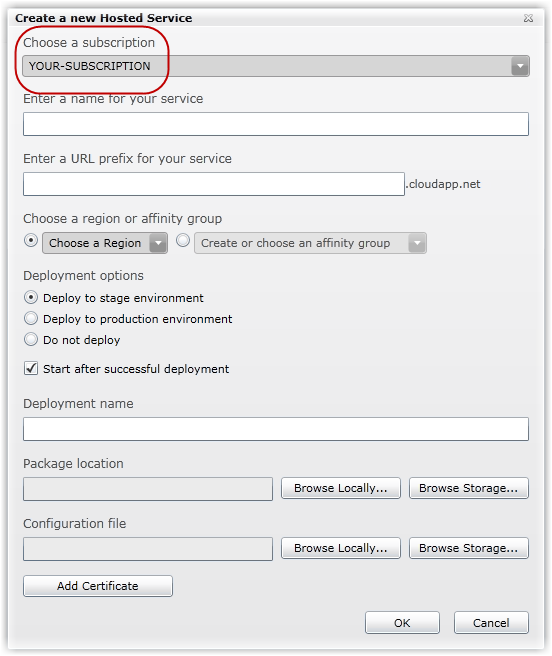
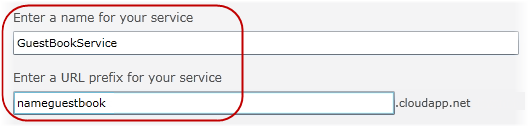
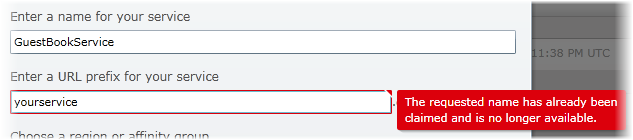
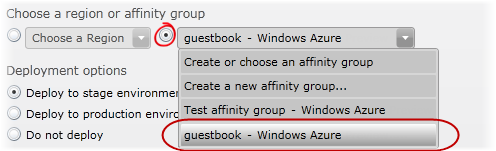
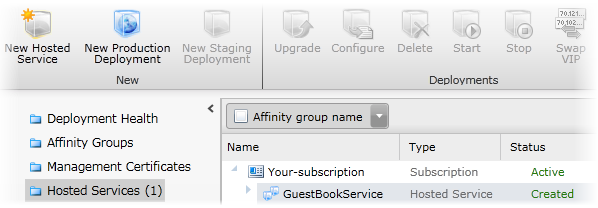
Verification

* 1. You now launch the updated application in the Windows Azure compute emulator to verify that the worker role can retrieve queued work items and generate the corresponding thumbnails.
  2. Press **F5** to launch the service in the local compute emulator.
  3. Switch to Internet Explorer to view the application. Provided you completed the verification section of the previous exercise successfully, you will see the guest book entry that you entered, including the uploaded image displayed in its original size. If you recall, during the last task of that exercise, you updated the web role code to post a work item to a queue for each new entry submitted. These messages remain in the queue even though the web role was subsequently recycled.
  4. Wait a few seconds until the worker role picks up the queued message and processes the image that you are viewing. Once that occurs, it generates a thumbnail for this image and updates the corresponding URL property for the entry in table storage. Eventually, because the page refreshes every few seconds, it will show the thumbnail image instead.
     1. 
     2. Figure 25
     3. Home page showing the thumbnail generated by the worker role
  5. If you are using Visual Studio 2010, in **Server Explorer**, expand the **Blobs** node in the **Windows Azure Storage** node, and then double-click the *guestbookpics* container. Notice that it now contains an additional blob for the generated thumbnail image.
     1. 
     2. Figure 26
     3. Blob container showing the blob for the generated thumbnail
  6. Add some more guest book entries. Notice that the images update after a few seconds once the worker role processes the thumbnails.
  7. Press **SHIFT + F5** to stop the debugger and shut down the deployment in the compute emulator.

Exercise 3: Publishing a Windows Azure Application

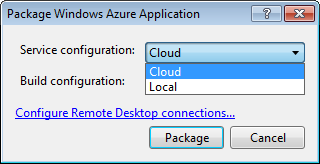
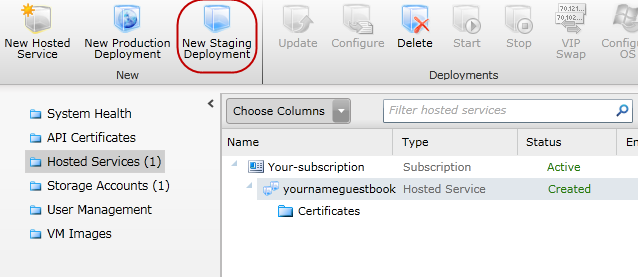
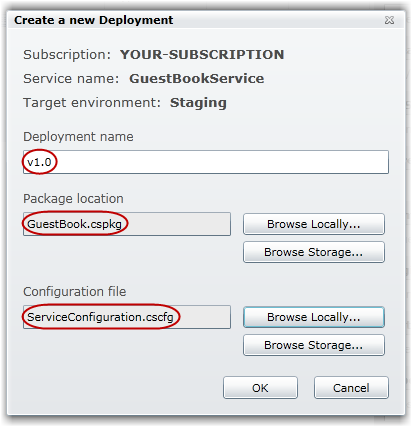
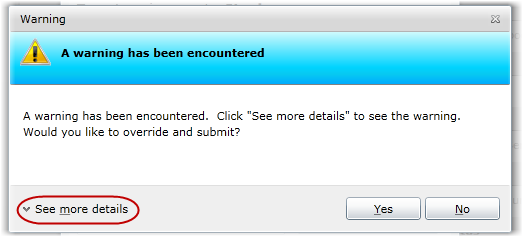
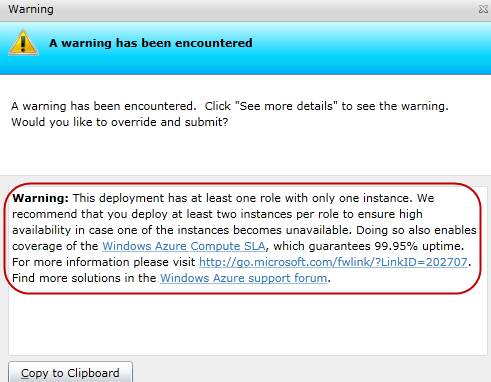
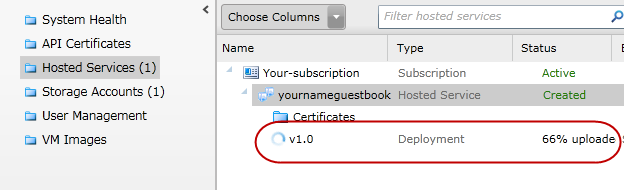
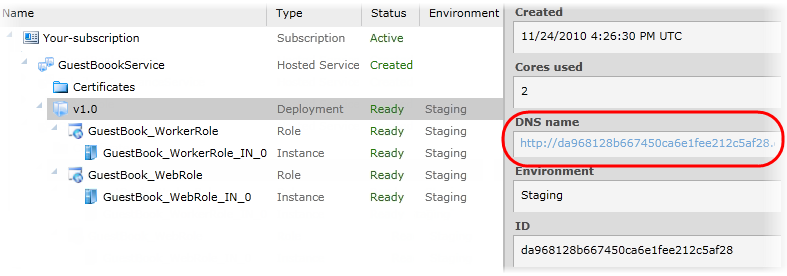
* 1. In this exercise, you publish the application created in the previous exercise to Windows Azure using the Management Portal. First, you provision the required service components, upload the application package to the staging area and configure it. You then execute the application in the staging area to verify its operation. Finally, you promote the application to production.
  2. **Note:** In order to complete this exercise, you need to sign up for a Windows Azure account and purchase a subscription.
  3. For a description of the provisioning process, see [Provisioning Windows Azure](http://blogs.msdn.com/david_sayed/archive/2010/01/07/provisioning-windows-azure.aspx).

Task 1 – Creating a Storage Account and a Hosted Service Component

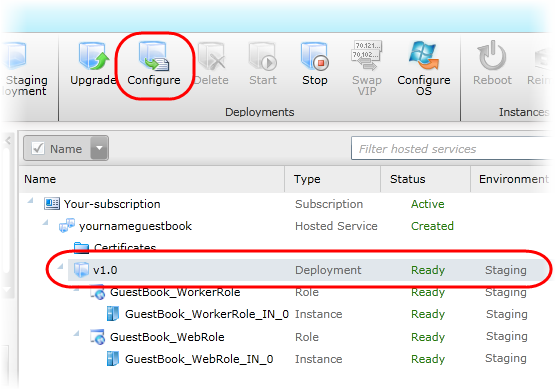
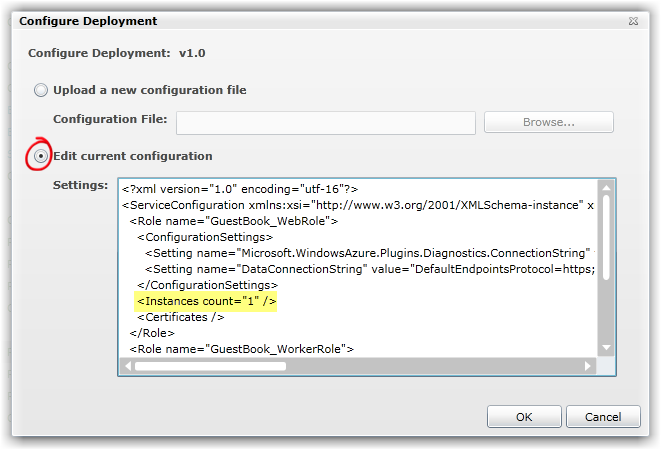
* 1. The application you publish in this exercise requires both compute and storage services. In this task, you create a new Windows Azure storage account to allow the application to persist its data. In addition, you define a hosted service component to execute application code.
  2. Navigate to [http://windows.azure.com](http://windows.azure.com/) using a Web browser and sign in using the Windows Live ID associated with your Windows Azure account.
     1. 
     2. Figure 27
     3. Signing in to the Windows Azure Platform Management Portal
  3. First, you create the storage account that the application will use to store its data. In the Windows Azure ribbon, click **New Storage Account**.
     1. 
     2. Figure 28
     3. Creating a new storage account
  4. In the **Create a New Storage Account** dialog, pick your subscription in the drop down list labeled **Choose a subscription**.
     1. 
     2. Figure 29
     3. Choosing a subscription to host the storage account
  5. In the textbox labeled **Enter a URL**, enter the name for your storage account, for example, ***<yourname>guestbook***, where *<yourname>* is a unique name. Windows Azure uses this value to generate the endpoint URLs for the storage account services.
     1. 
     2. Figure 30
     3. Choosing the URL of the new storage account
     4. **Note:** The portal ensures that the name is valid by verifying that the name complies with the naming rules and is currently available. A validation error will be shown if you enter a name that does not satisfy the rules.
     5. 
  6. Select the option labeled **Create or choose an affinity group** and then pick **Create a new affinity group** from the drop down list.
     1. 
     2. Figure 31
     3. Creating a new affinity group
     4. **Note:** The reason that you are creating a new affinity group is to deploy both the hosted service and storage account to the same location, thus ensuring high bandwidth and low latency between the application and the data it depends on.
  7. In the **Create a New Affinity Group** dialog, enter an **Affinity Group Name**, select its **Location** in the drop down list, and then click **OK**.
     1. 
     2. Figure 32
     3. Creating a new affinity group
  8. Back in the **Create a New Storage Account** dialog, click **Create** to register your new storage account. Wait until the account provisioning process completes and updates the **Storage Accounts** tree view. Notice that the **Properties** pane shows the **URL** assigned to each service in the storage account. Record the public storage account name—this is the first segment of the URL assigned to your endpoints.
     1. 
     2. Figure 33
     3. Storage account successfully created
  9. Now, click the **View** button next to **Primary access key** in the **Properties** pane. In the **View Storage Access Keys** dialog, click **Copy to Clipboard** next to the **Primary Access Key**. You will use this value later on to configure the application.
     1. 
     2. Figure 34
     3. Retrieving the storage access keys
     4. **Note:** The **Primary Access Key** and **Secondary** Access **Key** both provide a shared secret that you can use to access storage. The secondary key gives the same access as the primary key and is used for backup purposes. You can regenerate each key independently in case either one is compromised.
  10. Next, create the compute component that executes the application code. Click **Hosted Services** on the left pane. Click on **New Hosted Service** button on the ribbon.
      1. 
      2. Figure 35
      3. Creating a new hosted service
  11. 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**.
      1. 
      2. Figure 36
      3. Choosing your subscription
  12. 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**, for example, ***<yourname>guestbook***, where *<yourname>* is a unique name. Windows Azure uses this value to generate the endpoint URLs for the hosted service.
      1. 
      2. Figure 37
      3. Choosing a service name and URL
      4. **Note:** If possible, choose the same name for both the storage account and hosted service. However, you may need to choose a different name if the one you select is unavailable.
      5. **Note:** The portal ensures that the name is valid by verifying that the name complies with the naming rules and is currently available. A validation error will be shown if you enter name that does not satisfy the rules.
      6. 
  13. Select the option labeled **Create or choose an affinity group** and then pick the **guestbook** affinity group from the drop down list—this is the same affinity group that you defined earlier, when you created the storage account.
      1. 
      2. Figure 38
      3. Choosing an affinity group
      4. **Note:** By choosing **guestbook** as the affinity group, you ensure that the hosted service is deployed to the same location as the storage account that you provisioned earlier.
  14. 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.
  15. Click **OK** to create the hosted service and then wait until the provisioning process completes.
      1. 
      2. Figure 39
      3. Hosted service successfully created
  16. Do not close the browser window. You will use the portal for the next task.



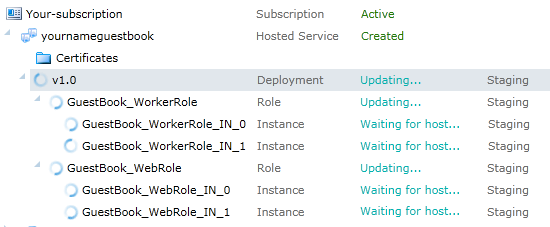
Task 2 – Publishing the Application to the Windows Azure Platform Management Portal

* 1. There are several alternatives for publishing applications to Windows Azure. The Windows Azure Tools for Visual Studio allow you to both create and publish the service package to the Windows Azure environment directly from Visual Studio. Another deployment option is the [Windows Azure Service Management PowerShell Cmdlets](http://code.msdn.microsoft.com/azurecmdlets) that enable a scripted deployment of your application. Lastly, the Windows Azure Platform Management Portal provides the means to publish and manage your service using only your browser. For more information about publishing applications, see the **Windows Azure Deployment** lab in this training kit.
  2. In this task, you publish the application to the staging environment using the Management Portal but first, you generate the service package using Visual Studio.
  3. If it is not already open, launch Visual Studioas administrator from **Start | All Programs | Microsoft Visual Studio 2010** by right clicking the **Microsoft Visual Studio 2010** shortcut and choosing **Run as administrator**.
  4. If the **User Account Control** dialog appears, click **Continue**.
  5. In the **File** menu, choose **Open** and then **Project/Solution**. In the **Open Project** dialog, browse to **Ex3-WindowsAzureDeployment** in the **Source** folderof the lab and choose the folder for the language of your preference (Visual C# or Visual Basic). Select **Begin.sln** in the **Begin** folder and click **Open**. Alternatively, you may continue with the solution that you obtained after completing the previous exercise.
  6. To configure the storage before publishing the service, open the **ServiceConfiguration.cscfg** file you will use to publish the project (e.g. **ServiceConfiguration.Cloud.cscfg**), which is located in **GuestBook** service. Replace the placeholder labeled [YOUR\_ACCOUNT\_NAME] with the **Storage Account Name** that you chose when you configured the storage account in Task 1. If you followed the recommendation, the name should follow the pattern ***<yourname>guestbook***, where *<yourname>* is a unique name. Make sure to replace both instances of the placeholder, one for the *DataConnectionString* and the second one for the *Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString*.
  7. Next, replace the placeholder labeled [YOUR\_ACCOUNT\_KEY] with the **Primary Access Key** value that you recorded earlier, when you created the storage account in Task 1. Again, replace both instances of the placeholder, one for each connection string.
     1. 
     2. Figure 40
     3. Configuring the storage account connection strings
  8. Generate the package to publish to the cloud. To do this, right-click the **GuestBook** cloud project and select **Package**. In the **Package Windows Azure Application** dialog, select the **Service Configuration** and the **Build Configuration** you will use from the dropdowns and then click **Package**.
     1. After Visual Studio builds the project and generates the service package, Windows Explorer opens with the current folder set to the location where the generated package is stored.
     2. 
     3. Figure 41
     4. Creating a service package in Visual Studio
     5. **Note:** Although the procedure is not shown here, you can use the Publish Cloud Service feature in the Windows Azure Tools to publish your service package directly from Visual Studio. To use this feature, you need to configure a set of credentials that you use to authenticate access to the management service using a self-issued certificate that you upload to the Management Portal.
  9. Now, switch back to the browser window and the Management Portal.
  10. At the portal, select the hosted service that you created in the previous step and then click **New Staging Deployment** on the ribbon.
      1. **Note:** A hosted service is a service that runs your code in the Windows Azure environment. It has two separate deployment slots: staging and production. The staging deployment slot allows you to test your service in the Windows Azure environment before you deploy it to production.
      2. 
      3. Figure 42
      4. Hosted service summary page
  11. In the **Create a new Deployment** dialog, to select a **Package location**, click **Browse Locally**, navigate to the folder where Visual Studio generated the package in Step 4 and then select **GuestBook.cspkg**.
  12. Now, to choose the **Configuration File**, click **Browse Locally** and select **ServiceConfiguration.cscfg** in the same folder that you used in the previous step.
      1. **Note:** The *.cscfg* file contains configuration settings for the application, including the instance count that you will update later in the exercise.
  13. Finally, for the **Deployment name**, enter a label to identify the deployment; for example, use **v1.0**.
      1. **Note:** The portal displays the label in its user interface for staging and production, allowing you to identify the version currently deployed in each environment.
      2. 
      3. Figure 43
      4. Configuring the service package deployment
  14. Click **OK** to start the deployment. Notice that the portal displays a warning message when you do this. Click **See more details** to review and understand the message.
      1. 
      2. Figure 44
      3. Reviewing the warnings
      4. **Note:** In this particular case, the warning indicates that only a single instance is being deployed for at least one of the roles. This is not recommended because it does not guarantee the service’s availability. In the next task, you will increase the number of instances to overcome this issue.
      5. 
  15. Click **Yes** to override and submit the deployment request. Notice that the package begins to upload and that the portal shows the status of the deployment to indicate its progress.
      1. 
      2. Figure 45
      3. Uploading a service package to the Windows Azure Platform Management Portal
  16. Wait until the deployment process finishes, which may take several minutes. At this point, you have already uploaded the package and it is in a **Ready** state. Notice that the portal assigned a **DNS name** to the deployment that includes a unique identifier. Shortly, you will access this URL to test the application and determine whether it operates correctly in the Windows Azure environment, but first you need to configure it.
      1. **Note:** During deployment, Windows Azure analyzes the configuration file and copies the service to the correct number of machines, and starts all the instances. Load balancers, network devices and monitoring are also configured during this time.
      2. 
      3. Figure 46
      4. Package successfully deployed and ready

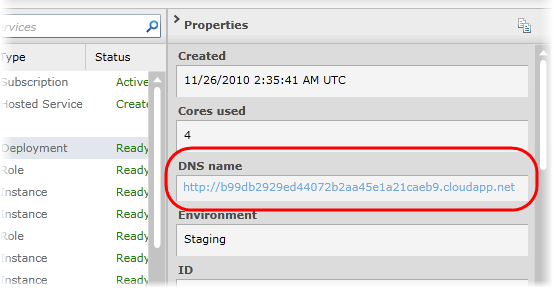
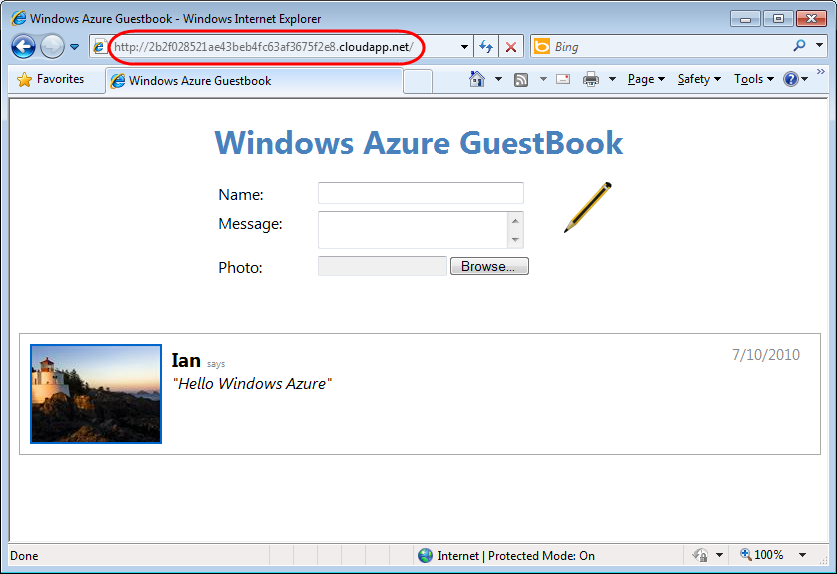
Task 3 – Configuring the Application to Increase the Number of Instances

* 1. Before you can test the deployed application, you need to configure it. In this task, you define the storage account settings for the application.
  2. In **Hosted Services**, select your GuestBook service and click **Configure** on the ribbon.
     1. 
     2. Figure 47
     3. Configuring application settings
  3. In the **Configure Deployment** dialog, select the option labeled **Edit current configuration**, locate the **Instances** element inside the **GuestBook\_WebRole** configuration and change its **count** attribute to *2*. Do the same for the **GuestBook\_WorkerRole** *c*onfiguration to also increase its instance count to *2*.
     1. 
     2. Figure 48
     3. Configuring the Instances count

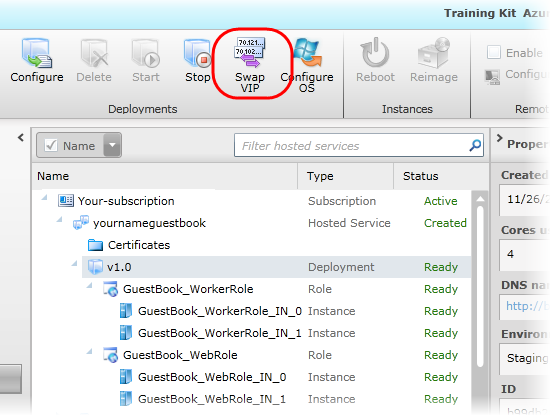
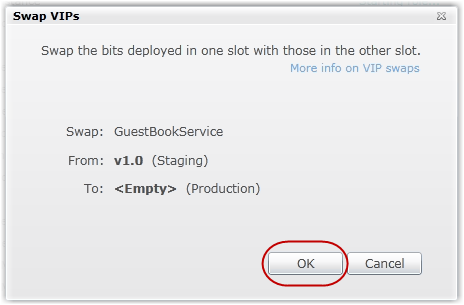
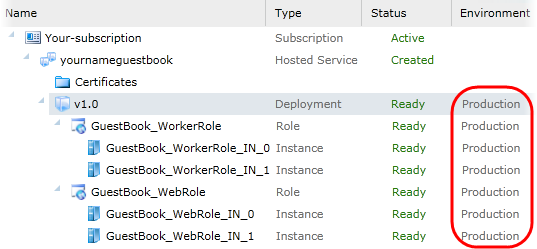
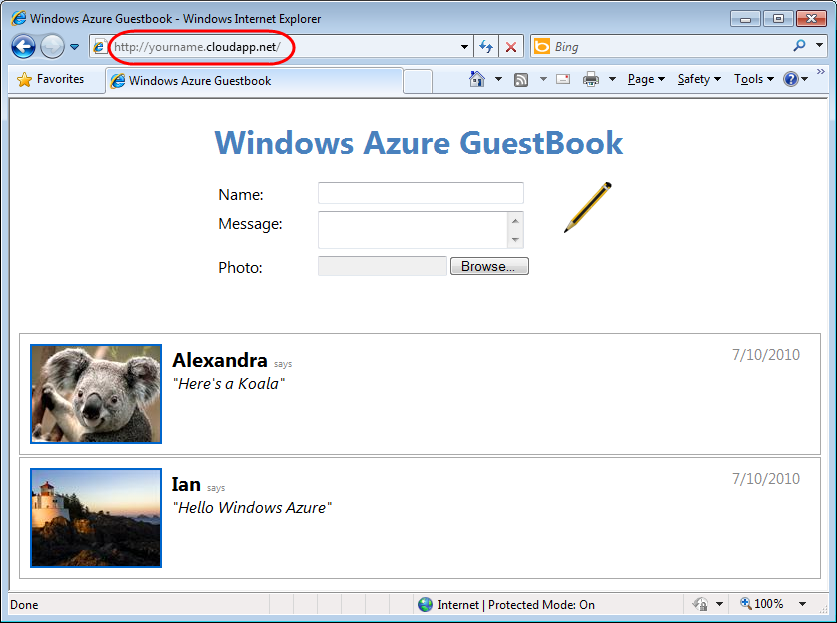
**Note:** The configuration is simply an XML document that contains the value of the settings declared in the service definition file. Its initial content is determined by the **ServiceConfiguration.cscfg** file that you uploaded earlier, when you deployed the package in Task 2.

* + 1. **Note:** The **Instances** setting controls the number of roles that Windows Azure starts and is used to scale the service. For a token-based subscription—currently only available in countries that are not provisioned for billing—this number is limited to a maximum of two instances. However, in the commercial offering, you can change it to any number that you are willing to pay for.
  1. Click **OK** to update the configuration and wait for the hosted service to apply the new settings.
     1. 
     2. Figure 49
     3. Updating the number of role instances
     4. **Note:** The portal displays a legend "Updating deployment..." while the settings are applied.

Task 4 – Testing the Application in the Staging Environment

* 1. In this task, you run the application in the staging environment and access its public endpoint to test that it operates correctly.
  2. In **Hosted Services**, select your GuestBook service and then click the link located in the right pane under **DNS name.**
     1. 
     2. Figure 50
     3. Running the application in the staging environment
     4. **Note:** The link shown for DNS name has the form *<guid>.cloudapp.net*, where *<guid>* is some random identifier. This is different from the address where the application will run once it is in production. Although the application executes in a staging area that is separate from the production environment, there is no actual physical difference between staging and production – it is simply a matter of where the load balancer is connected.
  3. If you wish, you may test the application by signing the guest book and uploading an image.
     1. 
     2. Figure 51
     3. Application running in the staging environment

Task 5 – Promoting the Application to Production

* 1. Now that you have verified that the service is working correctly in the staging environment, you are ready to promote it to final production. When you deploy the application to production, Windows Azure reconfigures its load balancers so that the application is available at its production URL.
  2. In **Hosted Services**, select your GuestBook service and then click **Swap VIP** on the ribbon.
     1. 
     2. Figure 52
     3. Promoting the application to the production slot
  3. On the **Swap VIPs** dialog, click **OK** to swap the deployments between staging and production.
     1. 
     2. Figure 53
     3. Promoting the application to the production deployment
  4. Wait for the promotion process to complete, which typically takes a few seconds.
     1. 
     2. Figure 54
     3. Application successfully deployed to production
  5. Click the **DNS name** link to open the production site in a browser window and notice the URL in the address bar.
     1. 
     2. Figure 55
     3. Application running in the production environment
     4. **Note:** If you visit the production site shortly after its promotion, the DNS name might not be ready. If you encounter a DNS error (404), wait a few minutes and try again. Keep in mind that Windows Azure creates DNS name entries dynamically and that the changes might take few minutes to propagate.
  6. Even when a deployment is in a suspended state, Windows Azure still needs to allocate a virtual machine (VM) for each instance and charge you for it. Once you have completed testing the application, you need to remove the deployment from Windows Azure to avoid an unnecessary expense. To remove a running deployment, go to **Hosted Services**, select the deployment slot where the service is currently hosted, staging or production, and then click **Stop** on the ribbon. Once the service has stopped, click **Delete** on the ribbon to remove it.

Summary

1. By completing this hands-on lab, you have reviewed the basic elements of Windows Azure applications. You have seen that services consist of one or more web roles and worker roles. You have learned about Windows Azure storage services and in particular, Blob, Table and Queue services. Finally, you have explored a basic architectural pattern for cloud applications that allows front-end processes to communicate with back-end processes using queues.